open source
PRESS

Dovecot

POP3/IMAP servers
for enterprises and ISPs

root's reading

Peer Heinlein

Dovecot

Table of Contents

2. Installation

3. POP3 and IMAP at protocol level

4. Introduction to the configuration

5. Authentication

6. How to configure Postfix as a relay in front of Dovecot

7. mbox, Maildir and mdbox: a comparison of storage formats

9. The IMAP namespace and shared folders

10. Setting up SSL and TLS

11. Quotas

12. Server-side email filtering with Sieve

12.1. Setting up Sieve

12.2. Out of office response via Sieve script

12.3. Converting old procmail scripts

12.4. Applying Sieve scripts to emails that have already been received

13. Email extensions

13.1. The recipient delimiter extends email addresses

13.2. Automatic saving of emails in IMAP folders

14. Daily maintenance of the email storage with doveadm

15. Performance tuning for more than 1000 simultaneous logins

15.1. Increasing the maximum number of clients

15.2. Reducing login processes

15.3. Increasing the file handles

15.3.1. Debian/Ubuntu

15.3.2. openSUSE/SLES

15.3.3. CentOS/RHEL
15.4. Push email and IDLE15.5. Using the write cache in the mail storage15.6. Single instance storage for attachments

16. Load-balancing cluster

17. Autoconfig and Autodiscover – automatic configuration of email clients

17.1. Autoconfig with Thunderbird

17.2. Autodiscover with Outlook

17.3. Automatic configuration for Mac OS, iOS and Windows LiveMail

17.4. Dynamic configuration scripts in PHP and Python

18. Webmailer as a complementary service

19. Migrating IMAP servers

Peer Heinlein

Dovecot

POP3/IMAP servers for enterprises and ISPs

Open Source Press

Peer Heinlein: Dovecot - POP3/IMAP-Server für Unternehmen und ISPs

Bibliographic information published by the Deutsche Nationalbibliothek

The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at http://dnb.dnb.de.

Editor: Markus Wirtz
Typesetting: textovia web application (textovia.com)
Graphic Designer: Olga Saborov
ISBN: 9783955391102 (E-Book Mobi) http://www.opensourcepress.de

Foreword by Timo Sirainen

It's been about 11 years now since I first started Dovecot development. Back then, I was working for a small company where we needed an IMAP server for our emails. I was the only guy there with Linux experience, so besides doing the actual coding I was hired for, I was also the sysadmin. I'm not really sure what other people would have done in my situation, but I'm definitely not very good sysadmin material. I didn't start by evaluating features of different open source IMAP servers, I started by evaluating their source code. I didn't like any of them. I finally ended up with Courier, but I immediately decided I could do something better and started thinking about it. A few months later, I had the first public Dovecot release out.

Since then, it's been pretty interesting and fun. For a long time I was working on Dovecot just in my free time. Sometimes on company time too, on behalf of various businesses. Eventually, several companies hired me directly to develop some new features for them. Rackspace hired me to work on-site for a year and Portugal Telecom for half a year. It was fun seeing how people live outside of Finland.

Developing Dovecot has always been kind of fun but sometimes also very frustrating, especially before the v1.0 release. For years I thought that Dovecot was usable for most people, but then bigger and bigger installations started using it and they kept finding more and more problems. At one point, I realized that the entire design was completely wrong, so I had to spend a couple of years rewriting the entire indexing code. Luckily, this turned out to be one of the main reasons for Dovecot's good performance today.

Soon I started wondering what to do with the rest of my life after Portugal. Since I couldn't really think of anything else, I finally decided to start my own Dovecot support company. I had been thinking about it already for over 5 years at least, but I didn't have the right people to help manage and push me into it. Luckily, an old friend of mine was just thinking about moving into new things, and he decided it was the perfect time to start the Dovecot company with me.

A couple of people were really upset that Dovecot was now becoming commercialized and that I should never write anything about my company to the Dovecot mailing list, but the majority seemed to be happy enough. One of my main goals with the company has always been to think about Dovecot's future. As I once mentioned in a customer meeting, I've only had one baby so far, and he's a 10-year-old named Dovecot. I wouldn't give him up for any amount of money. I want him to grow up old and take over the world. So the Dovecot company's job is to make sure that it happens, and this requires a way to make money and hire more Dovecot developers. Our original idea of providing support and consulting around Dovecot didn't start out too badly, but we later noticed it could never be enough to sustain the

quick growth and improvements that we foresaw for Dovecot. We had to figure out a few other incentives for companies to give us money, which was the start of a few closed source components to Dovecot.

This book and especially its timing marks an interesting turning point in Dovecot's life. It's not quite what it used to be, and hopefully it will become much better in the near future at a much faster pace. Peer first asked me to give a Dovecot talk for his 3rd mail server conference in 2007, but at that point in my life I really didn't want to give any Dovecot talks. But soon afterward I was more or less forced to do a few talks to some small crowds anyway, so I felt more comfortable giving my first public Dovecot talk at the next mail server conference.

Peer was already back then talking about just finishing his Cyrus/Courier/Postfix book and about adding Dovecot for the next one. It's now several years later, but I think it's the perfect time for this book to emerge!

Dovecot is still being developed at a fast pace, but there are no huge incompatible changes just around the corner, so this book should be useful for many more years to come.

Finally, I'd like to give thanks to everyone who has helped Dovecot become as awesome as it is today.

Timo Sirainen

January 2014, Helsinki/Finnland

Preface

I am happy to admit it: when I first heard 10 years ago that a new IMAP server called Dovecot was under development, I thought it was absolutely superfluous. There were Courier and Cyrus, both of which had their own areas of application, and it made no sense to me to introduce a third instead of developing Courier further. All the more as the first versions of Dovecot were very similar to Courier and were not able to cover the same range of functions. Then there were the usual launch difficulties and bugs, with the new software refusing to do what it is supposed to.

But soon (after a *very* short time) it became clear that Dovecot was catching up with Courier and Cyrus – and had even overtaken them. The scope of functions in Dovecot was far wider, the speed faster, the configuration clearer and more flexible, and the log file easier to read. In addition, Dovecot was RFC-conform, and the developers were introducing new features every week that brought joy to an admin's heart. In short, Dovecot is (and was) fun. And I know what I'm talking about, because I have already published an IMAP book about Cyrus and Courier that Dovecot has made obsolete. This book would like to be its worthy successor.[1]

For me there is no question about which IMAP server to use. Since version 2.x at the latest, Dovecot has been far superior to all the other candidates. Courier has no right to exist any more, and Cyrus is not much better. Dovecot versions 2.2.x provide options (in the cluster area, for example) that Cyrus admins would not dare to dream of.

According to measurements conducted in June 2013 by http://openemailsurvey.org/, Dovecot is far in front with more than 2.3 million installations and a diffusion rate of over 50%.

Most Cyrus administrators do not even know what they are missing. In my work as a consultant and trainer, I keep meeting people who defend Cyrus vehemently and are skeptical of Dovecot. As they learn a little more about Dovecot in their first hour of training, they start to get a little envious. After 2 hours, they start to get thoughtful – and after 3 hours, they get excited, but about Dovecot rather than Cyrus.

When the first day is over, even skeptics are usually determined to migrate to Dovecot – even though the truly interesting features have not even been discussed at that stage.

Dovecot is an outstanding IMAP server for all sizes.

- It is properly programmed and RFC-conform (Cyrus is not).
- It provides all the features you could want, such as quotas (Chapter 11), shared folders (Chapter 9) and Sieve filtering (Chapter 12).
- Its basic configuration allows plenty of performance tuning and improvements, so there

is no need to bludgeon load problems with hardware (Chapter 15).
- It contains a powerful command line tool, `doveadm`, that you can use to automate almost every tasks (Chapter 14 and Appendix A).
- In the Enterprise area, its special *mdbox* storage format provides a storage system that scales well even for millions of accounts (Section 7.3) but also allows the use of slower hard drives (Section 7.3.4) or even object storage (Section 20.1).
- It even allows active/active clusters to be implemented without great effort; with a little orientation you can create a stable and robust cluster system that you don't need to handle with kid gloves (Section 16.4).

By the way, the Debian distribution measures which packages are installed and how often – and the long-term charts show a clear trend: While Courier and Cyrus are on the way down, Dovecot is shooting upwareds.

I hope you will have the same experience in this book: that your interest (which obviously exists) in Dovecot will turn to excitement as you go through the chapters.

A powerful IMAP server like Dovecot contains many details that you need to know and consider. The flexibility of the IMAP protocol itself, which does not even define the namespace for IMAP folders, means there are plenty of ways to influence things, but of course these can also go wrong.

In the first years, Dovecot was easy to learn even without a book. But in the face of its current complexity, simply reading through the (very well) commented configuration and example files is no longer enough. Many details have an effect in different areas, so you need to keep an eye on the whole system whenever you make an adjustment. I keep encountering systems where a minor bad decision was made that had catastrophic effects on the whole a few years later.

This book wants to take you by the hand and lead you through the various subjects relevant to your (future? current?) IMAP server. Because everything is connected, the book is sorted by subjects and organizational matters rather than the technical structure of the software. That means that details are spread across the chapters, as they are discussed where they have an effect.

Our chapters describe how to perform tasks and resolve problems. This book is in some ways a complete work of reference, though the speed at which Dovecot is being developed has ensured that new features were released at the time of going to press and could therefore not be included. It is not a technical reference; that is what the Dovecot Wiki[2] and the man pages of the various tools are for, and they will always be more up-to-date than a book.

Once you have understood the concept, structure and procedure of Dovecot with the help of this book, everything else will be easy and quick to understand, and smaller updates to the tools do not create any difficulties in comprehension. Nevertheless, you will find technical

references in the appendix of this book; cross references will take you quickly to the relevant explanations in the book.

As with my Postfix book, there is a mailing list for Dovecot where readers and Dovecot admins discuss questions and help one another. The mailing list will also ensure that you are always aware of the latest developments. You can find the details and join the mailing list at http://www.dovecot-book.com. There you will find updates, corrections and other notes on the chapters of the book edition.

I also welcome feedback on the book, which you can send to <p.heinlein@heinlein-support.de>. Maybe we will even meet in person at a trade fair, conference or other event. Or you could drop in at our Berlin Linux Academy for a Postfix, Dovecot or other admin training course. And if you want to take advantage of our experience for a concrete project, my team and I will be happy to help on site: http://www.heinlein-support.de

My thanks go to the Heinlein Support team: after all, we have been running IMAP clusters for many years, which is where we tried and learned many of the things that are now found in this book as experience and best practice. All admins from Heinlein Hosting, colleagues from Heinlein Consulting and of course the good spirits in all the other areas have a direct or indirect share in this book. Thank you for that. I particularly want to thank Peer Hartleben, who has supported me at my company for 10 years and who co-authored the old IMAP book from which some text passages have been included in this book.

As in almost every book project, my children and wife were the ones who suffered most as *the book* turned into a fifth member of the family for 15 months, always there at the lunch table and stealing my free time. I hope it was worth it, and that you benefit from reading it.

So, have fun with your keyboard,

Peer Heinlein, February 2014

---

[1] "The book of IMAP – building a mail server with Courier and Cyrus", Open Source Press/No Starch Press 2007.

[2] http://wiki2.dovecot.org

Chapter 1. Protocols and Terms

What is a mail server? People often imagine a machine in some data processing center that takes care of email administration. In fact, a mail server is modular, consisting of a variety of components and programs, and communicates using various protocols. The same is true for commercial solutions that provide this variety under *one* product name. In most cases, the various tasks involved can be distributed among multiple servers, which means “the mail server” is actually composed of multiple machines.

1.1. An Overview of Terms

Mail servers use the *Simple Mail Transport Protocol* (SMTP) to communicate with each other in order to deliver emails. Clients such as Outlook, KMail, Thunderbird, LiveMail oder Evolution usually deliver their e-mails via SMTP to the *relay server* responsible for them. However, SMTP is suitable only for *sending* emails, not for *receiving* them, so SMTP cannot be used to retrieve your inbox or create email directories in it. As a consequence, a *Mail Transfer Agent* (MTA) that transports emails received from clients or other servers to their destinations has nothing to do with reception mechanisms such as POP3, IMAP or similar. In short: this book will describe SMTP servers such as Postfix, QMail, Exim or Sendmail only where Dovecot and the MTA come into contact. We will analyze the best way to receive and route Dovecot domains, and how to map the user list and the quota settings. Advanced knowledge of MTA, and the whole area of spam and antivirus software, are part of this subject and have already been extensively documented.

Post Office Protocol Version 3 (POP3) is a comparatively simple protocol with few configuration options, so pure POP3 servers require very little administration.

The *Internet Message Access Protocol* (IMAP), the big brother of POP3, is quite complex. This book will focus mainly on the numerous ways to influence the email retrieval process and and to administrate emails. When you have finished reading it, you will be able to implement challenging scenarios. Well-known IMAP servers always provide a POP3 daemon as well; for programmers dealing with the complexity of the IMAP protocol, there is no difficulty in offering the POP3 protocol as an addition from the same data base.

This book is all about Dovecot, the most powerful and modern open-source IMAP server. As Dovecot also contains a POP3 server, this book will often be referring to both services, even if we use the term IMAP server for the purpose of convenience. To be precise, Dovecot is a POP3, IMAP, LMTP, Managesieve, SASL and Postfix-Policy-Daemon server – with an SMTP server planned as well. That makes Dovecot a real all-rounder, as we will see again and again throughout this book.

LMTP, the *Local Message Transfer Protocol*, is closely related to SMTP, but is designed only for local use, such as transferring emails from one Mail Transfer Agent (such as Postfix) to other email components, specifically the *Mail Delivery Agent* (MDA). In this case it has an advantage over SMTP in that it is easier to determine the email addresses where local delivery was possible, and the email addresses where it wasn't. Unlike SMTP, *after* the *DATA* command LMTP outputs an individual status message for each recipient stating whose inbox the email was actually saved in. SMTP, by contrast, can only return one single status message at this point. If an email cannot be saved successfully in an inbox, the email must be rejected by the server for all recipients.

LMTP should be used only locally within a network, for example for delivery from an SMTP front relay (e.g. Postfix) to the actual email backend (Dovecot, Cyrus, Exchange) that is responsible for saving the email.

Groupware is software that manages tasks, calendars, email contacts and address books for multiple users. Some versions also include resource and room management, file management and other areas. Strictly speaking, Groupware is not really related to emails, but such software solutions usually contain their own email service, or just a webmailer that uses an IMAP server that is already running.

1.2. What makes IMAP so complex?

The POP3 server waits until a user has logged in successfully, and then transmits the unread messages saved in the user's inbox. Depending on the client request, transmitted messages are then either deleted to save space or marked as read and kept. This is not particularly challenging, so software cannot really go wrong here.

An IMAP server is rather different: instead of just delivering emails to users, it organizes all of the email administration. The user's mail client works as a type of "remote" for the inbox kept on the server.

An IMAP server makes the memory available and stores all emails, so it may be appropriate to set up quotas to force users to clear up from time to time and free up valuable memory. When a user creates a folder to sort his emails, the IMAP server has to map this folder structure and sort the emails accordingly. It is possible to search for senders or text blocks on the server, and IMAP also allows users to flag emails as read, unread or answered, and even create their own key words to flag emails. Shared folders accessed by several email users in parallel are another useful function.

An IMAP server allows users to maintain their inbox from multiple different computers; wherever they access their emails, they will find exactly the same data, rendering synchronization unnecessary.

All this is highly challenging for the IMAP protocol and therefore the programmers. For administrators, the effort required to configure IMAP servers is limited as soon as they have been connected to a user database. However, they offer plenty of decisions that can take you down a good or bad path.

Operating IMAP servers involves plenty of traps and technical difficulties that this book will describe in detail:

Performance
: As user numbers increase, the load on the IMAP server becomes apparent. It has to manage millions or billions of emails, operate thousands or hundreds of thousands of IMAP connections in parallel, and deal with search operations in emails or extensive copying. Depending on the scenario, consumption of file handles by IMAP servers can be very heavy and create a high I/O load on the data carriers.

Availability
: Today, email must be available around the clock as any failure can be damaging to business. Even with a robust IMAP server, any infrastructure above a certain size should ensure high availability in some form or another.

Storage

An email storage can grow to a considerable size. Even though it is easy today to implement multiple TByte hard disks in 1HE servers, IMAP servers require other storage concepts at ISP level, such as NAS, SAN or even object storage such as Ceph/Rados, Scality or S3.

Quotas

Here, again, some points should be considered during implementation so the server can be operated smoothly and requires no support.

Legal situation

Emails are subject to the sanctity of the mail; not many people know that negligent mistakes by the administrator may be sanctioned under criminal law. However, this is a general problem with mail servers, so this book will not discuss the problem in more detail.

Raise the curtain, clear the stage, and let's finally start installing and configuring Dovecot!

Part I. Dovecot for all purposes

Chapter 2. Installation

A variety of sources provide ready-made installation packages for Dovecot, including "daily builds" that allow you to install the latest version. I would advise you not to compile Dovecot yourself unless you want to create unnecessary work for yourself.

For a long time, Dovecot version branch 1.x ruled the world – Debian and Ubuntu distributions as well as older versions of Enterprise (SLES/RHEL) sometimes still contain versions from this series. But you won't want to install this branch any more, because versions from 2.0 onwards contain many relevant improvements that you do not want to do without.

At the time of going to press, the current version branches are 2.1.x and 2.2.x. They do not differ greatly from one another, so you can choose either. What is particularly interesting about the 2.2 branch is the overhauled *dsync* replication, which allows active/active clusters to be set up with real-time replication. Choose 2.2.x if you can. If your distribution only offers 2.1.x, you'd rather not fall back on third-party sources and have no special cluster requirements, 2.1 will do fine, and you can easily upgrade later.

However, for larger set-ups you should use a version from 2.0.16, where Dovecot author Timo Sirainen included the important *auto:* feature (see Section 7.5).

2.1. Installation under Debian/Ubuntu

If you want the latest versions, you should fall back on the backports platform.[3]

To integrate backports, you add the backport sources to `/etc/apt/sources.list`:

Debian

```
deb http://http.debian.org/debian wheezy-backports main
# deb http://http.debian.org/debian-backports squeeze-backports main
```

Ubuntu

```
deb http://archive.ubuntu.com/ubuntu lucid-backports main restricted u
```

If you are using a version other than *Squeeze*, *Wheezy* or *Lucid*, you will naturally need to change the name here and below.

Now execute the command `apt-get update` so your system learns about all new packages. In order to install a specific package from a backport, you have to add `apt-get` as a call parameter:

Debian

```
apt-get -t wheezy-backports install <package name>
```

Ubuntu

```
apt-get install <package name>/lucid-backports
```

Dovecot is separated into multiple individual packages in Debian/Ubuntu – you can see this using the `apt-cache search dovecot` command. First, install the following packages:

```
dovecot-core dovecot-common dovecot-imapd dovecot-pop3d dovecot-lmtpd
dovecot-managesieved dovecot-sieve
```

If you already know you will need to access SQL or LDAP, you should also install

```
dovecot-ldap dovecot-mysql dovecot-dovecot-pgsql
```

When we went to press, the official Debian Wheezy sources offered only version 2.1.7, and Ubuntu only 2.0.19 (Precise) or 2.1.9 (Lucid). Advanced configurations such as replicated clusters (Section 16.4) require version 2.2.x. As well as the official backports (Wheezy: 2.2.9), you can use the repository provided by Stephan Bosch; you need to take a little care, but he is an author involved in the Dovecot project and handles the maintenance of Sieve and Managesieve implementations (and generously proof-read the German version of this book).

In his repository he publishes hourly builds of the most recent version. Any bugs in the latest Dovecot releases turn up here immediately without verification as well, so automatic updates are not a sensible option in this case. But it is a great source for clean and up-to-date packages that allow you to set up and thoroughly test your server before it goes live.

If the versions from the official Debian backports are not adequate for you, you can integrate Stephan Bosch's repository like this:

Testing

```
deb http://xi.rename-it.nl/debian/ testing-auto/dovecot-2.2 main
```

Wheezy

```
deb http://xi.rename-it.nl/debian/ stable-auto/dovecot-2.2 main
```

Squeeze

```
deb http://xi.rename-it.nl/debian/ oldstable-auto/dovecot-2.2 main
```

Repositories are available for `dovecot-2.1` and `dovecot-2.0` as well as for `dovecot-2.2` (at least when we went to press).[4]

The packages provided by Stephan Bosch are signed, and you can install the keys like this

```
flash:~ # apt-get install debian-dovecot-auto-keyring
```

or

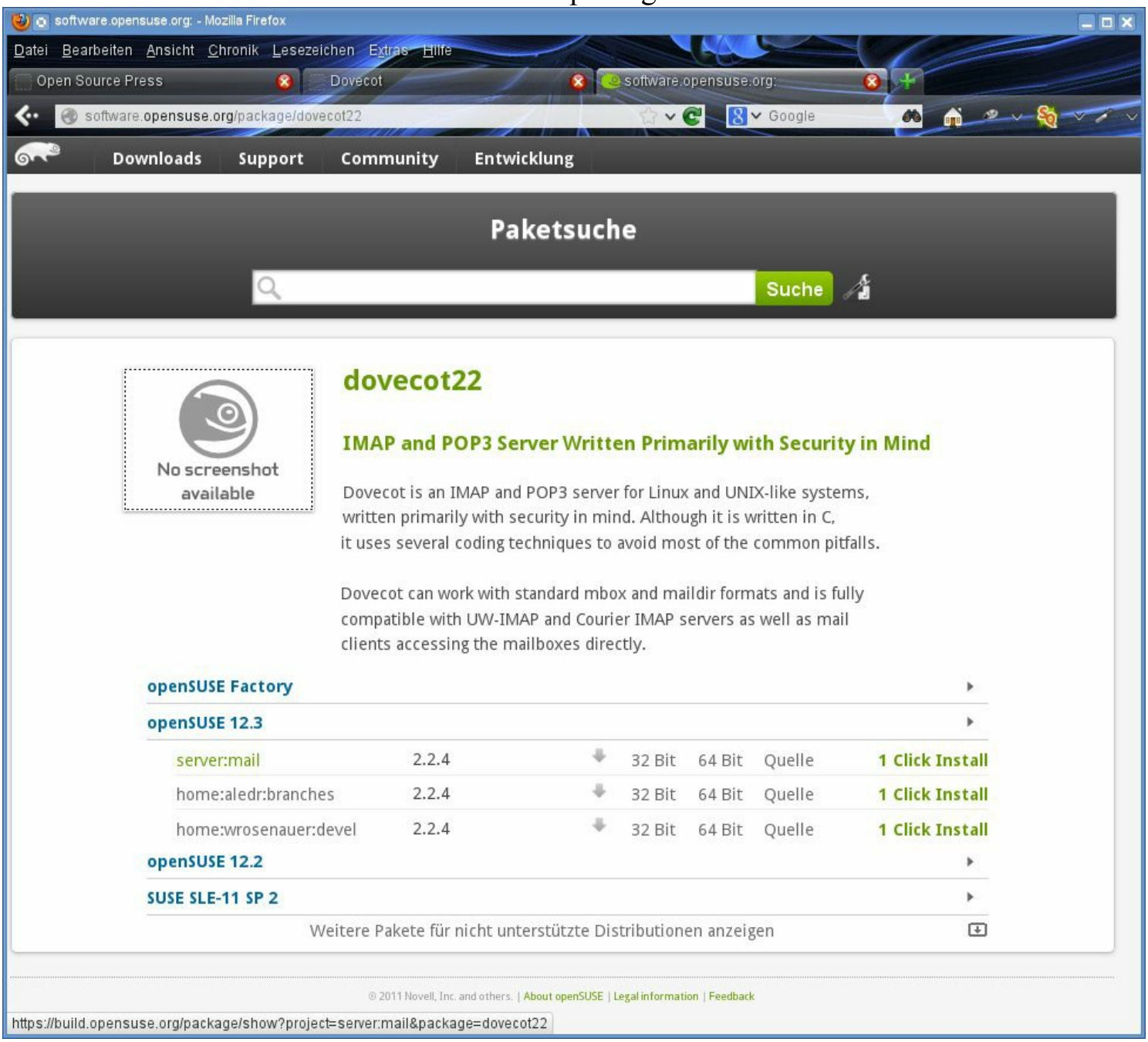
```
flash:~ # wget -O - http://xi.rename-it.nl/debian/archive.key | apt-key ad
```

2.2. Installation under openSUSE/SLES

One of the great things about SUSE distributions is that they always include pleasantly recent software. Check whether a version above 2.0.16, 2.1.x or even 2.2.x is available, and then proceed as usual.

For the most recent versions, you will find reliable packages in the SUSE-Build Service.[5]

Figure 2.1. In `server:mail` in the SUSE Build Service you will find reliable up-to-date Dovecot packages.



Look for `dovecot21` to find branch 2.1.x or (preferably) for `dovecot22` to find branch 2.2.x. Even if your search results are still marked as `unstable packages`, you can still access the

packages from the `server:mail` repository (see Figure 2.1). This repository is trustworthy, and its packages are subject to a certain amount of quality control.

You can add the new repository easily using the following command line:

```
flash:~ # zypper ar http://download.opensuse.org/repositories/server:/mail
```

Of course you need to change the URL at the end to match the version you need.

Unlike in Debian/Ubuntu, package `dovecot21` or `dovecot22` is enough – it already contains `pop3d`, `imapd` and the other daemons. If, however, you want to use MySQL or PostgreSQL, you will need to add

```
dovecot21-backend-mysql dovecot21-backend-pgsql
```

Support for LDAP is part of the main package.

2.3. Installation under CentOS/RHEL

CentOS 6.4 provides Dovecot, but it only offers a comparatively old version, 2.0.9, which does not contain the `auto:` feature described in Chapter 19 for operating different storage engines together.[6] You will find more recent packages on http://packages.atrpms.net/dist/.

Otherwise use package management to install Dovecot under CentOS. Support for Managesieve can be found in the `dovecot-pigeonhole` package. LDAP support is part of the main package; MySQL and PostgreSQL are in sub-packages `dovecot-mysql` and `dovecot-pgsql`.

2.4. Starting Dovecot for the first time

Start the Dovecot daemon and check whether everything is running properly. In the process list, Dovecot should look a bit like this:

```
flash:~ # ps ax | grep dovecot
14668 ?        Ss     0:00 /usr/sbin/dovecot
14669 ?        S      0:00 dovecot/anvil [2 connections]
14670 ?        S      0:00 dovecot/log
14672 ?        S      0:00 dovecot/config
flash:~ #
```

Ports 110 (POP3) and 143 (IMAP) should be open, in this case with IPv4 und IPv6:

```
flash:~ # lsof -i :110
COMMAND   PID USER   FD   TYPE  DEVICE SIZE/OFF NODE NAME
dovecot 14668 root   17u  IPv4 1080778      0t0  TCP *:pop3 (LISTEN)
dovecot 14668 root   18u  IPv6 1080779      0t0  TCP *:pop3 (LISTEN)

flash:~ # lsof -i :143
COMMAND     PID USER   FD   TYPE  DEVICE SIZE/OFF NODE NAME
dovecot   14668 root   24u  IPv4 1080789      0t0  TCP *:imap (LISTEN)
dovecot   14668 root   25u  IPv6 1080790      0t0  TCP *:imap (LISTEN)
```

And Dovecot should have greeted you in `/var/log/mail` (or `/var/log/mail.log`):

```
Dec 24 09:36:53 flash dovecot: master: Dovecot v2.0.16 starting up (core d
Dec 24 09:36:53 flash dovecot: ssl-params: Generating SSL parameters
Dec 24 9:36:54 AM flash dovecot: ssl-params: SSL parameters regeneration c
```

If you see something different, especially if port 110 or 143 is not open, you should check whether you have fully installed all packages.

2.5. The first login

In its default configuration, Dovecot will perform its user authentication against `/etc/passwd` and `/etc/shadow`.

You will find out how to integrate different authentication sources later on in Chapter 5, but authentication with local user accounts should work already.

Choose a local user account for testing purposes or quickly set up a test user:

```
flash:~ # useradd -m tux
flash:~ # passwd tux
Changing password for tux.
New Password: secret
Bad password: too simple
Reenter New Password: secret
Password changed.
```

Use the `doveadm` command (described in detail in Chapter 14) to check whether Dovecot recognizes the account and password properly. From Dovecot 2.2, you can do so like this:

```
flash:~ # doveadm auth test tux
Password:
passdb: tux auth succeeded
extra fields:
  user=tux

flash:~ # doveadm user tux
userdb: tux
  system_groups_user: tux
  uid       : 1003
  gid       : 100
  home      : /home/tux
```

In older versions, the phrase `doveadm auth test` is replaced with `doveadm auth`:

```
flash:~ # doveadm auth tux
Password:
passdb: tux auth succeeded
extra fields:
  user=tux
```

In case you are curious, you can also try to log in to POP3 and IMAP. Example for POP3:

```
flash:~ # telnet localhost 110
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
+OK Dovecot ready.
USER tux
```

```
+OK
PASS secret
+OK Logged in.
quit
+OK Logging out.
Connection closed by foreign host.
```

And for IMAP:

```
flash:~ # telnet localhost 143
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
* OK [CAPABILITY IMAP4rev1 LITERAL+ SASL-IR LOGIN-REFERRALS ID ENABLE IDLE
a login tux secret
a OK [CAPABILITY IMAP4rev1 LITERAL+ SASL-IR LOGIN-REFERRALS ID ENABLE IDLE
a logout
* BYE Logging out
a OK Logout completed.
Connection closed by foreign host.
```

We will take a close look at how exactly the protocols work in Chapter 3.

Your mail logfile should also contain traces of the connect:

```
Dec 25 08:25:38 flash dovecot: imap-login: Login: user=<tux>, method=PLAIN
Dec 25 08:25:47 flash dovecot: imap(tux): Disconnected: Logged out bytes=8
Dec 25 08:26:01 flash dovecot: pop3-login: Login: user=<tux>, method=PLAIN
Dec 25 08:26:03 flash dovecot: pop3(tux): Disconnected: Logged out top=0/0
```

If everything worked okay, this first quick test is enough. We will set up and describe everything step by step later on.

---

[3] http://backports-master.debian.org/Instructions/ provides detailed information on integrating backports.

[4] You will find the current status on http://wiki2.dovecot.org/PrebuiltBinaries.

[5] http://software.opensuse.org

[6] For details, see Section 7.5.

Chapter 3. POP3 and IMAP at protocol level

Whether you choose POP3 or IMAP as the protocol for fetching emails should depend on the intended purpose: one is simple and robust, the other is powerful and flexible. Dovecot speaks both protocols in parallel, so you can make a pragmatic decision on how your users should work.

Even though Dovecot is often referred to as the “IMAP server” for reasons of simplicity, that does not mean that Dovecot isn’t really a “POP3/IMAP server”.

3.1. POP3

Post Office Protocol version 3 (POP3) does not allow users to do much beyond loading emails from a server to a client: it logs in, views the list of contents, transfers emails, deletes emails and logs out, all using server port 110. The process requires comparatively few resources, and there is little to (mis)configure.

Emails are stored locally on the user's desktop PC; that saves expensive memory on the server and reduces its backup times. Users generally have to download all emails before deciding whether the subject and/or sender are interesting. However, most clients now support the use of filtering rules. Local storage allows users to edit their emails offline. This minimizes the time spent online and is suitable for those using a laptop on the go. However, if users access one mailbox from multiple computers, e.g. from a laptop and a desktop PC, POP3 provides no mechanisms allowing both email clients to work with identical data stocks.

The only choice a user can make in his client software is whether emails should remain on the server or be deleted after transmission. If the emails are left on the server, all other mail clients in use are able to download all emails as well. Good clients will recognize new messages as new ones by their stored message IDs, so emails will not be transmitted twice.

But this method also has some disadvantages: because the emails are not deleted, the mailbox on the server grows continuously. In addition, all emails are transmitted to all clients, even messages that the user has already deleted locally on another client. Nor is there a shared stock of all *sent* e-mails. As messages cannot be uploaded to the server, every email client has to store its sent folder locally.

3.1.1. Test-session

A POP session is easy to reproduce by hand:

```
peer@flash:~> telnet mail.example.com 110
Trying 192,168.50.50...
Connected to mail.example.com.
Escape character is '^]'.
+OK Dovecot ready.
USER tux
+OK
PASS secret
+OK Logged in.
```

We are automatically in the `INBOX` because other (IMAP) folders cannot be accessed using POP3. The `LIST` command provides an overview of all the messages saved there (nine in this example) and their respective length:

```
LIST
+OK 9 messages:
1 9586
2 1125022
3 53125
4 2451
5 5931
6 4943
7 4206
8 5231
9 9481
.
```

Use the `RETR` command to retrieve a message:

```
RETR 2
Return-Path: <p.heinlein@heinlein-support.de>
X-Original-To: p.heinlein@heinlein-support.de
Delivered-To: tux@example.com
Received: from [10.0.42.2] (unknown [10.0.42.2])
        (using TLSv1 with cipher DHE-RSA-AES256-SHA (256/256 bits))
        (Client did not present a certificate)
        by plasma.heinlein-support.de (Postfix) with ESMTP id BEA0581A4B
        for <tux@example.com>; Sat,  4 Jan 2014 01:02:01 +0200 (CEST)
From: Peer Heinlein <p.heinlein@heinlein-support.de>
To: Tux <tux@example.com>
Subject: Test message 2
Date: Sat, 4 Jan 2014 01:02:01 +0200
User-Agent: KMail/1.9.5
MIME-Version: 1.0
Content-Type: text/plain; charset="iso-8859-1"
Content-Transfer-Encoding: quoted-printable
Content-Disposition: inline
Message-Id: <201401040102.01895.p.heinlein@heinlein-support.de>
X-Length: 1519
```

```
Status: R
X-Status: NC
X-UID: 0

Hello!
I am a test message.

=2D-=20
Heinlein Support GmbH
Linux: Akademie - Support - Hosting

http://www.heinlein-support.de

Zwangsangaben lt. =A735a HGB:
HRB 93818 B / Amtsgericht Berlin-Charlottenburg,=20
Gesch=E4ftsf=FChrer: Peer Heinlein =A0-- Sitz: Berlin
```

It is just as easy to mark message 2 for deletion :

```
DELE 2
+OK Deleted.
```

But it is not actually deleted yet; that is done when the user logs out. That is why we can reverse the marking for deletion:

```
RSET
+OK Resurrected.
```

Instead of transferring an entire message, you can use the `Top` command to retrieve the full email header and the number of lines in the email body, which are specified as the second argument (`7` in this case):

```
TOP 2 7
Return-Path: <p.heinlein@heinlein-support.de>
X-Original-To: p.heinlein@heinlein-support.de
Delivered-To: tux@example.com
Received: from [10.0.42.2] (unknown [10.0.42.2])
        (using TLSv1 with cipher DHE-RSA-AES256-SHA (256/256 bits))
        (Client did not present a certificate)
        by plasma.heinlein-support.de (Postfix) with ESMTP id BEA0581A4B
        for <tux@example.com>; Sat,  4 Jan 2014 01:02:01 +0200 (CEST)
From: Peer Heinlein <p.heinlein@heinlein-support.de>
To: Tux <tux@example.com>
Subject: Test message 2
Date: Sat, 4 Jan 2014 01:02:01 +0200
User-Agent: KMail/1.9.5
MIME-Version: 1.0
Content-Type: text/plain; charset="iso-8859-1"
Content-Transfer-Encoding: quoted-printable
Content-Disposition: inline
Message-Id: <201401040102.01895.p.heinlein@heinlein-support.de>
X-Length: 1519
Status: R
X-Status: NC
```

```
X-UID: 0

Hello!
I am a test message.

=2D-=20
Heinlein Support GmbH
Linux: Akademie - Support - Hosting
.
```

There is an “idle” command that allows the client to keep the connection open:

```
NOOP
+OK Yup.
```

The connection is terminated using the `QUIT` command:

```
QUIT
+OK Bye-bye.
Connection closed by foreign host.
```

3.2. IMAP

The workings of *Internet Mail Access Protocol* Version 4 (IMAP), which the server offers on port 143, are completely different than those of POP3. IMAP is far more complex than POP3, and it supports sub-folders on the server (including folders that can be accessed by more than one user), complex searches and the upload of emails to the server.

In the original concept, emails are transferred to the client only when accessed and then stored there temporarily – almost like viewing a website. All emails remain on the server. That allows multiple email clients to access the same mailbox in parallel. The directory of the mailbox also allows the user to determine which emails should be transferred to the client.

IMAP is essentially designed to be operated online in real time. With slow dial-up connections, email retrieval can be slow due to latency.

Good email clients can perform a kind of "offline IMAP" and thereby combine the behavior of POP3 (local storage of emails) with that of IMAP (synchronizing the emails stored on the server). A user can place an IMAP mailbox on a laptop for offline use and then work with that mailbox while on a plane, for example.

Keeping data on the server requires valuable storage space. On the other hand, backing up the server[7] is enough to secure the emails of all users. If the hard drive on a client computer breaks, the user's emails are not affected (apart from outgoing messages if the user made only local copies.

3.2.1. Design of the IMAP-protocol

Unlike the SMTP and POP3 protocols, where you can write commands “by hand” for debugging purposes if you have to, IMAP is definitely not simple. But it can also do far more than the other email protocols.

IMAP itself rarely prescribes the status information (read, unread, important, new) an email has. The server and client can agree on their own additional *flags*. Next to permanent flags, there are also session-based ones (see also Section 7.2.3). Both types can be defined for each folder. That means the server and client have to agree on the flags they use and on the nature of those flags.

In IMAP, clients mainly work on the server’s data, so it contains search and selection functions for the client to use. That allows queries like “All unread emails AND from sender `geeko@example.com` AND message number above 300”.

Beyond that, IMAP also offers functions that allow clients to store many details locally and reuse them in more than one login – for example a locally stored table of contents. This is not trivial, because the email directory can change in the meantime. That is why IMAP numbers emails both with a *linearly increasing sequential number* without gaps, e.g. 1–399, and with a *unique ID (UID)* that is unique for every email and should never change. We use the word *should* because UIDs can actually change, so the client and server must identify and communicate about any modified UIDs.

That is why there is a *Unique Identifier Validity Value* that the server stores and transmits to the client along with the message. As long as the UID validity value remains unchanged, the client can assume that the assignment of UIDs to individual emails has not changed. The client then assigns the cached data to the emails on the server on the basis of the known UIDs. If, however, the server sets a new UID validity value, because of a software migration or an upload of a backup, the client must import all emails with all UIDs in full and rebuild the cache.

This complex design makes IMAP future-proof and flexible, because it allows new options and flags to be introduced without changes to the protocol.

Disconnected IMAP, also referred to as *offline IMAP*, is a good example: the email client stores all the data of the mailbox locally, the user can create folders or move, flag and delete emails offline – and the next time the client connects to the IMAP server, the client and server synchronize all changes. After some initial teething problems, this method now works surprisingly well and combines the advantages of POP3 and IMAP.

Another thing that makes IMAP sessions exciting and complex is that the client sends multiple commands to the server in parallel and can receive the answers in a different sequence. It is therefore possible to send a complex search request to the server and continue to upload new emails or create folders while waiting for the response.

All client requests are tagged with an unspecified but unique identifier. Our examples often use `a1`, `a2` or `a3`, but you could also use `abc`, `001` or `test` as tags. Responses from the server bear the same tag, so the client can still assign the responses even if they arrive a few minutes later and in a different order. It is important that the same tag is only used once at a specific time, because the server refers to the original command in its responses by means of the tag.

Once the server has responded to the command tagged `a1`, the next command can be tagged with `a1` again.

Such tags, `a1` in this case, must even be used login:

```
a1 login <username password>
a1 OK LOGIN OK
```

But not every server response is tagged: unfortunately, some server responses can contain more than one line. We therefore distinguish between tagged and untagged server responses. Untagged responses begin with an asterisk, while tagged responses bear the identifier chosen by the client and mark the end of the response.

In the case of the *NOOP* command, the server can send a multi-line response and untagged responses come into play. `NOOP` is used to perform *no* operation; clients send it to keep the connection open. However, some servers react by providing status information on the number of messages received since login (`13` in our next example) as well as the number of emails that have been read (`14`) or flagged for deletion (`22`):

```
a2 NOOP
* 22 EXPUNGE
* 23 EXISTS
* 13 RECENT
* 14 FETCH (FLAGS (\Seen \Deleted))
a2 OK NOOP completed
```

The server response consists of multiple untagged responses (introduced with `*`) and ends with `OK NOOP`, which is tagged `a2`.

3.2.2. Description of an IMAP session

A successful IMAP session begins when the server has greeted the client and then runs through four different states as shown in Figure 3.1:

Not Authenticated
: Typically the state between connection and successful authentication with user name and password. Only a few commands such as `STARTLS`, `LOGIN`, `LOGOUT` and `NOOP` are permitted.

Authenticated
: The client has been authenticated but has not yet selected a folder. It can obtain information on existing folders, subscribe or unsubscribe from them, create new ones or delete existing ones (see Section B.3).

  It is *not* possible to read or save emails in this state – after all, it is not clear which emails and directories are being referred to.

Selected
: The client has come out and selected an individual directory by means of `SELECT`, `EXAMINE` or `STATUS`. It can now place email commands that relate to emails in this directory. When the client deselects a directory by means of `CLOSE` or `EXPUNGE`, the connection returns to the Authenticated state. For an overview of all the commands available here, see Section B.4.

Logout
: The client has sent the `LOGOUT` command to the server. Only now does the server delete all the messages flagged as `\Deleted`. Should the connection be terminated in uncontrolled fashion, these emails should be retained. The server now says goodbye and severs the connection.

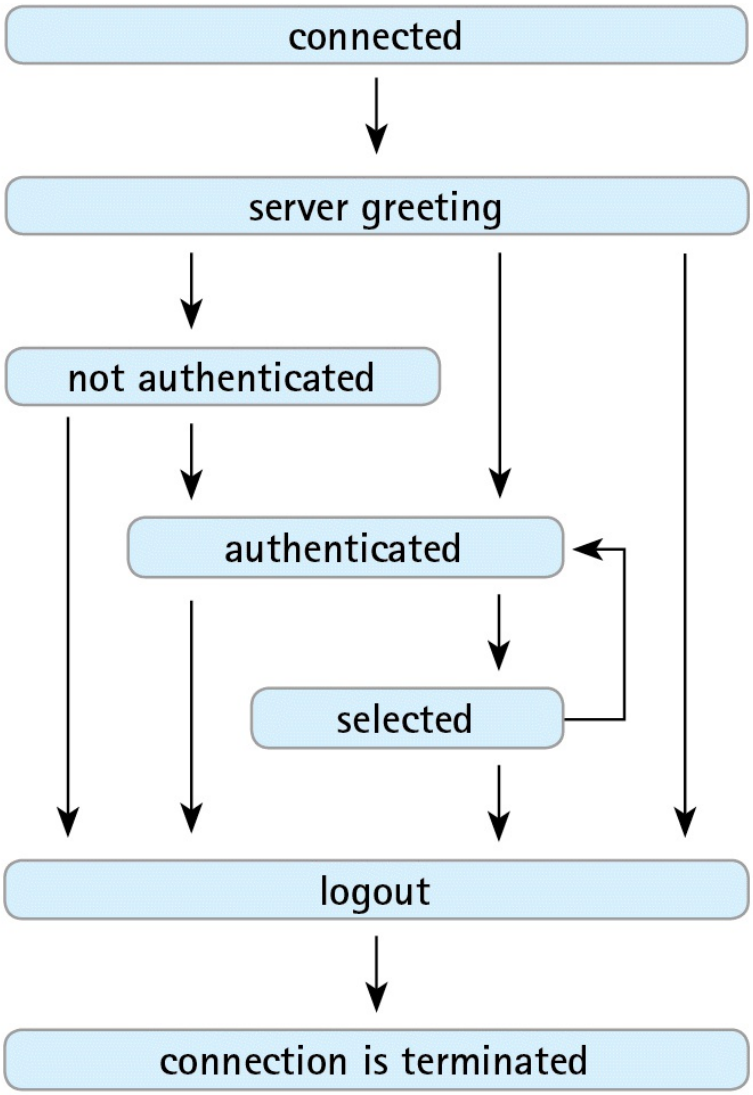
A normal IMAP connection always switches between “Authenticated” and “Selected”. It follows that not all commands are available at all times.

IMAP is therefore a session-based protocol comparable to FTP or SMTP. The opposite is HTTP, for example, where the client sends off each request individually and a request does not depend on the things that have already happened during this session. IMAP connections between the server and the client are usually kept open permanently, even for hours.

As a rule, clients check for new emails every few minutes, but the server can also inform the client of any new messages immediately. In PR and advertising, these are referred to as the pull and push procedures. From the server’s perspective, the push procedure is much more high-performance and desirable. The load involved in monitoring some directories for changes and informing the client is far lower than sending the client through any number of IMAP folders every few minutes just to find that nothing has changed. However, IMAP clients

now support the `IDLE` IMAP command, which allows clients to remain permanently connected to the server via TCP/IP and therefore be notified when a new email is received.

Figure 3.1. An IMAP connection is familiar with several states.



There are some problems if clients keep connections open permanently: if an IMAP session in Dovecot consumes between around 1.5 and 2 MB of RAM, it is quite apparent that a mail server with 4 GB RAM cannot provide for 2500 IMAP users at the same time. With POP3, it would be easy to server 25000 active email accounts, as every client immediately logs out and frees up resources. Luckily, current hardware has resolved these problems to a large extent. For more, see Chapter 15.

3.2.3. IMAP in practical terms

Even though RFC 3501, which describes IMAP, runs to far more than 100 pages, the document provides only a limited explanation of the protocol. Marc Crispin, the author of this document, writes: “Apart from an overview of the protocol in section 2, it is not really designed for those who want to understand how to handle the protocol.”[8]

Of course you do not need to have read the respective RFCs or know all the details of the protocol (as documented in Appendix B) to operate a server, and you certainly don’t need to know them by heart. But you should have an understanding of the basic process and of the technical options that the server offers the client.

It is easier to remember things you have already done, so it is a good idea for you to test the IMAP commands demonstrated below on an IMAP server yourself. It will be worth it when an error occurs, and of course it is nice to understand the server you are responsible for.

You don’t need any `root` permissions on the server; an email account accessible via IMAP on the server of an internet provider or email provider will do fine (the password is sent in plain text!). Connect to IMAP port 143 of the server:

```
peer@flash:~> telnet imap.example.com 143
Trying 192.0.2.12...
Connected to imap.example.com.
Escape character is '^]'.
* OK [CAPABILITY IMAP4rev1 LITERAL+ SASL-IR LOGIN-REFERRALS ID ENABLE IDLE
a login tux secret
a OK [CAPABILITY IMAP4rev1 LITERAL+ SASL-IR LOGIN-REFERRALS ID ENABLE IDLE
THREAD=REFERENCES THREAD=REFS MULTIAPPEND UNSELECT CHILDREN NAMESPACE UIDP
I18NLEVEL=1 CONDSTORE QRESYNC ESEARCH ESORT SEARCHRES WITHIN CONTEXT=SEARC
```

The server responds by presenting its *capabilities*. Not all servers support all IMAP features: the IMAP extensions that go beyond the core protocol are always optional.

In the example, the server states which version of the IMAP protocol (`IMAP4rev1`) and which extensions it supports. You can see quite clearly that the server offers the client far more options after login has been successful than before authentication.

Some extensions, including sorting functions such as `THREAD`, which the client uses to instruct the server to find discussion threads, shunt services from the client to the server, while others make it easier to retrieve email and directory structures and thereby save time and traffic (this category includes `UIDPLUS`, described in RFC 2359, `CHILDREN`, described in RFC 3348, and `NAMESPACE`, defined in RFC 2342).

In IMAP, only the `LOGIN` and `PLAIN` methods are mandatory, where the password is transmitted in plain text. Better methods, such as `CRAM-MD5` or `CRAM-SHA1`, are offered as an alternative by the server as shown in the example.

Here, the server also reveals that it supports extensions `IDLE` (RFC 2177, see Section 15.4) and `ACL` (RFC 4314, see Chapter 9). The latter is required if an IMAP folder is to be used by more than one user.

Log in with your user name and (plain text) password:

```
a1 LOGIN "tux" "secret"
a1 OK LOGIN completed
```

By responding `OK`, the server lets us know that we have been authenticated successfully and have now switched to the "Authenticated" state. Now call up a list of all existing directories:

```
a2 LIST "" "*"
* LIST (\HasNoChildren) "." "INBOX.Personal.Vacation"
* LIST (\HasNoChildren) "." "INBOX.Personal.Orchestra"
* LIST (\HasChildren)   "." "INBOX.Personal"
* LIST (\HasNoChildren) "." "INBOX.ToDo"
* LIST (\HasNoChildren) "." "INBOX.Test"
* LIST (\HasChildren) "." "INBOX.Book stuff"
* LIST (\HasNoChildren) "." "INBOX.Book stuff.LPIC-1"
* LIST (\HasNoChildren) "." "INBOX.Book stuff.Postfix 3"
* LIST (\HasNoChildren) "." "INBOX.Book stuff.Snort"
* LIST (\HasNoChildren) "." "INBOX.Book stuff.IMAP"
* LIST (\Unmarked \HasChildren) "." "INBOX"
a2 OK LIST completed
```

The flags in the third column reveal whether the directory in question contains sub-folders (`\HasChildren` – yes, `\HasNoChildren` – no) and messages: folders flagged as `\Unmarked` have not received any new emails since they were last accessed.

The fourth column specifies the hierarchy separator used by the subsequent folder specification: a point is used in our example (instead of a slash, which would produce a folder specification such as `INBOX/Personal/Vacation`).

If you have multiple folders, you can also specify a template for folder names:

```
a3 LIST "" "INBOX.Pers*"
* LIST (\HasNoChildren) "." "INBOX.Personal.Vacation"
* LIST (\HasNoChildren) "." "INBOX.Personal.Orchestra"
* LIST (\HasChildren)   "." "INBOX.Personal"
a3 OK LIST completed
```

Use the `SELECT` command to select a folder, in this case `Test` which is found under the `INBOX`:

```
a4 SELECT INBOX.Test
* FLAGS (\Draft \Answered \Flagged \Deleted \Seen \Recent)
* OK [PERMANENTFLAGS (\Draft \Answered \Flagged \Deleted \Seen)] Limited
* 3 EXISTS
* 1 RECENT
* OK [UIDVALIDITY 1175900586] Ok
* OK [MYRIGHTS "acdilrsw"] ACL
```

```
a4 OK [READ-WRITE] Ok
```

IMAP flags in an email and their significance

There are no fixed flags, so the server uses the keyword `FLAGS` to list the email flags permitted in this folder. In most cases, these will be:

`\Answered`
: Email has been answered

`\Deleted`
: Email has been flagged for deletion

`\Draft`
: Email is a draft and has not been sent yet

`\Flagged`
: Email is flagged as important

`\Seen`
: Email has been read

`\Recent`
: An email with this flag has arrived since the last login and has not yet been retrieved by a client.[9] Unlike the five system flags mentioned, `\Recent` cannot be set using the `STORE` command.

`\*`
: (not shown in this example) used by the server to indicate that it permits custom flags (see Section 7.2.3).

The other parts of the server response mean the following:

`<n> EXISTS`
: reveals how many messages this folder contains in total – in this case 3. The `<n> RECENT` line shows that only one new message was received since the last login.

`OK [PERMANENTFLAGS (<flag1> <flag2> <...>)]`
: lists all the flags that the client may change permanently. If the server does not provide this information, the client can assume that all flags can be modified.

`OK [UIDVALIDITY <n>]`
: specifies the currently valid UID validity value.

`OK [MYRIGHTS <n>]`
: informs the client of the read, write and delete permissions it has for emails in this directory.

`a4`
: this response `OK [READ-WRITE] Ok` contains the day of the request and reveals that the `SELECT` is thereby completed. If the client has write access to the folder, the server *should* add the information `[READ-WRITE]` to the `OK` command. If the client has only read permissions, the server *must* output `[READ-ONLY]`.

Access permissions and their significance

Overview of access permissions from `MYRIGHTS`:

`r`

The user may view the content of a mailbox.

`l`

The name of a mailbox can be displayed to the user, and the user can subscribe to the folder. However, `l` controls only the display in the listing. The actual access is controlled by the `r` flag. If a user knows the name of a folder, he could access that folder even without an `l` flag.

`s`

The user may flag a message as read or unread. The effect of this permission depends on whether the server sets flags for each file or for each email and user – and that can differ in Dovecot, depending on the version and storage format (Maildir/mdbox). If data are stored centrally, a third party will change the status of an email for all users when he changes the `\Seen` flag. If, on the other hand, `\Seen`-flags are saved for each user individually, every user sees only his own read/unread status.

`w`

The user may set keywords and all flags apart from `\Seen` and `\Deleted`.

`i`

The user may paste and copy messages.

`p`

The user may post emails to this folder – by means of Sieve filter scripts, for example, that move emails to sub-folders. Without the `p` flag, messages would continue to end up in the INBOX.

`k`

The user may create a new mailbox or a mailbox subdirectory and, if the `x` flag is set, may rename them as well.

`t`

The user may flag messages as `\Deleted`.

`e`

The user may have messages flagged as `\Deleted` actually deleted from the server (“expunge”).

`x`

The user may delete the IMAP folder (and, in combination with `k`, also rename it).

`a`

The user may set ACLs.

Viewing, copying and deleting emails

Now the client can retrieve messages or parts of messages. Email headers are enough to create a table of contents, for example. The `FETCH` command allows you to control precisely which emails (in this case, messages with sequential numbers 1 to 3)[10] and which parts of these emails should be transmitted. The keyword `ALL` causes the server to return the flags, arrival times, message size in bytes and the header fields `From`, `To`, `Cc`, `Reply-to`, `Message-ID`, `Date` and `Subject`:

```
a5 FETCH 1:3 ALL
* 1 FETCH (FLAGS (\Seen) INTERNALDATE "04-Jan-2014 01:03:06 +0200" RFC822
.SIZE 1647 ENVELOPE ("Sat, 4 Jan 2014 01:01:51 +0200" "Test message 1" (
("Peer Heinlein" NIL "p.heinlein" "heinlein-support.de")) (("Peer Heinlei
n" NIL "p.heinlein" "heinlein-support.de")) (("Peer Heinlein" NIL "p.hein
lein" "heinlein-support.de")) (("Tux" NIL "tux" "example.com")) NIL NIL N
IL "<201401040101.52187.p.heinlein@heinlein-support.de>"))
* 2 FETCH (FLAGS () INTERNALDATE "04-Jan-2014 01:03:06 +0200" RFC822.SIZE
 1646 ENVELOPE ("Sat, 4 Jan 2014 01:02:01 +0200" "Test message 2" (("Pee
r Heinlein" NIL "p.heinlein" "heinlein-support.de")) (("Peer Heinlein" NI
L "p.heinlein" "heinlein-support.de")) (("Peer Heinlein" NIL "p.heinlein"
"heinlein-support.de")) (("Tux" NIL "tux" "example.com")) NIL NIL NIL "<2
<00704070102.01895.p.heinlein@heinlein-support.de>"))
* 3 FETCH (FLAGS () INTERNALDATE "04-Jan-2014 01:03:06 +0200" RFC822.SIZE
1651 ENVELOPE ("Sat, 4 Jan 2014 01:02:10 +0200" "And test message 3" (("
Peer Heinlein" NIL "p.heinlein" "heinlein-support.de")) (("Peer Heinlein"
 NIL "p.heinlein" "heinlein-support.de")) (("Peer Heinlein" NIL "p.heinle
in" "heinlein-support.de")) (("Tux" NIL "tux" "example.com")) NIL NIL NIL
 "<201401040102.11133.p.heinlein@heinlein-support.de>"))
a5 OK FETCH completed.
```

Now let's take a look at the whole of message 2. To do so, we will use `BODY[]`, a subcommand of `FETCH`:

```
a6 FETCH 2 BODY[]
* 2 FETCH (BODY[] {1646}
Return-Path: <p.heinlein@heinlein-support.de>
X-Original-To: p.heinlein@heinlein-support.de
Delivered-To: p.heinlein@heinlein-support.de
Received: from [10.0.42.2] (unknown [10.0.42.2])
        (using TLSv1 with cipher DHE-RSA-AES256-SHA (256/256 bits))
        (Client did not present a certificate)
        by plasma.heinlein-support.de (Postfix) with ESMTP id BEA0581A4B
        for <tux@example.com>; Sat,  4 Jan 2014 01:02:01 +0200 (CEST)
From: Peer Heinlein <p.heinlein@heinlein-support.de>
To: Tux <tux@example.com>
Subject: Test message 2
Date: Sat, 4 Jan 2014 01:02:01 +0200
User-Agent: KMail/1.9.5
MIME-Version: 1.0
Content-Type: text/plain;
  charset="iso-8859-1"
Content-Transfer-Encoding: quoted-printable
Content-Disposition: inline
Message-Id: <201401040102.01895.p.heinlein@heinlein-support.de>
```

```
X-Length: 1519
Status: R
X-Status: NC
X-UID: 0

Hello!
I am a test message.

=2D-=20
Heinlein Support GmbH
Linux: Akademie - Support - Hosting

http://www.heinlein-support.de

Zwangsangaben lt. =A735a HGB:
HRB 93818 B / Amtsgericht Berlin-Charlottenburg,=20
Gesch=E4ftsf=FChrer: Peer Heinlein =A0-- Sitz: Berlin
)
a6 OK FETCH completed.
```

We can also extract individual header lines:

```
a7 FETCH 2 BODY[HEADER.FIELDS Message-ID]
* 2 FETCH (BODY[HEADER.FIELDS ("Message-ID")] {59}
Message-Id: <201401040102.01895.p.heinlein@heinlein-support.de>

)
a7 OK FETCH completed.
```

Of course we can also copy messages – to another folder, or, even simpler, to the same folder, `INBOX.Test`.

```
a8 COPY 2:3 INBOX.Test
a8 OK [COPYUID 1175900586 2:3 4:5] COPY completed.
```

Messages 2 and 3 are now available as messages 4 and 5 as well. It does not make sense to keep duplicates in a folder, so we add the `\Deleted` flag. As a result, they are deleted the next time an `EXPUNGE`, `SELECT` or `CLOSE` command is given (but not if the folder is exited with `UNSELECT`, in line with the IMAP extension from RFC 3691):

```
a9 STORE 2,3 +FLAGS \Deleted
* 2 FETCH (FLAGS (\Deleted))
* 3 FETCH (FLAGS (\Deleted))
a9 OK STORE completed
```

The IMAP server now provides different status information on this folder – there are no longer any new messages, even though the total number of messages was increased by copying in the meantime:

```
a10 NOOP
* 5 EXISTS
* 0 RECENT
a10 OK NOOP completed
```

The messages flagged as deleted can still be used for the time being. Only when we leave the folder by selecting a new directory will the messages actually disappear from the server.

Searching for email content

There are powerful ways to search for specific email content on the server: you can search by age and size of a message, or by sender, header lines, IMAP flags and of course the actual content. In response to the following request, the server returns the sequential numbers of all unread messages:

```
a11 SEARCH UNSEEN
* SEARCH 3 5
a11 OK SEARCH done
```

Email number `1` no longer appears, because it was already marked as `\Seen` at the start of the session, as the `FETCH` command in the example shows.

If you want to search the first four messages for all messages flagged for deletion, the command is

```
a12 SEARCH 1:4 DELETED
* SEARCH 2 3
a12 OK SEARCH done
```

In this case, the result is messages 2 and 3.

Of course you can also search the contents. The `SEARCH` command is not case sensitive.

```
a13 SEARCH ALL TEXT Heinlein
* SEARCH 1 2 3 4 5
a13 OK SEARCH done
```

Careful: `TEXT` allows you to search the raw data of the email, so this search could miss emails that contain the term you are looking for if it is coded. The example will not return any hits, for example, if the word “Heinlein” contains a soft line break in the quoted-printable-coded email text and is therefore actually `Hein=(crlf)lein`.

In most cases it makes more sense to use `BODY[]`, the `SEARCH` subcommand, instead of `TEXT`.

These simple examples already show that the IMAP protocol offers clients numerous ways to outsource even complex tasks to the server efficiently and reduce transmission volumes. Unfortunately, email clients implement the IMAP protocol at very different levels of quality.

Even webmailers, which should be grateful for the variety of options, do not make full use of them. Instead of just drawing those messages from the server that need to be displayed and then displaying them, many retrieve all the emails and then make their selection themselves while using a lot of memory.

3.2.4. Subscribing to IMAP folders

From a list of all folders in an IMAP mailbox, a client can use the `SUBSCRIBE` command to place individual folders onto a subscription list, and remove them again with `UNSUBSCRIBE`. The list of subscribed folders is saved directly on the IMAP server.

Whether a directory is subscribed or not only makes a difference when it comes to the `LIST` and `LSUB` IMAP commands: `LIST` lists all IMAP folders, whether they are subscribed or not. It therefore returns all IMAP folders even if the server contains a subscription list. In contrast, the server will respond to the `LSUB` command (short for "list subscribed") with a list of subscribed directories only.

Apart from this small difference, `LIST` and `LSUB` behave identically. That means that a client can always choose whether to view all directories or only the subscribed ones.

The latter option is sensible for laptop users with UMTS dial-up, and for locations where traffic is expensive or connections are slow. Instead of laboriously synchronizing all IMAP folders when a mailbox is retrieved, only the subscribed IMAP folders are synchronized – as long as the IMAP client is configured correctly. This saves users both money and time while they are out and about. At the same time, an IMAP client at a stationary office desk can be configured to not support subscribed directories for this mailbox, so that it continues to send out the `LIST` command and always replaces the entire data pool.

Where and how you subscribe to IMAP directories is up to your IMAP client. Usually there is a function that enables users to subscribe to individual directories from all listed folders, but there is also a setting determining whether subscriptions should be ignored.

---

[7] by clever use of `rsync` for example: http://www.heinlein-support.de/web/support/wissen/rsync-backup/

[8] http://www.faqs.org/rfcs/rfc3501.html

[9] If several clients access one mailbox, only one of them will see the `\Recent` flag of an email according to the standard.

[10] If you would rather specify UIDs instead of sequential numbers, use the `UID FETCH` command.

Chapter 4. Introduction to the configuration

If you installed Dovecot from a package in your distribution, you should find the configuration in `/etc/dovecot`. Even though the large number of files may appear confusing at first, there is just one large ASCII config file that is split into sub-files.

`/etc/dovecot/dovecont.conf` is where you start. This is where all files are included in `/etc/dovecot/conf.d/*` – nothing unusual so far.

But right at the end of the `10-auth.conf` file there is one unusual feature: from there, additional config snippets `*.conf.ext` are included, and they are used to switch PAM, SQL or LDAP on and off as the authentication backend. The auth modules relating to SQL and LDAP in turn require access data or additional configuration of the SQL statements or LDAP filters, so `auth-sql.conf.ext` contains a reference to `dovecot-sql.conf.ext` and `auth-ldap.conf.ext` contains a reference to `dovecot-ldap.conf.ext` – but we will discuss that in detail in Chapter 5 when we explain the authentication process.

Depending on the Dovecot version and installed sub-packages of the distribution, the file tree of the configuration files will look something like this:

```
dovecot.conf
+--conf.d/10-auth.conf
  +--conf.d/auth-checkpassword.conf.ext
  +--conf.d/auth-deny.conf.ext
  +--conf.d/auth-ldap.conf.ext
      +-- refers to ../auth-ldap.conf.ext
  +--conf.d/auth-master.conf.ext
  +--conf.d/auth-passwdfile.conf.ext
  +--conf.d/auth-sql.conf.ext
      +-- refers to ../auth-ldap.conf.ext
  +--conf.d/auth-static.conf.ext
  +--conf.d/auth-system.conf.ext
  +--conf.d/auth-vpopmail.conf.ext
+--conf.d/10-director.conf
+--conf.d/10-logging.conf
+--conf.d/10-mail.conf
+--conf.d/10-master.conf
+--conf.d/10-ssl.conf
+--conf.d/15-lda.conf
+--conf.d/15-mailboxes.conf
+--conf.d/20-imap.conf
+--conf.d/20-lmtp.conf
+--conf.d/20-managesieve.conf
+--conf.d/20-pop3.conf
+--conf.d/90-acl.conf
+--conf.d/90-plugin.conf
+--conf.d/90-quota.conf
+--conf.d/90-sieve.conf
```

Configuration of Dovecot takes place in ASCII text, which is easy to read. Many of the parameters have names similar to those in popular mail server Postfix. In short, the configuration is a pleasure for admins.

You will find that many commented-out parameters have been prepared in the configuration files:

```
# Log file to use for error messages. "syslog" logs to syslog,
# /dev/stderr logs to stderr.
#log_path = syslog
```

This setting corresponds to the default value, so the value can remain commented out if you do not want to make any changes. That is also familiar from Postfix and other good software.

Unlike Postfix, Dovecot from version 2.0 makes considerable use of *sections* that contain specific settings for specific modules or areas in curly brackets (`{ }`). To put it another way: configuration parameters are always defined within their "context".

Here are two small examples to illustrate: while the plugins to be loaded are defined generically in `10-mail.conf`:[11]

```
mail_plugins = quota acl
```

`mail_plugins` are specified more precisely in the configuration of the `pop3` or `imap` protocol. While `mail_plugins` are applied in POP3 unchanged,

```
protocol pop3 {
[...]
  # Space separated list of plugins to load (default is global
  # mail_plugins).
#  mail_plugins = $mail_plugins
[...]
}
```

additional entries are added to the default `$mail_plugins` parameter for the context of the IMAP protocol:

```
protocol imap {
[...]
  # Space separated list of plugins to load (default is global
  # mail_plugins).
  mail_plugins = $mail_plugins imap_quota imap_zlib
[...]
}
```

We will come across this again in different parts of the book.

4.1. The doveconf tool

Dovecot has another similarity with Postfix: the tool `doveconf`, which has the same effect as the command `doveadm config`, is used to read out and check the configuration, the same function `postconf` performs in Postfix. The call parameters are also based on `postconf`:

`doveconf`
: lists the currently valid configuration, which is composed of the values in the configuration files and any other default values.

`doveconf -n`
: lists all parameters of the current configuration files that are *not* set to their default values. This provides you with a quick and compact overview of the individual settings of a server.

`doveconf -N`
: lists all parameters of the current configuration files, even if they remain set to their default values.

`doveconf -d`
: lists all current parameters with their default values (and ignores the current configuration file).

Because the configuration is often more than one line long and contains the above sections, *doveconf*, unlike *postconf*, cannot be used to edit parameters. To do that, you still have to enter the respective files with a text editor.

You can, however, query specific parameters in their specific context:

```
flash:~ # doveconf mail_plugins
mail_plugins = quota zlib

flash:~ # doveconf  -f service=imap mail_plugins
mail_plugins = quota zlib imap_quota  imap_zlib
```

You can also select a whole section to be queried:

```
flash:~ # doveconf plugin
plugin {
  quota = maildir:User quota
  quota_rule = *:storage=1G
  quota_rule2 = Trash:storage=+100M
  sieve = ~/.dovecot.sieve
  sieve_dir = ~/sieve
  zlib_save = gz
  zlib_save_level = 6
}
```

4.2. Where and how to store your local configuration

There are two ways to adapt parameters of the Dovecot configuration:

1. You can edit the value in the configuration file where it is commented and, if equipped with a hash, where it is prepared. That prevents duplicate entries and also makes it easy to follow for colleagues who do not work with Dovecot every day.
2. You can enter all adapted values in `/etc/dovecot/local.conf`. As this file is included at the end of `dovecot.conf` by means of an `include_try`, you always have "the last word" and can finally define the value to which a parameter points:

```
# Most of the actual configuration gets included below. The filenames are
# first sorted by their ASCII value and parsed in that order. The
# 00-prefixes in filenames are intended to make it easier to understand
# the ordering.
!include conf.d/*.conf

# A config file can also try to be included without giving an error if
# it's not found:
!include_try local.conf
```

I urgently advise you to choose version a) at first and change all parameters directly in the configuration snippets themselves. Experience shows that administrators can find their way around more easily and make fewer mistakes.

Of course, you will have to adapt existing configuration files to modified files from the installation packages when performing major updates, but that is rarely necessary and also not difficult. And of course you can always dump your personal configuration by calling `doveconf -n` – while `doveconf -n > local.conf` would pave the way for updates without manual transmissions.

4.3. Activating log messages

When you first start working with Dovecot, it is helpful to have descriptive log files with plenty of explanatory information. You should adjust the verbosity of your Dovecot first:

In `10-logging.conf`:

`auth_verbose = yes`
: writes more detailed log messages on the authentication process.

`auth_debug = yes`
: goes one step further and writes the debugging of the authentication prompts into the log file.

`mail_debug = yes`
: logs in detail how Dovecot searches for the user's mail directory, how it analyzes it and how it specifies the related details.

If you are setting up a pure mail server that no normal user has shell access to, you should also instruct Dovecot to display the names of the POP3 and IMAP processes in such a way that the user's login name, IP address, SSL/TSL status and the currently executed IMAP command are displayed. That will make debugging considerably more pleasant later on, for example when determining the user who owns the IMAP process that is occupying 100% of a CPU kernel according to `top`.

Add the following to `/etc/dovecot/dovecot.conf`:

```
verbose_proctitle=yes
```

After a restart/reload, a user's POP3 and IMAP processes will usefully be displayed as follows:

```
1770 ?  S  0:00 dovecot/imap [klaus@example.com 10.0.40.6 IDLE]
```

There is another parameter that you need to adapt right at the start so that you can reproduce the examples described in the next chapters: use `mail_location` in file `10-mail.conf` to instruct Dovecot to save the test emails from the next chapters in the *Maildir* format in directory `~/Maildir`, i.e. the Maildir folder in the home directory of the user in question:

```
mail_location = maildir:~/Maildir
```

Later on, we will adapt the configuration for virtual users without home directories and for other storage formats, but you should start off in the way described above, because that will allow us to discuss user authentication in Dovecot in the next chapter.

4.4. Specifying the IMAP namespace

There is another highly important decision you need to make right at the start before you set up the first user live on your mail server: the structure of the IMAP namespace.

There are two important questions you need to consider:

- Which characters separate the hierarchy levels between IMAP folders? Usually you would use a point (.), and this is the default value in Dovecot. But you could also use a slash (/).
- Do IMAP folders have to be under the INBOX in the hierarchy, or may they be located at the same hierarchical level? By default, folders may be located at the same level as the INBOX.

You will find these settings in the `10-mail.conf` file, fairly close to the top in the long `namespace inbox` block (in older versions of Dovecot, it is just `namespace` without `inbox`):

```
namespace inbox {
  # Namespace type: private, shared or public
  #type = private

  # Hierarchy separator to use. You should use the same separator for all
  # namespaces or some clients get confused. '/' is usually a good one.
  # The default however depends on the underlying mail storage format.
  separator = .

  # Prefix required to access this namespace. This needs to be different
  # for all namespaces. For example "Public/".
  prefix = INBOX.

  # Physical location of the mailbox. This is in same format as
  # mail_location, which is also the default for it.
  #location =

  # There can be only one INBOX, and this setting defines which namespace
  # has it.
  inbox = yes

  # If namespace is hidden, it's not advertised to clients via NAMESPACE
  # extension. You'll most likely also want to set list=no. This is mostly
  # useful when converting from another server with different namespaces
  # which you want to deprecate but still keep working. For example you
  # can create hidden namespaces with prefixes "~/mail/", "~%u/mail/" and
  # "mail/".
  #hidden = no

  # Show the mailboxes under this namespace with LIST command. This makes
  # the namespace visible for clients that don't support NAMESPACE
  # extension.
```

```
  # "children" value lists child mailboxes, but hides the namespace
  # prefix.
  #list = yes

  # Namespace handles its own subscriptions. If set to "no", the parent
  # namespace handles them (empty prefix should always have this as "yes")
  #subscriptions = yes
}
```

Settings `separator` and `prefix` are the important ones here – you can ignore the others for the time being.

4.4.1. The right hierarchy separator

There are probably historical reasons why the point is nearly always chosen as the hierarchy separator. It does no harm as a directory name in the file system, while the slash was used as hierarchy separator in Linux/Unix.

In *shared folders* however, where users can allow other users to access their own IMAP folders, points may also occur in the user name, with unlovely results (for more information, see Chapter 9). Migrating a namespace during operation is highly unpleasant if you have already connected hundreds or even thousands of mail clients that may be caught off guard by such adjustments.

If you are starting off with a new empty IMAP server

If there are no existing data to migrate, *now* is the best time to set things on the right track and set your hierarchy separator to `/`. This hierarchy separator is not usually used in user names and will not restrict your options later on. Dovecot has no difficulties saving these folders to the correct file system.

In this case, enter the following in the section: `separator=/`

If you are dealing with existing data

Leave everything the way it is for the time being. Check the namespace that the old IMAP system was set up in and configure the same namespace in Dovecot. Use the `List` IMAP command to determine from the IMAP server which hierarchy separator is used. In this example, it is a point:

```
a1 LOGIN "tux" "secret"
a1 OK LOGIN completed
a2 LIST "" "*"
* LIST (\HasChildren) "." INBOX
```

And in this example, a `/` is set up as the hierarchy separator:

```
a2 LIST "" "*"
* LIST (\HasChildren) "/" INBOX
```

You should be very wary of modifying the namespace, particularly if you have insufficient experience of Dovecot and IMAP migrations. Do not change too much at one time; the situation can be complicated and confusing enough when you change your IMAP software, and migrating data while also changing the namespace will not make things easier.

Once you have completed the move to Dovecot, you can take your time when making the necessary adjustments later on. Take a look at the section on migrating IMAP data in Section 19.4.

Apart from that, there are almost no restrictions when it comes to the naming of IMAP folders: except for the hierarchy separator, you can use almost any character in the name of an IMAP folder: commas, spaces or even specific national characters (such as umlauts). The latter are then stored in the file system in 7-bit coding.[12]

4.4.2. Choosing the folder prefix

Different IMAP servers disagree on whether directories may exist in parallel to the INBOX or only under it. Can you have directories `INBOX`, `friends` and `company`? Or can you only have `INBOX`, `INBOX.friends` and `INBOX.company` (or `INBOX/friends` and `INBOX/company`)?

The implementations out there in the wild differ accordingly:

- Courier IMAP only allows directories under the INBOX
- Cyrus and Dovecot allow both versions
- All other systems vary

The rule is: if you are migrating existing data, try not to modify the namespace as well. The task is challenging enough as it is. You should address the namespace later on in a second step, once the situation has calmed down and you are familiar enough with the Dovecot system. You will find detailed information (and warnings) later on in Chapter 19.

If you have the opportunity to start from scratch, you should make the right decision at this point and consider the following:

- If you intend to use shared folders later on, it can make sense to clearly separate private and shared folders as shown in Figure 4.1. In that case, there will be the `INBOX` with all its sub-folders and the `shared` folder with all its sub-folders (as well as a `public` folder if you decide to introduce it). This will avoid the problems that occur when a user sets up a regular folder named `shared`...

Figure 4.1. Your own folders are clearly separated from the shared folders of other users

- Power users who integrate multiple accounts into their mail client in parallel will be

very pleased if they can collapse all sub-folders under the `INBOX`. The INBOX folders of the various accounts will remain open and usable, as is the case with the company account in Figure 4.2. I myself have dozens of email accounts in my email client and would be unable to work if I were unable to hide all subfolders at the push of a button.

Figure 4.2. If all folders are under the INBOX, the whole mailbox can be collapsed so that the INBOXes remain usable.

- At the same time, there is no real drawback if you only permit folders under the INBOX. It just makes folder names longer, because they have to be called `INBOX.sent`, for example.

If you (can) follow my advice, set the suitable prefix in the `namespace inbox` section of the `10-mail.conf` file. Remember that the prefix must end with the hierarchy separator. Depending on the setup, the entry should be `INBOX.` or `INBOX/` – and not just `INBOX`:

```
namespace inbox {
  type = private
  hidden = no
  ignore_on_failure = no
  inbox = yes
  list = yes
  location =
  prefix = INBOX/
  separator = /
  subscriptions = yes
}
```

4.4.3. Standardized names for trash, sent & Co.

For years, or rather decades, every email client irritatingly had its own ideas on what to call the IMAP folder for sent messages: `sent`, `sent messages`, …? Once Outlook started creating these folders with translated names ("sent objects"), there was inevitable chaos when different mail clients or webmailers had to work together in one IMAP account and every mail client set up the folder structure it considered correct.

Worse, plenty of mail clients displayed names for these "special folders" that were different from the actual IMAP names. As a result, you could never know for sure what the name of the underlying IMAP folder actually was.[13]

That is why RFC 6154 defines an IMAP extension that mail clients use to agree with the server on the IMAP names of these *special-use mailboxes*. It allows standardized IMAP folder names to be assigned centrally, while every client can provide its own choice of name to its users.

In Dovecot versions from 2011 onwards, the definitions of these folder names can be found in the `/etc/dovecot/conf.d/15-mailboxes.conf` file. The `namespace inbox` section, which is already defined in `10-mail.conf`, is added there.[14] If you cannot find the `15-mailboxes.conf` file, your version of Dovecot is too old.

If you like, you can change the naming here – if you do not need to map structures that have developed over time, you should not make any changes and instead enjoy the fact that uncontrolled growth will be prevented in future:

```
# NOTE: Assumes "namespace inbox" has been defined in 10-mail.conf.
namespace inbox {

  #mailbox name {
    # auto=create will automatically create this mailbox.
    # auto=subscribe will both create and subscribe to the mailbox.
    #auto = no

    # Space separated list of IMAP SPECIAL-USE attributes as specified by
    # RFC 6154: \All \Archive \Drafts \Flagged \Junk \Sent \Trash
    #special_use =
  #}

  # These mailboxes are widely used and could perhaps be created
  # automatically:
  mailbox Drafts {
    special_use = \Drafts
  }
  mailbox Junk {
    special_use = \Junk
  }
  mailbox Trash {
    special_use = \Trash
```

```
  }

  # For \Sent mailboxes there are two widely used names. We'll mark both
  # of them as \Sent. User typically deletes one of them if duplicates are
  # created.
  mailbox Sent {
    special_use = \Sent
  }
  mailbox "Sent Messages" {
    special_use = \Sent
  }

  # If you have a virtual "All messages" mailbox:
  #mailbox virtual/All {
  #  special_use = \All
  #}

  # If you have a virtual "Flagged" mailbox:
  #mailbox virtual/Flagged {
  #  special_use = \Flagged
  #}
}
```

It is up to you to decide whether to map a junk folder: if you can do without spam tagging and instead filter spam and viruses in Postfix in real time using the proxy filter mode, there is not really any need for that folder.

The ability to map “virtual folders” as shown at the end of the listing requires manual intervention in the file system and would go beyond the scope of this book.[15]

---

[11] These parts of the configuration serve only to illustrate the syntax. We will explain the significance of plugins in later chapters.

[12] Every now and then, particularly in universities, you will come across very old `uw-imap` installations that simply stored 8-bit folder names in the file system uncoded. In this case, you will need shell scripts to repair the situation. IMAP folders belonging to Korean exchange students are particularly enjoyable in this context, whether you have a German or a Korean keyboard. My advice is: you should know when you have lost – and in this case you have. Use `*` as your wildcard to address the folders in the file system, throw out the names and number such folders sequentially. Once they can be found in Dovecot, users can give them proper names; Dovecot will then code and save the special characters correctly.

[13] In foreign-language versions, it got even more exciting, because Outlook would first look for existing English folder names; if it found a folder called `sent`, it would use that folder, but display the foreign language name. If the folder did not exist, Outlook would create it with the name in the other language and display it to the user in that language, but English versions of Outlook could then no longer understand the name…

[14] The `15-mailboxes.conf` file can be used to add the definitions of the special folders at a later stage without messing up the administrator’s settings in `10-mail.conf`.

[15] For more information, go to http://wiki2.dovecot.org/Plugins/Virtual.

Chapter 5. Authentication

In this chapter we will take a look at the various ways in which Dovecot accesses a variety of authentication sources. There are plenty of details that are worth knowing and will make your life easier or, if you make the wrong choices, harder. On the next pages I will describe all configuration versions with their advantages and drawbacks, so the chapter will have plenty of twists and turns. I still advise you to read it in full, because a good understanding of the authentication process and all its aspects is essential.

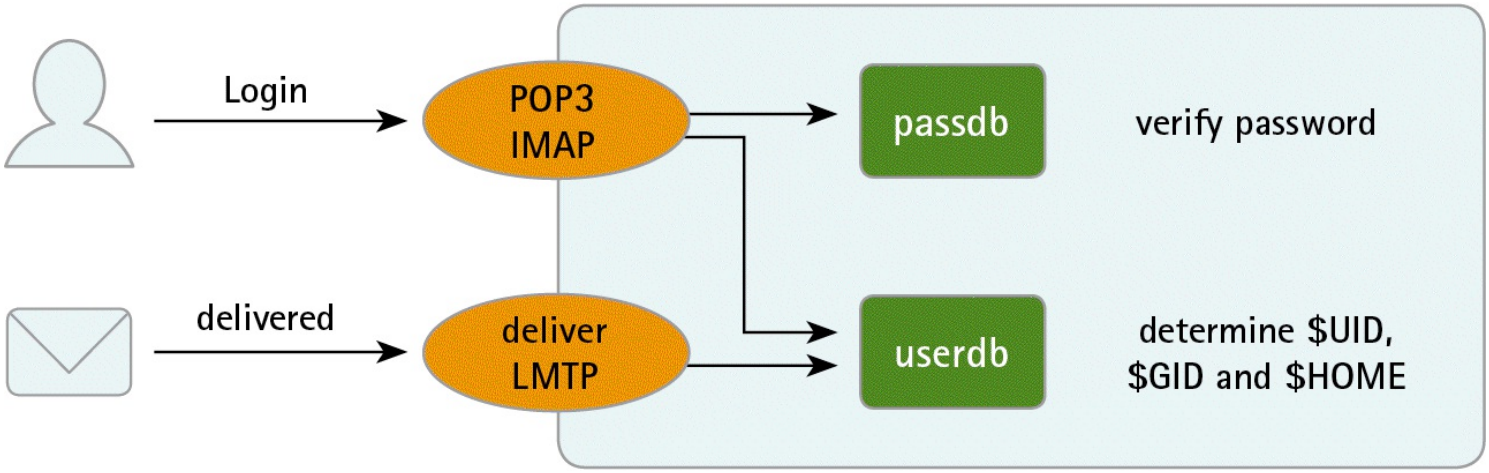
Dovecot requires the following information for successful authentication:

- User name
- Password
- UID of the user under which the email files are saved
- GID of the user under which the email files are saved
- The home directory of the user where email files, Sieve scripts and other individual items are stored

It is hardly surprising that this is the information saved in `/etc/passwd` and `/etc/shadow` – after all, every Linux service relies on more or less the same information.

Dovecot distinguishes between two types of authentication requests, both of which you need to know, and which are illustrated in Figure 5.1.

Figure 5.1. Dovecot requires `userdb` and `passdb` lookups at various points.



`passdb` lookup
: Here, Dovecot simply verifies the connection between the user name and password – that is all. The `passdb` lookup is required whenever a user logs in with POP3, IMAP or to Managesieve and Dovecot has to decide whether to grant that user access.

`userdb` lookup

Here, Dovecot determines the user's environment, i.e. his UID, GID and the path to his home directory. After successful login (i.e. a successful `passdb` lookup) Dovecot performs a `userdb` lookup to find out more about the user who has just logged in.

But this is not the only place where it is used: Dovecot requires a user's `userdb` lookup again and again, mainly when Dovecot receives an email for a user and has to save it in the right place with the right file permissions. The password is of no importance here, but Dovecot does need to know the environment. If you plan to introduce *shared folders* later on (see Chapter 9), Dovecot must be able to determine the environment of various users so that one user can access the email data of another user.

The distinction between `userdb` and `passdb` lookup is not that unusual; the Linux system is also familiar with it. Passwords are stored in `/etc/shadow`, while the user's environment is stored in `/etc/passwd`. Once again, there is a clear distinction between a password request during login and the fact that the Linux system needs to know which user names belong to which UIDs (and vice versa) in order to perform file listings or manage processes.

The `doveadm` Dovecot tool is a convenient way to debug `passdb` and `userdb` lookups. You can clearly see the distinction between authentication and user environment here:

`doveadm auth test`
: asks for a password and authenticates the user (`passdb` lookup):

```
flash:~ # doveadm auth test peer
Password:
passdb: peer auth succeeded
```

In older versions of Dovecot, the command was simply `doveadm auth`:

```
flash:~ # doveadm auth peer
Password:
passdb: peer auth succeeded
```

`doveadm user`
: performs a `userdb` lookup and returns the environment of the user in question (*userdb lookup*):

```
flash:~ # doveadm user peer
userdb: peer
  system_groups_user: peer
  uid       : 1000
  gid       : 100
  home      : /home/peer
```

Particularly if you set up your authentication on the basis of SQL or LDAP/AD later on, these queries will provide important assistance in checking that everything works properly and in debugging.

5.1. Basic settings

In order to understand the following examples of different authentication methods, you should first familiarize yourself with some of the details of `10-auth.conf`.

In configuration information, you can use a number of variables to refer to an email address and/or login name:

`%n` (local part)
: Refers to the local part, i.e. the part before the `@` sign. If the user name is `klaus@example.com`, `%n` corresponds to `klaus`. If the login name is just `klaus` without a domain part, `%n` will continue to refer only to `klaus`.

`%d` (domain part)
: Refers to the domain part, i.e. the part after the `@` sign. If the user name is `klaus@example.com`, `%d` corresponds to `example.com`. If the login name is just `klaus` without a domain part, `%d` will remain empty.

`%u` (email address)
: Refers to the full email address or login name, i.e. `klaus@example.com` or, in the case of a user name such as `klaus` that consists only of a local part, just to `klaus`.

The `auth_username_format` parameter in `10-auth.conf` is important. It adapts the user name *before* it is transmitted to the authentication process.

```
auth_username_format = %n
```

cuts off the domain part. Entering `klaus@example.com` will still result in a search for `klaus`.

```
auth_username_format = %u
```

contains the user name as specified in the email delivery or POP3/IMAP login.

It is very useful to take this opportunity to make sure that the names are always converted to lower-case characters. Later on, when you have email storage paths calculated from the email address or login name, a mixture of upper and lower case characters will otherwise lead to different paths in the file system. Add `L` to the variable specifications to ensure lower-case characters here.

For local parts in the authentication backend, you should always specify

```
auth_username_format = %Ln
```

or, for email addresses,

```
auth_username_format = %Lu
```

in order to ensure standardized spellings – otherwise unpleasant surprises are inevitable.

5.2. Authentication sources

By default, Dovecot is familiar with the following authentication sources:

- System user, i.e. `/etc/passwd` and `/etc/shadow`
- SQL queries matched against MySQL or PostgreSQL
- LDAP queries, either via LDAP binds or LDAP proxy user
- Kerberos[16]
- Queries in `passwd-file`, i.e. a Dovecot-specific password file in `/etc/dovecot/users`
- `checkpassword`
- `vpopmail`
- `static`

In practice, the first four are the most relevant, while `checkpassword`, `vpopmail` or tricks using `static` are a rather more exotic choice and therefore not described further in this book.[17]

The includes at the end of the `10-auth.conf` file determine which authentication method is used:

```
##
## Password and user databases
##

#!include auth-deny.conf.ext
#!include auth-master.conf.ext

!include auth-system.conf.ext
#!include auth-sql.conf.ext
#!include auth-ldap.conf.ext
#!include auth-passwdfile.conf.ext
#!include auth-checkpassword.conf.ext
#!include auth-vpopmail.conf.ext
#!include auth-static.conf.ext
```

5.3. /etc/passwd & /etc/shadow

By default, `auth-system.conf.ext` is currently active. It defines a `userdb` lookup in `/etc/passwd` and a `passdb` lookup in PAM and therefore usually in `/etc/shadow`:

```
passdb {
  driver = pam
  # [session=yes] [setcred=yes] [failure_show_msg=yes] [max_requests=<n>]
  # [cache_key=<key>] [<service name>]
  #args = dovecot
}

userdb {
  # <doc/wiki/AuthDatabase.Passwd.txt>
  driver = passwd
  # [blocking=no]
  #args =
}
```

An IMAP server for real Linux users in a system is quick to set up and even works from the get-go as long as you have set `auth_username_format=%Ln` as described in Section 5.1 , as `/etc/passwd` can only contain user names made up of local parts.

However, this version is not particularly useful for a true mail server:

- You will hardly want to set up a real user in the system for every email account (even if there are various ways to ensure that such users cannot log in).
- `/etc/passwd` can store only user names, not email addresses (and email addresses can be very useful as login names).
- `/etc/shadow` can store only hashed passwords, so it is not possible to offer secure password methods such as CRAM-MD5 or DIGEST-MD5 (for more, see Section 5.10).
- A standard Linux system knows a maximum of 65353 different user UIDs – but large mail servers have hundreds of thousands or even millions of users.

5.4. passwd-file

For anyone who wants simple user management in ASCII files but is unwilling to fall back on `/etc/passwd`, Dovecot offers `passwd-file`, which has exactly the same structure as `/etc/passwd` but is located in `/etc/dovecot/users`, so users entered there have nothing to do with the Linux system.

`passwd-file` has the following syntax:

```
username:password:uid:gid:realname:home:extra_fields
```

In contrast to the "real" `/etc/passwd`, it already contains the password – and the last column is not used to define a login shell, so it can later on contain other settings such as quota rules (see Chapter 11).

The nice thing about `passwd-file` is that the user name in the first column can now contain an `@` and therefore a full email address. The following entries are no problem as long as `auth_username_format=%Lu` is set as described in Section 5.1:

```
bla@example.org:{PLAIN}test:10000:10000::/srv/vmail/example.org/bla::
blubb@example.com:{PLAIN}test:10000:10000::/srv/vmail/example.com/blubb::
```

Dovecot caches this `passwd-file` and independently detects changes to it. You can therefore maintain even extensive user lists simply with a text editor. Particularly in smaller setups, where it is not worth setting up a proper user management structure, `passwd-file` is often the best method for user management. If you maintain `passwd-file` with a text editor, it is conversely your responsibility to ensure that no syntax or typing errors appear.

But `passwd-file` can also be useful in large setups: We once exported a provider's user data to such a `passwd-file` by means of a cron shell script, and the entire system with more than 100,000 users ran stably and quickly. Thanks to good caching in Dovecot, it took just a few milliseconds to authenticate a user. In particular, there are benefits to the fact that you do not depend on other services to run a working mail server. The drawback is of course that changes are no longer available on the system in real time and are instead delayed until the next cron run.

5.5. LDAP queries/active directory

Of course you can also choose LDAP as the authentication source for Dovecot. In principle, Dovecot proceeds no differently than other program with an LDAP backend, so you have to configure the host, access data, LDAP query filter and so on. However, there are some things you need to bear in mind.

First, you should ask yourself which of the two methods should be used to check the passwords in LDAP:

Authentication binds
: In this case, Dovecot uses the password transmitted by the client and attempts to log on to the LDAP or AD server with this login name and password. If login is successful, this combination of user name and password was obviously correct.

Password lookups
: In this case, LDAP is used like a database. Dovecot has read access to all user data records on the LDAP server by means of a specific *proxy user*, and uses the login name to find the correct data record. The password field is then simply read out of this data record, so Dovecot determines whether the client has transmitted the correct password.

If you want to authenticate against an ActiveDirectory server, you have to use authentication binds. In an AD server, you will never be able to read out an entry's password field – so password lookups will not work here. In turn, this (seeming) security ensures that, next to Kerberos, the client can only use plain text passwords and the PLAIN and LOGIN methods to authenticate itself with AD setups – secure challenge-response processes cannot work, because Dovecot would then not find out the plain text password it needs to log on to the AD (see Section 5.10). Definitely a mixed blessing.

If you want to use AD or LDAP as your authentication backend, you must first activate LDAP in `10-auth.conf` by activating the correct include at the end (and deactivating other methods if applicable):

```
#!include auth-deny.conf.ext
#!include auth-master.conf.ext

#!include auth-system.conf.ext
#!include auth-sql.conf.ext
!include auth-ldap.conf.ext
#!include auth-passwdfile.conf.ext
#!include auth-checkpassword.conf.ext
#!include auth-vpopmail.conf.ext
#!include auth-static.conf.ext
```

`auth-ldap.conf.ext` defines the `passdb` and `userdb` settings – though reference is at first made only to another configuration file containing the necessary LDAP settings:

```
passdb {
  driver = ldap

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

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

`dovecot-ldap.conf.ext` appears confusing at first, because various LDAP versions are possible, and because authentication binds and password lookups are prepared as alternatives in the same file even though they require different procedures.

5.5.1. To begin with: LDAP analysis with ldapsearch

If you want to connect Dovecot to LDAP, you have to know the structure of your LDAP server, i.e. the attributes it uses to save user names and email address(es), and whether the LDAP attributes of the passwords can be read out (OpenLDAP) or not (ActiveDirectory). It is also important to know whether quota information and home directories are stored, and under which attribute names.

First, check whether your LDAP reveals user data to non-authenticated users, and if so, which ones.

Place an anonymous request:

```
ldapsearch -h ldapserver.example.com -b ou=user,o=example,c=com
```

Check manual access with the user name and password of an admin account:

```
ldapsearch -h ldapserver.example.com -b ou=user,o=example,c=com -x -W -D
cn=adminaccount,ou=admin,o=example,c=com
```

The values are:

`-x`
: use simple bind

`-W`
: request password

`-h`
: host name of the LDAP server

`-D`
: bindDN of the admin account

`-b`
: search basis, i.e. the DN where the search for the user should take place

This is a good opportunity to mention the following:

`-LLL`
: suppresses all comments from `ldapsearch`, and returns a pure LDIF file for further processing, in case you need or want to modify your LDAP schema automatically during the migration.

If access to the LDAP data is successful, you should analyze the LDIF output more precisely and remember the required LDAP attributes. You should also determine the correct LDAP search filter that will effectively provide you with all legitimate user DNs.

In this example, an ActiveDirectory server is accessed with a user name and password by means of a search filter that only returns users with a recorded email address:

```
ldapsearch -h dc1.vc.example.com -b dc=users,dc=example,dc=com -LLL -x -W
-D adminaccount@example.com "(&(objectclass=user)(mail=*))"
```

Before you go on, you should continuously adapt your DNs and search filters until you receive meaningful results.

5.5.2. Configuration of password lookups (OpenLDAP)

Now we come to the actual configuration of the LDAP access. We begin by specifying the LDAP server or servers in the `/etc/dovecot/dovecot-ldap.conf.ext` file:

```
# Space separated list of LDAP hosts to use. host:port is allowed too.
hosts = 192.168.5.10 192.168.10.10
```

Now you have to enter the bind DN and bind password of an LDAP user reserved for that purpose that allows Dovecot (read) access to all user data records along with their password attribute. Even if you were able to read out extensive user information in the previous section with an anonymous bind, you should (with luck) only be able to read out password attributes in the context of a bind with the aid of an admin DN:

```
# Distinguished Name - the username used to login to the LDAP server.
# Leave it commented out to bind anonymously (useful with auth_bind=yes).
dn = cn=dovecotdummyuser,dc=example,dc=com

# Password for LDAP server, if dn is specified.
dnpass = secretpassword
```

Tell Dovecot you do not want to use *authentication binds*:

```
auth_bind = no
#auth_bind_userdn =
```

Section 5.5.3 describes the alternative method of using *authentication binds*, and you can skip it if you like. Go straight to Section 5.5.4 to make the other LDAP settings.

5.5.3. Configuration of authenticated binds (Active Directory, OpenLDAP)

The configuration of authenticated binds differs only slightly from the method using password lookups that was described earlier. First, specify the required LDAP/AD server or servers in `/etc/dovecot/dovecot-ldap.conf.ext`:

```
# Space separated list of LDAP hosts to use. host:port is allowed too.
hosts = 192.168.5.10 192.168.10.10
```

Even if you intend to have the authentication of user passwords performed by authenticated binds, Dovecot still requires its own LDAP user that it can use for (read-only) requests to the LDAP server.

After all, Dovecot still needs to be able to read out a user's environment as part of the `userdb` request, and may need to find the LDAP account corresponding to the email address provided, in order to then verify the transmitted password in a second step in the context of an authentication bind to LDAP.

```
# Distinguished Name - the username used to login to the LDAP server.
# Leave it commented out to bind anonymously (useful with auth_bind=yes).
dn = cn=dovecotdummyuser,dc=example.com,dc=com

# Password for LDAP server, if dn is specified.
dnpass = secretpassword
```

Some LDAP servers have a very open configuration and return plenty of data on the registered users even without `bind_dn`. ActiveDirectory servers were also often surprisingly chatty for anonymous connects. I showed you how to test that in Section 5.5.1. If you can get all the information you require without `bind_dn`, you can do without the configuration of the Dovecot user via `dn` and `dnpass` in this case.

However, unlike in password lookups, you have to tell Dovecot to use the user's login data for authentication with the LDAP server:

```
# Use authentication binding for verifying password's validity. This works
# by logging into LDAP server using the username and password given by
# client.
# The pass_filter is used to find the DN for the user. Note that the
# pass_attrs is still used, only the password field is ignored in it.
# Before  doing any search, the binding is switched back to the default
# DN.
auth_bind = yes
```

Of course, Dovecot needs to know which DN to use when registering with the server for verification. There are two versions.

The local part of the email address corresponds to the name of the LDAP

object

If the local part of the email address already corresponds to the common name or UID of an LDAP object, for example because the local part of the email address is identical to the Windows login name, you can provide Dovecot directly with the `auth_bind_userdn`. This configuration saves on `passdb` lookups and therefore LDAP requests at first, but you have to invest those saved lookups later on in `userdb` lookups, because *prefetching* is no longer possible (for more information, see Section 5.7). In addition, this version requires more effort to implement and is therefore not necessarily advisable.

1. Use an LDAP browser or another tool in your AD to take a look at your user list, choose a sample user and thereby determine the DN under which the object is saved.

   ```
   # If authentication binding is used, you can save one LDAP request per
   # login if users' DN can be specified with a common template. The
   # template can use the standard %variables (see user_filter). Note tha
   # you can't use any pass_attrs if you use this setting.
   #
   # If you use this setting, it's a good idea to use a different
   # dovecot-ldap.conf.ext for userdb (it can even be a symlink, just as
   # long as the filename is different in userdb's args). That way one
   # connection is used only for LDAP binds and another connection is use
   # for user lookups. Otherwise the binding is changed to the default DN
   # before each user lookup.
   auth_bind_userdn = cn=%Ln,ou=People,dc=firma,dc=local
   ```

   As shown in Section 5.1, you can specify `%n`, `%u` and `%d`, and you should always use them in lower case (`%Ln`) to be on the safe side.

   Use the `ldapsearch` requests shown earlier to quickly determine the structure of your LDAP or AD server and find out exactly which DN your user objects are saved under.

2. You can (if you want) now optimize the performance by asking Dovecot to perform the LDAP requests for `userdb` and `passdb` lookup at the same time. That makes up for the lack of prefetching. Two separate authentication sources here, or they must at least appear separate to Dovecot. Create a symlink:

   ```
   flash:~ # cd /etc/dovecot
   flash:/etc/dovecot # ln -s dovecot-ldap.conf.ext dovecot-ldap-userdb.c
   ```

   Now enter the two paths in `conf.d/auth-ldap.conf.ext`:

   ```
   passdb {
     driver = ldap
     args = /etc/dovecot/dovecot-ldap.conf.ext
   }

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

```
}
```

The bind DN has to be determined by means of an LDAP lookup

If your LDAP tree has a broad structure with users located on different LDAP branches, or if the email addresses or local parts of the email addresses differ from the names of the user LDAP objects, Dovecot first has to determine the correct auth DN by means of a password lookup. In that case, it uses the `pass_filter` configured (in Section 5.5.4) to perform an LDAP request and determines the correct DN of the corresponding LDAP object. In a second step, this DN is then used for authentication.

If you first have to determine the DN dynamically by means of a request, you cannot and must not enter `auth_bind_userdn` here. You should therefore leave it commented out.

Usually you will have to (and prefer to) choose the second version and have the auth DN determined dynamically.

5.5.4. Setting LDAP search filters for userdb and passdb requests

Whether you use authenticated binds or password lookups as your method, both versions come together again at this point, and the next sections apply to all setups based on LDAP.

Of course, like in any such LDAP request, there must be a specification of the search basis under which the suitable user entries are searched for in LDAP. In larger LDAP trees in particular you should restrict the number of objects and branches to be searched as far as possible:

```
# LDAP base. %variables can be used here.
base = ou=users,dc=example,dc=com
```

Now you have to provide Dovecot with the search filters and the evaluation of the LDAP attributes for both the `userdb` request and the `passdb` request. Let's begin with the `userdb` request.

If your standard user in OpenLDAP has object class `inetOrgPerson` and typically has his email address in the `mail` LDAP attribute, the filter should look like this:

```
# Filter for user lookup. Some variables can be used (see
# http://wiki2.dovecot.org/Variables for full list):
#   %u - username
#   %n - user part in user@domain, same as %u if there's no domain
#   %d - domain part in user@domain, empty if user there's no domain
user_filter = (&(objectClass=inetorgperson)(mail=%u))
```

On an ActiveDirectory server, your `user_filter` will probably look similar to one of these two examples:

```
user_filter = (&(objectClass=Person)(sAMAccountName=%u))
user_filter = (&(objectClass=organizationalPerson)(sAMAccountName=%u))
```

Now you have to inform Dovecot which LDAP attributes to transfer to which Dovecot attributes. Please note that the allocation is performed according to the pattern `ldapattribute=dovecotattribute`, so from left to right: `ldapattribute` moves to `dovecotattribute`. Instinctively you would assume that the allocation goes from right to left, as is the case with variables.

If you have set up your LDAP according to the following schema, for example:

```
uid: klaus
mail: klaus.mueller@example.com
password: test
homeDirectory: /home/klaus
uidNumber: 1000
gidNumber: 100
```

you would import these LDAP attributes into Dovecot as follows:

```
# User attributes are given in LDAP-name=dovecot-internal-name list. The
# internal names are:
#   uid - System UID
#   gid - System GID
#   home - Home directory
#   mail - Mail location
#
# There are also other special fields which can be returned, see
# http://wiki2.dovecot.org/UserDatabase/ExtraFields
user_attrs = homeDirectory=home,uidNumber=uid,gidNumber=gid
```

Careful: to be precise, the syntax is not just `ldapattribute=dovecotattribute`, but consists of three parts: `ldapattribute=dovecotattribute=defaultvalue` is the full phrase. If you have decided, for example, that all Dovecot users should have an identical user and group ID of `100000`, it is pointless to save a user ID for all users in LDAP. Instead, you can have the required user ID returned "hard-coded" in the LDAP report:

```
user_attrs = homeDirectory=home,=uid=10000,=gid=10000
```

That means that`"nothing"' in LDAP is packed into the`uid` or `gid` Dovecot attribute, so it is given the default value `10000` – a rather unusual syntax, but I hope the examples will illustrate. And yes, the "leading equal sign" is not a printing error, because the omitted LDAP attribute is still listed there mentally.

Now you can easily go one step further: if the UID and GID are determined automatically for every user, why can't the same happen for the storage path to the user's emails. I recommend the following setting:

```
user_attrs = =home=/srv/vmail/%Ld/%Ln,=uid=10000,=gid=10000
```

User `klaus@example.org`, for example, will automatically receive his email data under `/srv/vmail/example.org/klaus` – and your LDAP contains only the email address and password, nothing more.

```
dn: cn=k.test,dc=example.com,ou=users,dc=heinlein-support,dc=de
objectClass: inetorgperson
objectClass: top
userPassword:: dGVzdA==
cn: k.test
mail: k.test@example.com
```

The drawback is that Dovecot calculates a new home directory if a user's email address changes, for example because he gets married. In that case, you have to make sure that the directory in the file system is changed at the same time as the email address. It is up to you to decide which version suits you better.

You can now configure the `passdb` request in the same way as the `userb` request, because there Dovecot only needs to verify the password. In most cases, the `pass_filter` will have the same structure as the `user_filter`:

```
pass_filter = (&(objectClass=inetorgperson)(mail=%u))
```

The password attribute makes its grand entrance in the evaluation of the `passdb` attributes:

```
pass_attrs =  userPassword=password
```

If you want to skip the SQL section below, you can move on to Section 5.7, which describes how to minimize the number of LDAP requests.

5.6. SQL databases

Like LDAP requests, SQL databases present no difficulties for Dovecot as authentication sources. Dovecot supports MySQL, PostgreSQL and SQLite databases, all of which are luckily configured in exactly the same way. Go to `10-auth.conf`, activate the prepared include of file `auth-sql.conf.ext` and deactivate any other methods that you do not need:

```
#!include auth-deny.conf.ext
#!include auth-master.conf.ext

#!include auth-system.conf.ext
!include auth-sql.conf.ext
#!include auth-ldap.conf.ext
#!include auth-passwdfile.conf.ext
#!include auth-checkpassword.conf.ext
#!include auth-vpopmail.conf.ext
#!include auth-static.conf.ext
```

`auth-sql.conf.ext`, which has now been activated, contains the configuration file of a `userdb` and a `passdb` request that refers to the configuration file with the SQL access data and request strings:

```
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
}
```

In file `dovecot-sql.conf.ext`, you first specify whether the database in question is a MySQL, PostgreSQL or SQLite database:

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

Use the `connect` parameter to set up the main access data for the SQL database. For MySQL or PostgreSQL, that would be

```
connect = host=sql.example.com dbname=virtual user=virtual password=foo
```

Use `port=` to specify a different port. If there are several SQL servers available in the cluster, make multiple `host=` entries in this `connect` parameter in order to achieve load

distribution and fail safety.

For access via a Unix socket, you may interpret the `host` parameter generously and enter the required path:

```
connect = host=/var/run/mysql.sock dbname=virtual user=virtual password=fo
```

If access is to SQLite, however, you should enter the path to the SQLite database directly:

```
connect = /etc/dovecot/authdb.sqlite
```

Now define the SQL query that reads the required attributes out of your SQL database. In the `password_query` (= `passdb` lookup) we are (as shown above) initially interested only in the user name and password:

```
password_query = SELECT userid AS user, pw AS password \
   FROM users WHERE userid = '%Lu'
```

Here is a slightly more complex example that manages the local part and the domain part in separate columns and also checks whether the user account is active:

```
password_query = \
  SELECT username, domain, password \
  FROM users WHERE username = '%Ln' AND domain = '%Ld' \
  AND active = 'Y'
```

You can use the placeholders `%n`, `%d` and `%u` that were described in Section 5.1. Make sure that, for reasons of uniqueness, you always use them in the lower-case version `%Ln`, `%Ld` or `%Lu`. Email addresses can be spelled in all manner of ways, and your SQL query would treat upper and lower case letters differently.

In the `user_query` (= `userdb` lookup) you have to set the home directory, UID and GID of the user. That is easy if the columns were named analogously to the Dovecot attributes:

```
user_query = SELECT home, uid, gid FROM users WHERE userid = '%Lu'
```

Alternatively you have to allocate the column names to the respective Dovecot attributes:

```
user_query = SELECT dir AS home, user AS uid, group AS gid FROM users wher
```

If you followed our idea and gave all email users the same user and group ID of `10000`, you can hard-code the required result straight into the SQL query and save yourself the effort of dragging the `uid` and `gid` columns around for every user and filling them with the right values:

```
user_query = SELECT home, 10000 AS uid, 10000 AS gid FROM users WHERE user
```

You could also have the home directory specified:

```
user_query = SELECT /srv/vmail/%Ld/%Ln AS home, 10000 AS uid, 10000 AS gid
```

However, if a user name changes, you also have to change the data directory on the hard drive accordingly (and on time) – so it is up to you to choose which method is more practical for you.

Later on in the book, when we set up quotas or specific proxy setups in the cluster, we will read out other attributes from the SQL database and use them in `password_query` and `user_query`. For now, the simple query shown here will be absolutely sufficient.

5.7. Performance optimization: prefetching with LDAP or SQL

In a heavily used system with multiple logins, you want to reduce the number of database or LDAP queries as far as possible. Dovecot's separation into a `passdb` request and a `userdb` request, howerver, generates two requests for user logins.

That can be improved. During the `passdb` request, Dovecot can already request and "pre-mark", in part or completely, the `uid`, `gid`, and `home` fields it will need for `userdb`, (*prefetching*).

If all three fields are available after the `passdb` query, Dovecot can omit the `userdb` query completely. That means you could halve your request volume here.[18]

Prefetching is not for LDAP auth binds with `auth_bind_userdn`

If you use the authentication binds described above to process your LDAP requests (e.g. to ActiveDirectory) and you have already informed Dovecot of the correct DN of the user attribute by means of `auth_bind_userdn`, Dovecot will not perform any `passdb` lookups. The prefetching described here in the `passdb` lookup can no longer take place, and Dovecot will have to perform the `userdb` request. That is not really a bad thing, because you have already halved your request volume.

For prefetching, assign the corresponding data to the `userdb` attributes in the `passdb` request. They now have to be named with prefix `userdb_` in the `passdb` request: `uid` becomes `userdb_uid`, `gid` becomes `userdb_gid` and `home` consequently becomes `userdb_home`. The following two examples make it clear. In LDAP, the `userdb` attributes can be assigned to the `pass_attrs` as follows:

```
# If you wish to avoid two LDAP lookups (passdb + userdb), you can use
# userdb prefetch instead of userdb ldap in dovecot.conf. In that case
# you'll also have to include user_attrs in pass_attrs field prefixed with
# "userdb_" string. For example:
pass_attrs = uid=user,userPassword=password,\
  homeDirectory=userdb_home,uidNumber=userdb_uid,gidNumber=userdb_gid
```

For values with a fixed definition (as shown in Section 5.5.4):

```
pass_attrs = uid=user,userPassword=password,\
  =userdb_home=/srv/vmail/%Ld/%Ln,=userdb_uid=10000,=userdb_gid=10000
```

And in MySQL, the `userdb` attributes can similarly be imported in advance by means of prefetching:

```
# If you wish to avoid two SQL lookups (passdb + userdb), you can use
# userdb prefetch instead of userdb sql in dovecot.conf. In that case
# you'll also have to return userdb fields in password_query prefixed with
# "userdb_" string. For example:
password_query = \
  SELECT userid AS user, password, \
  home AS userdb_home, uid AS userdb_uid, gid AS userdb_gid \
  FROM users WHERE userid = '%Lu'
```

Or with hard-coded values:[19]

```
password_query = \
  SELECT userid AS user, password, \
  /srv/vmail/%Ld/%Ln AS userdb_home, 10000 AS userdb_uid, \
  10000 AS userdb_gid \
  FROM users WHERE userid = '%Lu'
```

> **You will continue to require a `userdb` request!**
>
> You still need to configure the "actual" userdb query, because the receipt of an email via LMTP or `dovecot-lda`, and access to shared folders, automatically result in pure `userdb` requests (as described above) that have to work without a preceding `passdb` lookup and therefore without prefetched content. LDAP filters and SQL queries can be modified quickly by using cut & paste – you simply need to address the attribute names once with `userdb_` in the name and once without.

In order to ensure that Dovecot actually uses prefetching and omits unnecessary `userdb` queries, you first have to enter an *additional* specific `userdb` lookup in your `auth-ldap.conf.ext` or `auth-sql.conf.ext`. It must come *before* the actual `userdb` lookup so that Dovecot can cancel if it already has all the required values:

For LDAP, the configuration would be as follows:

```
passdb {
  driver = ldap

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

# "prefetch" user database means that the passdb already provided the
# needed information and there's no need to do a separate userdb lookup.
# <doc/wiki/UserDatabase.Prefetch.txt>
```

```
userdb {
  driver = prefetch
}

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

In the same way, the SQL configuration receives its own `userdb` lookup with prefetching:

```
passdb {
  driver = sql

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

# "prefetch" user database means that the passdb already provided the
# needed information and there's no need to do a separate userdb lookup.
# <doc/wiki/UserDatabase.Prefetch.txt>
userdb {
  driver = prefetch
}

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

5.8. Different or identical UIDs?

Real users in `/etc/passwd` have to have their own UID; for virtual users in SQL, LDAP or the Dovecot `passwd-file`, you should do without and give everyone the *same* UID:

- Identical UIDs for all email data are far easier to handle for you as an administrator, for example when you make changes to the data by means of shell scripts or want to restore backup data.
- Shared folders (see Chapter 9) are very difficult or even impossible for differing user IDs.
- A Linux system knowns only 65,353 different UIDs. That number could be too low.
- Dovecot has access to all users' email data anyway. While it is not true to say that a separation of user IDs has no benefits in terms of security, the benefits are pretty slight. To put it another way: I believe the benefits are too slight to make the additional effort worthwhile.

My recommendation is:

- Use a specific UID and GID for all email users and therefore for all email data. UID/GID `10000` has proved to be suitable.
- Under this UID/GID, set up a user/group called `vmail` in order to block this UID/GID and to ensure that a name is displayed in process lists and file listings:

```
flash:~ # groupadd -g 10000 vmail
flash:~ # useradd -u 10000 -g 10000 -s /bin/false vmail
```

Make sure that all users in the authentication process always receive UID and GID `10000`.

5.9. How not (!) to assign UID, GID and HOME

Section 5.5.4 (LDAP) and Section 5.6 (SQL) describe how you can have the actual values returned hard-coded as the result of the authentication request.

I urgently advise you not to try another method that is recommended by a variety of (mainly older) instructions. According to these instructions, you should solve the task using normal Dovecot parameters and set the UID/GID in the Dovecot configuration file `10-mail.conf` in the following manner:

```
mail_uid = 1001
mail_gid = 1001
```

In older Dovecot 1.x versions, it is `user_global_uid` and `user_global_gid`:

```
user_global_uid = 1001
user_global_gid = 1001
```

According to those instructions, you are also supposed to leave the user's home path empty, because you can define the path for the storage of email data via variables in `10-mail.conf` directly in `mail_location`:

```
mail_location = maildir:/srv/vmail/%d/%n/Maildir
```

I urgently advise you not to use this configuration! For the following reasons:

- `user_global_uid` and `user_global_gid` are obsolete and should/can no longer be used.
- In practice, there are small but important differences between receiving the 10000 ID via LDAP/SQL query as a "result" that accidentally is always the same and configuring it in Dovecot using the `mail_uid` parameter. The difference lies in the question which `root` or user permissions the Dovecot modules start with. In practice, the method using `mail_uid` threw up repeated difficulties, particularly with more complex issues such as shared folders, and required complicated workarounds and additional configuration steps.
- A user's home path in particular plays a role in several parts of the Dovecot configuration, not just in `mail_location`. You would therefore have to adapt the configuration (superfluously) in many areas. The Sieve plugin, for example, uses `$HOME` by default to calculate the storage path for scripts and the temporary working directory – and other modules do the same.
- It has emerged that it is much easier to set the home path *once* cleanly as the result of the LDAP/SQL query and then be able to work with `~` or `%h` the rest of the time. You can then leave many parameters in their default settings that are based on the home path. That is better and more convenient than putting the path together at various points using

`%d/%n`. And I have on occasion caught a module automatically accessing `$HOME` – and failing when `$HOME` was empty.

- The `auto:` mode and the migration of email formats while the system is running absolutely require a clearly specified home directory (see Section 7.5). If you instead rely on coding the storage path directly in `mail_location`, you will encounter problems sooner or later.

So forget this method – even if it is described in plenty of instructions and how-to guides.

5.10. Saving passwords: plain text or hash?

At first glance, it may seem sensible to store users' passwords in various databases not in plain text but rather in hashed form using the Crypt, MD5 or SHA algorithm. To be precise, a hash is not an encryption mechanism, as there is no way to work out the original value from a hash. Compare it to the cross sum of a number: 526 has the cross sum 13, but 13 will never return you clearly to 526. However, cross sums can have any starting value; in a cryptographic hash, it is highly likely that there is no second value with the same hash result. But that of course depends on the quality of the hash algorithm. It has been possible to break down MD5 far further than the theoretical 2^{64} probabilities, so that this algorithm is now considered broken and non-secure. SHA, in contrast, is still considered secure.

The advantage is that neither the administrator nor unauthorized third parties can easily obtain the plain text password. When migrating to a different IMAP software, however, this can become a lethal trap (see Chapter 19).

5.10.1. Authentication methods PLAIN/LOGIN

The transmission methods LOGIN and PLAIN transmit the password in plain text and should therefore be used only if secured with SSL/TLS. It is true that the user name and password are base64-coded before transmission, but that serves more to secure against potentially difficult special characters in the password (national special characters, quotation marks, backticks), so base64 does not really prevent reading by unauthorized parties. After all, base64 is not an encryption method, but rather a coding from which the original can be calculated at any time.

The difference between PLAIN and LOGIN is purely political in nature. While PLAIN has always been defined in RFC 4616 and transmits the user name and password in a single line, Microsoft surprised everyone many years ago by introducing the LOGIN procedure (which is not defined in the RFC) where the user name and password are transmitted base64-coded in *two* lines. Apart from that, there is no difference between PLAIN and LOGIN. One is RFC, one is Microsoft, and the fact that we now uselessly have both procedures is due to tactical power play by a software manufacturer. In practice, you will therefore always offer PLAIN and LOGIN in parallel.

Warning

In the past, support for LOGIN has occasionally been packeted separately in "silent protest"; that is the case in the `postfix-login` package, for example. In addition, the default configuration sometimes contains PLAIN but not LOGIN (even though the software could provide both).

5.10.2. Authentication methods CRAM/DIGEST

Unlike PLAIN/LOGIN, the CRAM procedures (*challenge response authentication method*) such as CRAM-MD5, CRAM-SHA1, CRAM-SHA256, or even APOP, which is only available for POP3, are better protected and make unauthorized tapping of the password during transmission almost impossible.

To this end, the server returns an individual session key for every login (*challenge*). The client and server then use this key to calculate a hash value for the password that is only valid for this specific session (*response*). An attacker can not use this value later on to log in; at the same time, he is unable to draw conclusions about the original password from the *response*.

As the server and client have to perform the same calculations in parallel, the server needs to have the plain text password in its database.

To put it another way: if the password is saved as a hash in a seemingly safe manner, the client can only log in for technical reasons using the insecure methods of LOGIN and PLAIN. Only if the server can access the password in plain text are all secure methods such as CRAM-MD5, CRAM-SHA1, CRAM-SHA256 or APOP possible. This is a fundamental problem that has nothing to do with Dovecot or with the choice of software in general.

But there are exceptions that should not really exist: some systems surprisingly offer CRAM-MD5 even though they only possess hashes of the passwords. These, however, are HMAC hashes.[20]

The explanation is as simple as it is sobering: In the challenge-response procedure, the first step is to hash the password with the respective hash procedure. For CRAM-MD5, the password `secret` becomes the HMAC-MD5 hash

```
f16f9cd57afad6931bff9508ef68ea2db1b62513b604cf995e4f882bed6d4f1a
```

The client then calculates this with the `<challenge>` from the server to produce the `<response>`. If CRAM-SHA1 is used, the client calculates the password hash in the same way using HMAC-SHA1.

If software saves every password as an HMAC-MD5 hash, that looks secure from the outside. But that is a serious miscalculation: if an attacker gains possession of the hash, he can use it to calculate all further steps of the challenge-response process and then log in correctly. Like all other parties, the attacker does not need the actual plain text password at all.

Using the HMAC-MD5 hash is therefore just as secure or non-secure as saving the password in plain text. To put it another way: the true password used by the client and server *is* the hash.

So it is true to say that challenge-response processes are only possible if the server knows the plain text password. And a public HMAC hash allows third parties to log in easily using the CRAM process.

Admittedly, saving only the hash values does provide a small advantage. If an attacker obtains these data, he can "only" use the challenge-response process and is unable to log in using plain text methods (such as logging on to a web frontend). This may be nice for users who use the same password for everything, be it mail servers, FTP accounts, online shops or leisure portals. But anyone who believes that such a multi-function password is not available to dozens of others in plain text is clearly beyond help.

In any case, there are also drawbacks to the saved HMAC-MD5 hashes, as processes based on other hash processes are no longer possible. CRAM-SHA is no longer possible if there is an HMAC-MD5 hash.

Anyone who instinctively brands the saving of plain text passwords as unsafe and believes that saving password hashes is the safe and steady way to go should pause and think about the procedure once again. That is why it makes sense to remember the following arguments when deciding how to proceed further:

- If all users have to use an SSL/TlS-secured connection to log in, ideally with the right certificate, the password is difficult to intercept even during plain text transmission in PLAIN/LOGIN. In this case, protection is increased by non-evaluable hash storage in the database, and there is no need to use CRAM procedures. It is then better not to use plain text passwords.
- If users can also log in without an SSL/TLS-secured connection (but why?), the plain text transmission of the password must be prevented. Stoving the password in plain text in the database would be the lesser of two evils.
- If only a hash is saved for fear that the administrator or attacker can read the password out of the database, login has to take place via PLAIN/LOGIN. That would result in an absurd situation: while the administrator, who has access to everything anyway, is not supposed to read the password, practically every intern or intruder to a major ISP or backbone operator can read said password every time there is a plain text login.

  Work colleagues or neighbors at the internet café are also given easy access to the password. Last but not least, the administrator could also sniff out the password (simply by using `tcpdump`), quite apart from the fact that nearly all mail server software (including Dovecot) can write the password directly to the logfile upon request. It is enough to write `auth_debug_passwords` into the Dovecot configuration.
- And there is another thing you should not forget: the administrator can read all of a user's emails anyway; he simply needs to look at the hard drive to do so. Considering this aspect, saving passwords in plain text changes nothing.
- Users should be encouraged to use different passwords for different providers and

services. It may be difficult to explain to some people, but then they should not get annoyed because the administrator now knows their "secret" password that they use on any number of other occasions.

As you can see, the whole situation depends on whether connections have to be via SSL/TLS or are also possible in plain text.

Again and again, I encounter systems with my customers that are operated without SSL/TLS. Often, this is because the firewall is supposed to monitor the network traffic on layer 7 for safety reasons (!) in order to protect the systems… The fact that this "safety" allows anyone to read the content of the transmitted data clearly demonstrates the absurdity of this argument. If the password is then sent unsecured every time and not even CRAM is used, the entire logic falls to pieces.

The only acceptable argument against saving passwords in plain text is the fact that an attacker will obtain *all* passwords if he manages to access the user database successfully. However, if the attacker obtains the MD5 hash it would be just as bad.

What is the actual risk? Is the danger from sniffers not far greater and more likely? Won't an attacker who gains access to all of the user data already have access to the authentication service or all the emails and therefore no longer require any passwords?

Important as it is to secure a system as well as possible, it is just as important to find out exactly *where* the danger lies. And this danger lurks during the normal and unsecured transport of data across the internet, comes from a sniffing attack by a neighbor's PC, or exists because of the generally low quality of passwords.

5.10.3. Setting up the CRAM process in Dovecot

If you decide to offer CRAM and save passwords in plain text, don't forget to modify the permitted authentication methods in `10-auth.conf`.

If passwords are saved as hashes, only PLAIN and LOGIN will usually be permitted:

```
# Space separated list of wanted authentication mechanisms:
#   plain login digest-md5 cram-md5 ntlm rpa apop anonymous gssapi otp
#   skey gss-spnego
# NOTE: See also disable_plaintext_auth setting.
auth_mechanisms = plain login
```

If the database contains plain text passwords, the permitted methods should be expanded as follows:

```
auth_mechanisms = plain login digest-md5 cram-md5 apop
```

5.10.4. PLAIN/LOGIN only secure via SSL/TLS

You can also consider removing PLAIN and LOGIN from the list completely. If you want to continue providing PLAIN or LOGIN, you should at least use `disable_plaintext_auth` to make sure that these two procedures are only permitted in conjunction with secure (TLS) connections (see Section 10.4):

```
disable_plaintext_auth = yes
```

That means that only CRAM procedures will be allowed on non-secure channels, while all procedures are available on secure ones.

5.10.5. How password hashes are saved

If you want to save the hash values of your passwords, Dovecot needs to know that they are hashes, and which procedure is used. For this purpose, Dovecot uses the procedure that is standard for LDAP and MySQL, where the hash method is placed in front of the password in curly brackets.

This example shows all the versions for password `test`:

```
{PLAIN}test
{CRYPT}4ZtRq3xJ6dR1s
{MD5}$1$47PZafRb$yaF4iBIaA8iCTCJV/JPnK1
{PLAIN-MD5}e8636ea013e682faf61f56ce1cb1ab5c
{LDAP-MD5}6GNuoBPmgvr2H1bOHLGrXA==
{SMD5}8jI00LUVTivwWdg2YYOv/tMxXKo=  (Salted)
{CRAM-MD5}e02d374fde0dc75a17a557039a3a5338c7743304777dccd376f332
{SHA}kGByAB793z4R5tK1eC9Hd/4Dhzk=
{SSHA}6pJi9UmtiW5DxPLSDKJN3B//5pG1HTnd  (Salted)
{SHA256}n4bQgYhMfWWaL+qgxVrQFaO/TxsrC4Is0V1sFbDwCgg=
{SSHA256}41MaIjmxca3/sgzedFCcZXn5s3afDLkvN5h6+UTjCHiy0PMt  (Salted)
{SHA512}jYR+AdIrqpafcfo2K03iHJ4Tx4grzqE7pcaorg1x/IyXAMguAIemXIs3vSn1NnR/KM
lnK+wcrndi0snza2AT8g==
{SSHA512}vx4qjIPiKrFyQrLCghRFNX/PlX1tABkLsoBItXgO4Hnl6W36fsa9svkxEKgX5Ul6W
A+BgxCOl8mhhTSrawkfplCN/3Y=  (Salted)
```

You can have the different hashes created conveniently by using the `doveadm pw` command:

```
flash:~ # doveadm pw -s sha256
Enter new password: secret
Retype new password: secret
{SHA256}rdsPXngmyFfXN20b2bwzwMVEeQourJYUSoryKxKYyUA=
```

By default, Dovecot assumes that the underlying passwords are base64-coded. If you have to fall back on hex-coding, you can specify the required password type:

```
flash:~ # doveadm pw -s sha256.hex
Enter new password: secret
Retype new password: secret
{SHA256.HEX}addb0f5e7826c857d7376d1bd9bc33c0c544790a2eac96144a8af22b1298c9
```

For more details on the saving of passwords and on hashing with salts, consult the corresponding Wiki page.[21]

You can use these syntaxes anywhere that Dovecot reads the password itself: Dovecot's own passwd-file `/etc/dovecot/users`, but also LDAP, MySQL or other databases.

Should you also have passwords with methods that are not marked explicitly in the prefix, you can specify the default method that Dovecot should assume in such cases in `10-auth.conf`. Files `dovecot-ldap.conf.ext`, `dovecot-mysql.conf.ext` and `dovecot-`

`dict-auth.conf.ext` contain the parameter `default_pass_scheme`, which you should set to `plain` or the hash procedure you use:

```
# Default password scheme. "{scheme}" before password overrides this.
# List of supported schemes is in: http://wiki2.dovecot.org/Authentication
default_pass_scheme = PLAIN
```

If the passwd-file is used, you can transmit the default scheme in the `passdb` module:

```
passdb {
  driver = passwd-file
  args = scheme=CRYPT username_format=%u /etc/dovecot/users
}
```

Dovecot nevertheless always looks at the password prefix first and, if in doubt, assumes the hash method specified there, so you can store various hashing processes in your database in parallel. This can be necessary, for example, if you transfer a lot of old user data with passwords in the MD5 format during a migration while keeping your own passwords saved in plain text or in an SHA hash. Mixed operation is therefore possible, as you can determine easily using `doveadm auth test`.[22]

5.10.6. Login name or email address as login?

If you already have experience of Postfix or other MTAs, you will be familiar with the difference between “real” and “virtual” domains. Postfix, for example, distinguishes whether a domain is specified in `$mynetworks` or in lookup tables `$virtual_alias_domains` or `$virtual_maps`.

The background to this procedure is the fact that traditional shell accounts whose data are stored in `passwd` and `shadow` always consist of a user name and never of an additional domain part. That can cause problems if you have to map accounts such as `info@` under a variety of domains at the same time.

The traditional way is to forward these domains to “real” accounts using the `virtual` maps. Older versions of the *Confixx* and *Plesq* software, for example, create innumerable accounts in the style of `web15p3`: 15th domain, 3rd mailbox.[23] Users have to enter these login data in their mail client as their user name for POP3/IMAP access.

Actually you can give the accounts any name you like – you could even simply give them numbers such as `1234567`, but that is not particularly pretty or easy to remember. In particular when you have to clear up and delete old domains and mailboxes, it will be difficult to navigate unless you maintain a database of allocations.

But there is a more convenient way: even though the `@` sign used not to be a permitted character in the user name of a shell account, all current clients have long been used to accepting a full email address as the login name. That means the owner of `info@example.com` can simply enter `info@example.com` in his mail client – and only needs to remember the password.

Such user names cannot be mapped using `passwd` and `shadow`, because the `@` sign is still not permitted there. In MySQL, PostgreSQL and LDAP, however, you can create corresponding accounts – just like in the passwd-file.

Using the login name, the corresponding authentication modules search the database/ directory for the data record that contains this name in its ID field and then access the password, user and group ID and the path to the email directories.

If you have to manage a number of different domains, MySQL, PostgreSQL and LDAP are therefore far superior to the traditional shell accounts as ways of storing user information.

In this case, make sure that `auth_username_format=%Lu` is set so that it transfers the entire email address to the authentication process. If only `%Ln` is set, Dovecot would only transfer the local part in front of the `@` sign and thereby chop up the login name.

Here are the arguments in a nutshell:

Email address as login name

Pros

- An email address is always unique; even if you as a provider have many customers and domains, conflicts cannot develop. There is no need to use number games on common names.
- Importantly, Sieve scripts run in the context of the email address; that is important for automatic out-of-office replies, where Sieve demands to see the user's own name in the *header-to:* field of the e-mail for security reasons – which is not possible if there is a difference between the login name and the email address. Using email addresses as the login name means it is not necessary to enter the email address manually when defining the Sieve filter (see Section 12.2).
- In shared folders, users can use email addresses (which they know) to set permissions in their ACLs. If you use login names, users have to know and enter the login names of the users they want to grant access to. That may be possible at companies, but in a university environment, the login names of students are usually not known – and often contain the registration numbers that may be used when publishing exam results.

Cons

- Users can change their names, for example when they get married. They are unlikely to get annoyed because they have to change the login names for their email programs. Postmasters, however, would then have to provide a mechanism that renames email home directories if the storage path is automatically calculated from the login name (if `mail_location = maildir:/srv/vmail/%Ld/%Ln/Maildir` is used, for example).

Login names with only a local part

Pros

- If the user's name or the server's email domain change, the login name still remains the same.
- In Windows-based environments, a login name is often already involved.

Cons

- Auto responders in Sieve scripts require email addresses to be entered manually in the Sieve script.
- Auto responses from Sieve are usually generated with the user's login name and not his email address. As a result, additional work may be required later on in Postfix using `sender_canonical_maps`.

When does which solution make sense

You can use the following principles as a rule of thumb:

- For providers, it often makes sense to use email addresses as login names. After all,

there is no connection here to Windows IDs.

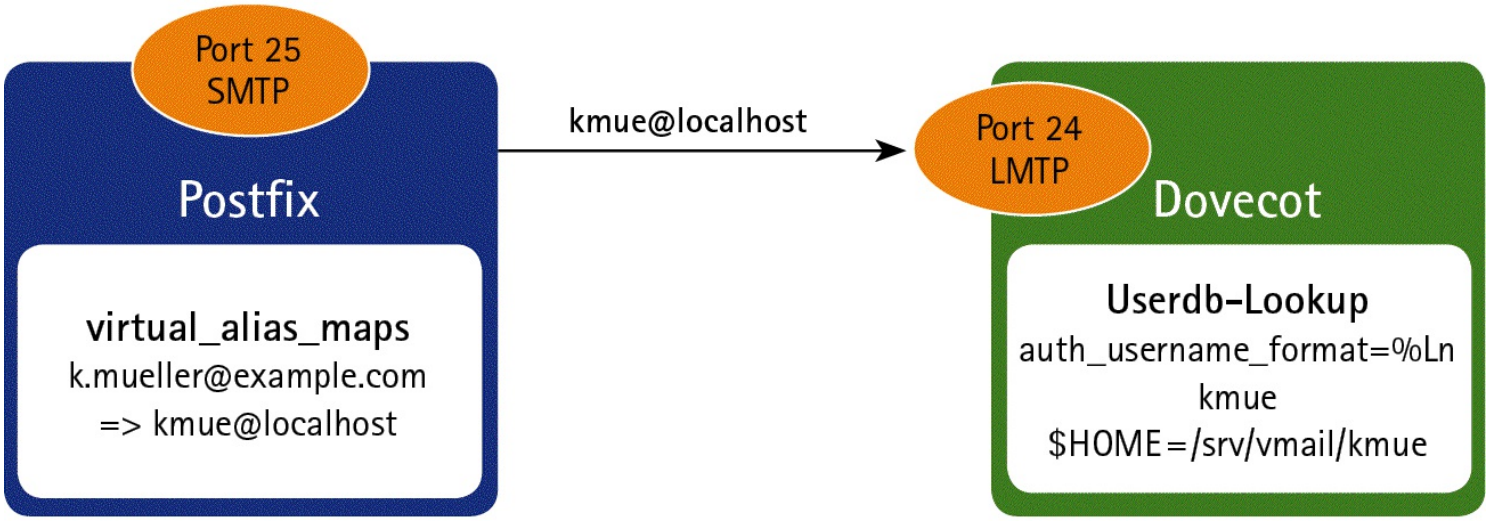
- For universities, there are good reasons for both versions. Email addresses are more user-friendly because of Sieve and shared folders, but login IDs are often easier to maintain.
- Businesses with a strong Windows structure and AD authentication require login names.
- Businesses with a free IT infrastructure should try to use email addresses, which are more user-friendly.

5.10.7. Practical suggestion: different userdb and passdb queries!

If you decide to work with login names, you should follow this advice that has proven remarkably useful in practice:

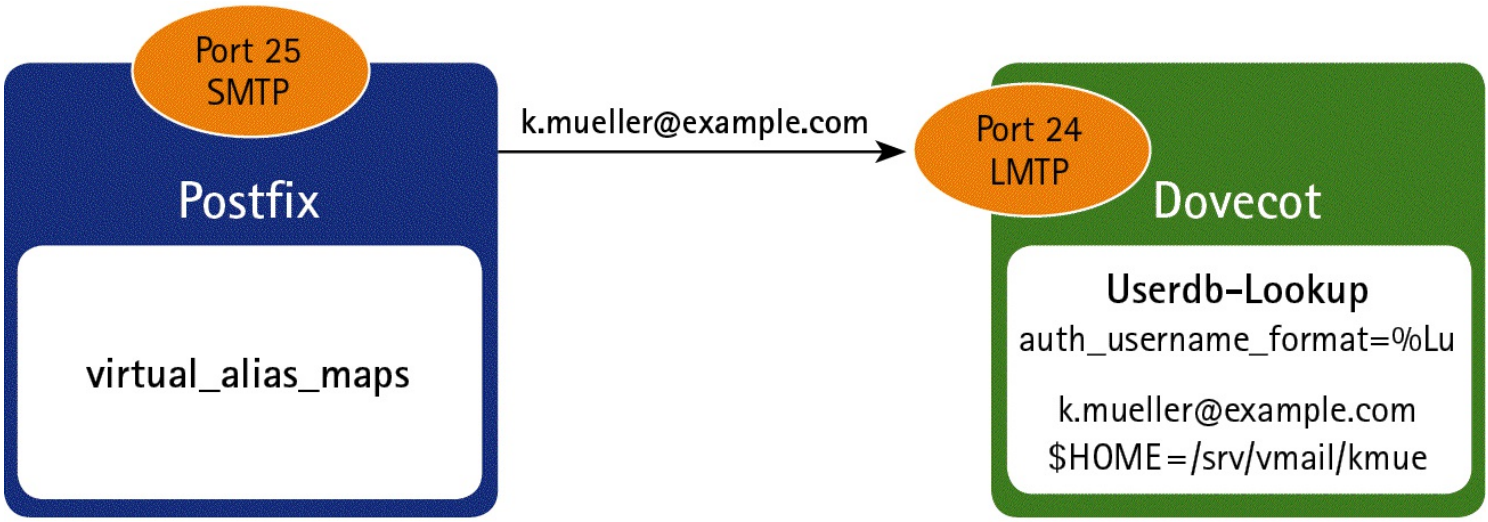
Plenty of instructions suggest you rewrite the email address to `<loginname>@<hostname>` at the mail server level (=Postfix) (e.g. using a `virtual_alias_maps` query in Postfix) so that these emails are routed to Dovecot in a "tidy" fashion. However, that would require Dovecot (as described above) to use `auth_username_format` to access `%Ln` and evaluate only the local part `<loginname>` (Figure 5.2). That in turn involves the drawbacks described above (Sieve scripts, shared folders).

Figure 5.2. In many instructions, conversion is performed with a Postfix virtual map.



It is much better to continue to route the email and email address to Dovecot in their original form as shown in Figure 5.3, i.e. route the email domains used to Dovecot via `relay_domains` and `transport_maps` as described in Section 6.3.

Figure 5.3. Easier, and clearer in the log files: route emails directly to their destination with their email addresses.



Allow the `passdb` query to be matched only to the user's login name; after all, the user should only be able to log in with this login name. In our LDAP example, that would be:

```
# Filter for user lookup. Some variables can be used (see
# http://wiki2.dovecot.org/Variables for full list):
#   %u - username
#   %n - user part in user@domain, same as %u if there's no domain
#   %d - domain part in user@domain, empty if user there's no domain
pass_filter = (&(objectClass=inetorgperson)(uid=%u))
```

But be careful. Use `%u` and not `%n`. Otherwise there is a danger that the local part of an email address unintentionally matches a `uid` of the same name.

Then you should set up the `userdb` query so that you record (only) the email address, so that LMTP can determine the user record on the basis of the email address – in LDAP, for example, with a `user_filter` in the style of:

```
# Filter for user lookup. Some variables can be used (see
# http://wiki2.dovecot.org/Variables for full list):
#   %u - username
#   %n - user part in user@domain, same as %u if there's no domain
#   %d - domain part in user@domain, empty if user there's no domain
user_filter = (&(objectClass=inetorgperson)(mail=%u))
```

If (for whatever reason) you want to allow delivery via the local part of the email address, you could choose:

```
user_filter = (&(objectClass=inetorgperson)(|(mail=%u)(uid=%u)))
```

However, that would make it difficult to calculate the home directory automatically on the basis of `%d` and `%u` because the local part of the email address and the login name are different, so different paths would result every time. You can now store and evaluate the home directory in the LDAP data as a fixed value. You could also have the home directory calculated by having the `uid` from the found LDAP object replaced via placeholder `%$`:

```
# There are also other special fields which can be returned, see
# http://wiki2.dovecot.org/UserDatabase/ExtraFields
user_attrs = uid=home=/srv/vmail/%$  [...]
```

If you use SQL as your database system, you have to adapt your configuration in the same way as shown in these LDAP examples.

That way, you route your emails transparently to their destination on the basis of the email address, and there is no need for additional configuration, database queries and address conversions in the upstream MTA. Also, and this is important, you make sure that the Sieve script is called up by the LMTP for the context of the email address as well. Any `vacation` script then also "sees" itself in the `mail header-to:` field and can trigger the required auto response, so that all these problems no longer exist.

User logins, by contrast, which are first validated using the `passdb` query, are only usable for login names and not for email addresses, so there can be no unwanted confusion.

Use `doveadm auth` in all versions to check your configuration.

5.11. Master user

The *master user* is the skeleton key amongst access options. With the master user and the corresponding master password, you can log on under the identity of any user – which can be very helpful for debugging purposes ("what does this look like to the user?") or for migration purposes (reading out or saving the email data) within the applicable data protection guidelines. Of course, this account is incredibly valuable, so you should choose the password very carefully.

In order to activate the master user, you first have to go to `10-auth.conf` and define the `auth_master_user_separator`, i.e. the special character that separates the user name and the master user name. Usually, the asterisk `*` is used in this case, because it is not used in email addresses or login names and no conflicts can result:

```
auth_master_user_separator = *
```

Then you have to go to the end of `10-auth.conf` and, in addition (!) to your existing authentication, activate the configuration snippet `auth-master.conf.ext`:

```
# <doc/wiki/UserDatabase.txt>

#!include auth-deny.conf.ext
!include auth-master.conf.ext

# !include auth-system.conf.ext
#!include auth-sql.conf.ext
#!include auth-ldap.conf.ext
!include auth-passwdfile.conf.ext
#!include auth-checkpassword.conf.ext
#!include auth-vpopmail.conf.ext
#!include auth-static.conf.ext
```

If you take a look at the `auth.master.conf.ext` file, you will see that it is simply a standard `passdb` query with its own (separate) passwd-file with `master=yes` set as a special flag.[24] That allows Dovecot to recognize that it must extract the user name via the `master_user_separator` in the event of hits in this passwd-file and then, after successful login, use a suitable `userdb` query to determine and assume the identity of the actual user:

```
passdb {
  driver = passwd-file
  master = yes
  args = /etc/dovecot/master-users

  # Unless you're using PAM, you probably still want the destination user
  # to be looked up from passdb that it really exists. pass=yes does that.
  pass = yes
```

```
}
```

If you now create a master user in `/etc/dovecot/master-users`

```
adminuser:{PLAIN}secretsecret::::::
```

and restart Dovecot, you can log in as `username*mastername`, e.g. as `test@example.com*adminuser`:

```
flash:~ # telnet localhost 143
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
* OK [CAPABILITY IMAP4rev1 LITERAL+ SASL-IR LOGIN-REFERRALS ID ENABLE IDLE
a login test@example.com*adminuser secretsecret
a OK [CAPABILITY IMAP4rev1 LITERAL+ SASL-IR LOGIN-REFERRALS ID ENABLE IDLE
a logout
* BYE Logging out
a OK Logout completed.
Connection closed by foreign host.
```

By the way, you can also use the `doveadm` command to apply a master login. It doesn't make any sense, but it demonstrates nicely how Dovecot proceeds. At the end of the passdb lookup, the user name is set again, and Dovecot goes on with that user name:

```
flash:~ # doveadm auth test test@example.com*adminuser
Password:
passdb: test@example.com*adminuser auth succeeded
extra fields:
  user=test@example.com
```

5.12. Some extras

Dovecot offers additional control options in the form of `extra fields`, which you can use individually for each user during the authentication process. A distinction is made between *passdb extra fields* and *userdb extra fields* – depending on whether a `passdb` or `userdb` lookup reads out the attributes.

5.12.1. Passdb extra fields

The `passdb extra fields` typically refer to a user's login:

`user`
: resets the user name for the subsequent processes, including changing upper and lower case letters.

`allow_nets`
: restricts the permitted user login to the specified areas of the network.

`proxy` and `proxy_maybe`
: forwards the user's connection to another backend server (for more information, see Section 16.2).

`host`
: defines the target host for `proxy`/`proxy_maybe` or creates a login referral.

`nologin`
: prevents the user from logging in.

`nodelay`
: prevents a delayed response in the event of incorrect login data (which actually provides protection from brute force attacks).

`nopassword`
: the user may log in with an empty password.

The precise definition of passdb extra fields is different for different authentication backends. If the passwd-file is used, the attributes are listed in the last column and separated by spaces:

```
user:{plain}pass:10000:10000::/home/user::proxy=y host=127.0.0.1
```

If LDAP is in use, the LDAP attribute is assigned to the corresponding extra field in the `pass_attrs`:

```
pass_attrs=uid=user, userPassword=password, ProxyAttribute=proxy, hostName
```

The definition of the `password_query` in SQL is performed in the same way:

```
password_query = SELECT userid as user, password, \
    'Y' as proxy, host \
    FROM users WHERE userid = '%u'
```

The passdb extra fields are returned as the result of the `passdb` query in the event of a `doveadm auth test` command:

```
flash:~ # doveadm auth test klaus@example.com
Password:
passdb: klaus@example.com auth succeeded
extra fields:
  user=klaus@example.com
  host=10.152.188.113
  proxy_maybe=yes
```

However, you require the user's password for this purpose. In a simpler but slightly different format, you can also debug the passdb extra fields with the `doveadm auth lookup` command:

```
flash:~ # doveadm auth lookup klaus@example.com
passdb: klaus@example.com
  user       : klaus@example.com
  host       : 10.152.188.113
  proxy
```

5.12.2. Userdb extra fields

The options provided by the userdb extra fields are not usually relevant in practice, simply because they mainly offer ways to influence shell users with a “real login”:

`nice`
: sets the specified nice value for the Dovecot proceses of this user.

`chroot`
: creates a *chroot* in the specified directory for this user and overwrites any global `mail_chroot` in the Dovecot configuration.

`system_groups_user`
: defines user names with groups read out from `/etc/group`. Shared folders for system groups can be implemented on this basis.[25]

`userdb_import`
: makes it possible to save additional extra fields as a tab-separated field. That allows multiple attributes to be saved in one field instead of maintaining multiple separate fields for them. That is helpful in rigid database structures such as SQL.

`uidgid_file`
: sets the user’s UID and GID on the basis of the file specified here.

In principle, plenty of settings in the Dovecot configuration can be overwritten using userdb extra fields, which in turn makes this function rather interesting. The following are particularly relevant in practice:

`mail_plugins`
: overwrites the Dovecot configuration `mail_plugins` and thereby enables individual features for users.

`mail`
: overwrites the Dovecot configuration `mail_location` and thereby enables individual storage paths, but also storage formats (Maildir/mdbox) – this can be useful in step-by-step migrations, where the conversion is to be performed user by user.

`quota_rule`
: overwrites the generic quota rules for this individual user. For details, see Section 11.5.

If passwd-file is used, userdb extra fields are also listed in the last column of the passwd-file and separated by spaces – but they have to be equipped with the prefix `userdb_`.

```
user:{plain}pass:10000:10000::/home/user::userdb_mail=maildir:~/Maildir us
```

If LDAP is used, the attributes are allocated in `user_attrs` as usual:

```
user_attrs = homeDirectory=home,uidNumber=uid,gidNumber=gid,quotaDovecot=q
uota,mail_plugins
```

The definition in the `user_query` in SQL is similar:

```
user_query = SELECT home, uid, gid, \
   '*:storage=100M' as quota_rule, mail_plugins \
```

```
    FROM users WHERE userid = '%u'
```

Use the `doveadm user` command to read out a user's `userdb extra fields`:

```
flash:~ # doveadm user klaus@example.com
field   value
uid     10000
gid     10000
home    /srv/vmail/example.com/klaus
mail    maildir:~/Maildir
quota_rule      *:storage=200M
```

5.13. A list of all users – the iterate query

In some cases, Dovecot has to have specific tasks performed in sequence for every existing user – e. g. the `doveadm purge` command (see Section 7.3.3) or when you set up a replication of the email directories (see Section 16.4). But it can also be helpful for you as an administrator to easily create a list of all accounts.

For accounts in the system (`/etc/passwd` alias `auth-system.conf.ext`) or in the passwd-file (`/etc/dovecot/users` alias `auth-passwdfile.conf.ext`) Dovecot has no problems generating the user list, as it can simply read out the list of users. For SQL or LDAP queries, it depends on their actual design. Particularly if you use authentication binds in LDAP/AD, Dovecot is unable to obtain a list of user accounts without access permissions.

For this reason, Dovecot has a special SQL or LDAP query, the *iterate query*, to determine a user list there. Here you can also modify the `SELECT` query or the LDAP search filter if necessary. Make sure that these queries are set and working.

For SQL, you will find a prepared entry right at the end of the `dovecot-sql.conf` file:

```
# Query to get a list of all usernames.
iterate_query = SELECT username AS user FROM users
```

And in LDAP, also at the end of `dovecot-ldap.conf`:

```
# Attributes and filter to get a list of all users
iterate_attrs = uid=user
iterate_filter = (objectClass=posixAccount)
```

Transfer your filters here that you already found for `user_filter` and/or `pass_filter`.

Use the `doveadm user '*'` command to see if everything works. Make sure you place quotation marks or inverted commas around the asterisk, otherwise your shell will use a list of all files in the current directory.

```
flash:~ # doveadm user '*'
klaus@example.com
susi@example.com
micha@example.com
```

If, for some reason, you have difficulties allowing Dovecot access to your user list, you need not despair. Even though it is convenient to run the `doveadm` command via the `-A` parameter as a loop across all users, you still have other ways to generate such a user list in the context of a shell script as long as there is a connection between the directory name and the email address/login name:

```
# Adapt path to your email directories
```

```
MAILPATH=/srv/vmail

cd $MAILPATH

# The script assumes /srv/vmail/example.com/susi
for DOMAIN in * ; do
    cd $MAILPATH/$DOMAIN
    for USER in * ; do

          echo $USER@$DOMAIN
# insert your tasks here.

    done
done
```

There are some small restrictions, in that users without an email directory are not included (but what kind of users are they?) and that deleted users whose email directory has not yet been deleted are also included in the list. However, `doveadm` will not find these users and will therefore generate an error and terminate the operation. That means there are no serious consequences.

While we're on the subject: this is a good way to identify and delete dead email directories. You should, however, check the following script for your scenariou carefully before letting it run automatically. It is a good idea to comment out `rm -rf` at first (as demonstrated) or equip it with an `echo` to check the output of the script.

```
# Adapt path to your email directories
MAILPATH=/srv/vmail

cd $MAILPATH

# The script assumes /srv/vmail/example.com/susi
for DOMAIN in * ; do
    cd $MAILPATH/$DOMAIN
    for USER in * ; do

      if `doveadm user $USER@$DOMAIN` ; then
           echo $USER@$DOMAIN exists
      else
           echo $USER@$DOMAIN does not exist
#          rm -rf $MAILPATH/$DOMAIN/$USER
      fi

    done
done
```

---

[16] Dovecot is also able to perform Kerberos authentication in various ways: GSSAPI or using `pam_krb5`. More information is provided in the Dovecot Kerberos readme: http://wiki2.dovecot.org/Authentication/Kerberos

[17] If you want to know more after reading this book, you will find documentation on http://wiki2.dovecot.org/AuthDatabase.

[18] To be precise, there are other, unspecific attributes such as quota rules that you can also prefetch. I will illustrate that later on, for example in Section 11.8.2.

[19] You will find the configuration examples cited here prepared in `/etc/dovecot/dovecot-ldap.conf.ext` and `/etc/dovecot/dovecot-sql.conf.ext`.

[20] HMAC is short for *Hash Message Authentication Code*; HMAC-MD5 is calculated slightly differently than the "normal" MD5. Both procedures are therefore not identical, even though they are both often referred to as "MD5 hash" (see http://en.wikipedia.org/wiki/HMAC).

[21] http://wiki2.dovecot.org/Authentication/PasswordSchemes

[22] If you save in plain text, you have also activated the CRAM procedures in the `auth_mechanisms` as described above. Users with an MD5 password will be offered a CRAM login by Dovecot, but they would not be able to use it successfully. That is usually not a problem, because the desktop mail clients of the affected users have usually grown over time, are configured for PLAIN or LOGIN and therefore simply ignore the CRAM offered by the server. In practice that works very well. But be careful. If your webmailer tries to determine the server's method automatically, it will of course fail in such cases because it tries to use CRAM for *all* users.

[23] That does not necessarily speak for the quality of the software.

[24] You can query the master user in a similar manner using SQL, LDAP or other methods. But why?

[25] For reasons of space, and because email accounts for shell users are rarely relevant in practice, we will not discuss this topic any further here.

Chapter 6. How to configure Postfix as a relay in front of Dovecot

If you want emails to be delivered to the user's email storage, there are several ways of doing this:

As long as Dovecot saves emails in an open storage format such as mbox or Maildir, the locally running MTA (i.e. Postfix, Exim, Qmail, Sendmail etc.) can save the emails themselves to the email storage of the respective user. But that means the MTA has to know what users there are, which location the emails are to be saved in, and with which user ID.

That is fairly simple if you provide emails for local shell users from `/etc/passwd`. However, as soon as you have users in different authentication sources, it is difficult (and unnecessary) to build a copy in the MTA of the authentication that already exists for Dovecot. And of course you would be unable to use Dovecot-specific storage formats such as mdbox (see Section 7.3).

It is better if the MTA can transfer the emails to be saved to the appropriate IMAP server, which can then take care of saving the email in the correct format on the basis of its own knowledge. In this case, the MTA just needs to know what email addresses there are, but it has nothing further to do with the details of the user.

Configuration is much simpler, and storage formats such as mdbox become possible; but even with the open mbox and Maildir formats, the Dovecot-specific storage process is better able to update the in-house caches. Advanced tricks such as proxy operation on distributed servers (Section 16.2) can now also be performed using Dovecot-specific processes. Lastly, this is the only way to securely implement Sieve filters (Chapter 12) and quota rules (Chapter 11).

As long as you do not have to rely on delivering emails via special MDAs such as `procmail` or similar, you should probably choose to transfer the emails directly to Dovecot and have them stored by Dovecot itself.[26]

But even if you leave the saving of emails up to Dovecot, there are still two ways to transfer the emails:

Version 1

Emails are transferred to Dovecot program `dovecot-lda` (which used to be called `deliver`) via pipe and call parameters.

Dovecot contains the `dovecot-lda` (*local delivery agent*) program which receives emails via pipe, determines the correct storage path for the user and saves the email in the users mail storage. In earlier versions, `dovecot-lda` was named `deliver` and it

appears under this name in instructions and how-to guides on the internet. A symlink from `deliver` to `dovecot-lda` ensures that both names work now.

However, it is not particularly efficient to deliver emails via `dovecot-lda`, because a separate process has to be started for every email; also, emails with multiple recipients have to be transferred multiple times and saved individually. Due to a lack of protocol and bidirectional communication, it is very difficult to return errors and feedback to Postfix. Dynamic recipient validation (Section 6.4) is not possible (and that would be an extremely desirable feature!). In the case of virtual users with individual IDs, problems quickly develop in the access permissions for email folders, as Postfix does not want to call up the `dovecot-lda` process with `root` permissions and therefore has difficulties receiving the IDs of the recipient users.

Version 2

It is therefore better to transfer emails to Dovecot's LMTP daemon via the LMTP protocol.

This version is efficient and fast, because the daemon has already been started; emails with multiple recipients are transmitted only once along with the recipient list, just like for SMTP. There are problems with file access permissions. In addition, the LMTP daemon could be addressed directly from the SMTP relays via TCP/IP, so there is no need for a (resource-consuming) detour via a locally running SMTP server, which then has to pipe the emails to `dovecot-lda`.

I highly recommend you use the method involving the `lmtpd`. Piping emails to `dovecot-lda` usually brings no benefits but considerable disadvantages.

However, as there are plenty of instructions circulating on the basis of the `dovecot-lda`, and as I regularly encounter old Dovecot 1.x versions that are not familiar with `lmtpd`, I have included the configuration of both versions.

6.1. Delivery via dovecot-lda

Enter the `dovecot-lda` call as the specified transport method in the Postfix `master.cf`. At the end of it, you will find several pipe calls and should add the following:

```
dovecot   unix  -       n             -       5       pipe
  flags=DRhu user=vmail:vmail argv=/usr/lib/dovecot/dovecot-lda
  -f ${sender} -d ${recipient}
```

The example assumes that the user and group name `vmail` is used for all users in Dovecot, so that Postfix can call up the `dovecot-lda` program as user `vmail` from the start. Depending on the distribution, or if you have compiled Dovecot yourself, you may have to modify the path to `dovecot-lda`. Make sure you do not set the number of parallel `dovecot-lda` processes too high in the last-but-one column. It will do you no good to run too many `dovecot-lda` jobs in parallel; at some point, things just don't get any faster. Five parallel processes should be a workable average, but of course that depends on the email system and its I/O performance.

In this example, Postfix transfers variable `${recipient}`. That corresponds to the full email address of the recipient, i.e. `klaus@example.com`. If you work internally with only the user IDs without the domain part (e.g. with accounts in `/etc/passwd`), you *could* in principle just have the local part of the email address transferred here, i.e. `klaus`. In this case, you *could* fall back on `${user}` instead of `${recipient}`, or ideally even `${mailbox}`, because that would also transfer any email extensions, i.e. `klaus+extension` (for more tricks, see Chapter 13):

```
dovecot   unix  -       n       n       -       5       pipe
  flags=DRhu user=vmail:vmail argv=/usr/lib/dovecot/dovecot-lda
  -f ${sender} -d ${mailbox}
```

When I say you *could* use `${user}` or `${mailbox}`, that does not mean I recommend it, because there is a far more elegant way: stick with the full email address with `${recipient}` (as described above) and instead use the parameter `auth_username_format=%Ln` within Dovecot (Section 5.1).

Postfix will forward the email with its entire address to Dovecot, and Dovecot itself can then decide how best to map this within the Dovecot authentication. That way, you will remain flexible for configuration and strategy changes, e.g. if you switch from `dovecot-lda` to LMTP, where you always transfer the full email address to Dovecot and have to use the method involving `auth_username_format`.

People often overlook in practice that this method only allows one recipient to be transferred per pipe call, because the number of call parameters for `dovecot-lda`is highly limited by its very nature. Many admins do not notice this for a long time, because test emails are typically

sent to one recipient and therefore do not allow the error to occur. An email sent to multiple envelope recipients will however remain in the P queue with a temporary error:

```
-Queue ID- --Size-- ----Arrival Time---- -Sender/Recipient-------
80B2513F813      265 Fri Aug  2 19:10:03  root@dovecottest2.site
                                 (mail system configuration error)
                                     klaus@example.com
                                     susi@example.com
```

The Postfix log files show that this is related to the `D` flag that was defined in `master.cf` in the pipe call to the `dovecot-lda` process (`flags=DRhu`):

```
2013-08-02T19:10:03.534336+02:00 dovecottest2 postfix/pipe[2329]: warning:
```

Flag `D` causes Postfix to add an individual `Delivered-To:` header line to the email it is supposed to save in order to save the `Envelope-To:` address. Naturally that is not compatible with a multiple recipient email (see `man pipe`). On the other hand, there would be problems with multiple recipient emails even without the `D` flag, because the shell does not know an unlimited number of call parameters and it would no longer be possible to call the `dovecot-lda` command if there are too many recipients.

You therefore need to instruct Postfix to use only one recipient per delivery; an email with multiple recipients would then result in multiple separate deliveries in sequence. If you used this method (as described above) to set up a transport method called `dovecot:`, you should make the following entry in the Postfix `main.cf`:

```
dovecot_destination_recipient_limit=1
```

You can use the transport method `dovecot:` as the routing destination in a Postfix `transport_maps`:

```
example.com     dovecot:
```

I want to emphasize once again that I am only describing this method because there are some very old installations and for the purpose of completeness. Wherever possible, you should choose the LMTP method instead.

6.2. Delivery via Dovecot via LMTP

First you should activate the Dovecot LMTP daemon; in Debian/Ubuntu-based distributions, make sure you have also installed the `dovecot-lmtpd` package. Other distributions already contain the `lmtpd`.

In `10-master.cf`, a variety of Dovecot listeners are already prepared in section `service lmtp`. This entry would open a Unix socket under file name `/var/run/dovecot/lmtp`:

```
service lmtp {
  unix_listener lmtp {
    #mode = 0666
  }
}
```

However, we prefer to operate Postfix in a chroot environment under `/var/spool/postfix`, so Postfix would not be able to reach this socket.

Instead of changing the chroot settings in Postfix, you should simply open an additional LMTP socket under `/var/spool/postfix/private/lmtp-dovecot` and add the following to `10-master.cf`:

```
service lmtp {
  unix_listener lmtp {
    #mode = 0666
  }

  unix_listener /var/spool/postfix/private/lmtp-dovecot {
    # mode = 0666
    user = postfix
    group = postfix
  }

}
```

> **Do not assign a socket name twice**
>
> As Postfix already uses `/var/spool/postfix/lmtp` to address its own `lmtp` module, the Dovecot socket must have a completely different name – in this example, we have used `lmtp-dovecot`!

The most flexible method, however, is to open a TCP/IP socket instead of a Unix socket for this purpose. There will be no problems with chroot environments or with access permissions and user names:[27]

```
service lmtp {
  unix_listener lmtp {
    #mode = 0666
  }

  # Create inet listener only if you can't use the above UNIX socket
  inet_listener lmtp {
    # Avoid making LMTP visible for the entire internet
    address = 127.0.0.1
    port = 24
  }
}
```

After restarting Dovecot, you should find yourself with a functioning LMTP port:

```
flash:~ # lsof -i :24
COMMAND  PID USER   FD   TYPE  DEVICE SIZE/OFF NODE NAME
dovecot 9735 root   33u  IPv4 1818944      0t0  TCP localhost:24 (LISTEN)

flash:~ # telnet localhost 24
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
220 flash.heinlein-support.de Dovecot ready.
quit
221 2.0.0 Client quit
Connection closed by foreign host.
```

Whether you use a Unix socket or a TCP/IP socket: once Dovecot is ready to start, we can take care of the configuration of Postfix as the upstream email relay.

6.3. Configuration

There are plenty of instructions and how-to guides on the internet describing how to install Postfix as the email relay in front of Dovecot. Unfortunately, a lot of them are questionable in terms of content and therefore not to be recommended. The methods described there work "somehow", but in many areas do not correspond to the basic structure of Postfix. Often, they will generate complications that in turn have to be remedied laboriously using workarounds.

The problem is that many of these instructions tell you to enter the required domains in `$mydestination` in Postfix. However, Postfix logic dictates that email domains and host names should be entered there only if Postfix itself is the final physical destination of the routing of the email. For all domains in `$mydestination`, Postfix expects the users listed after to be found in `/etc/passwd` and `/etc/aliases`. To be more precise: Postfix checks the recipients in the `$mydestination` domains using `$local_recipient_maps`, which by default however evaluates a glibc call via `/etc/passwd`:

```
flash:~ # postconf local_recipient_maps
local_recipient_maps = proxy:unix:passwd.byname $alias_maps
```

This method can be the right one if you actually operate Dovecot with local shell accounts in `/etc/passwd` and you want Postfix to provide "final" storage for the email; in other words, if you install Dovecot, make virtually no changes to the standard configuration and save emails in the mbox format, for example, in `/var/mail/user` and `/home/user`. But when and where does that still happen?[28]

As a rule, Dovecot users can be found in `/etc/dovecot/users` or are wholly swapped out to SQL or LDAP. In that case, Postfix has nothing to do with the user data and would not even be able to store the emails itself.

Many instructions therefore tell you to set `local_recipient_maps` to empty, which completely cancels out the recipient verification.[29] In order to allow Postfix to save the seemingly local email, these instructions often want the LMTP call towards Dovecot or the transport method `dovecot` defined in `master.cf` to be used as `mailbox_command`:

```
mailbox_command = lmtp:[127.0.0.1]
```

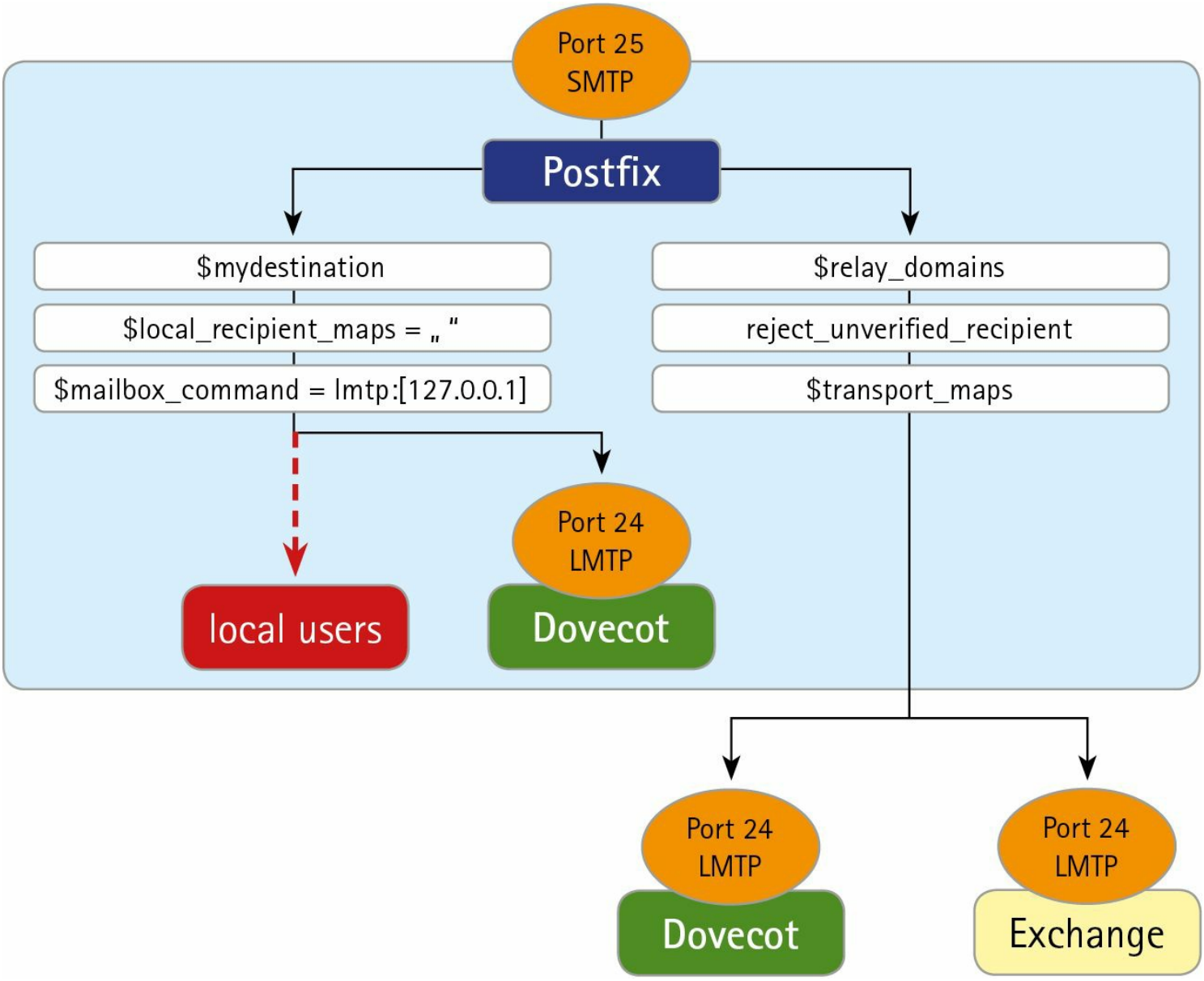
or

```
mailbox_command = dovecot:
```

The problem is that this completely cancels out the local email system. Shell accounts in `/etc/passwd` no longer work, and local accounts are unable to receive emails. However, this is often essential with regard to cron jobs, monitoring jobs and other details. In quite a few of

these broken setups, I can show that important system alerts go unread every day, because `root@localhost` and `wwwrun@localhost` is also delivered to Dovecot via LMTP – and these important system accounts are often not even set up in the user data there, so that these important messages are lost (see Figure 6.1).

Figure 6.1. The local email operation is disconnected – and Postfix believes it is responsible for the Dovecot domain.

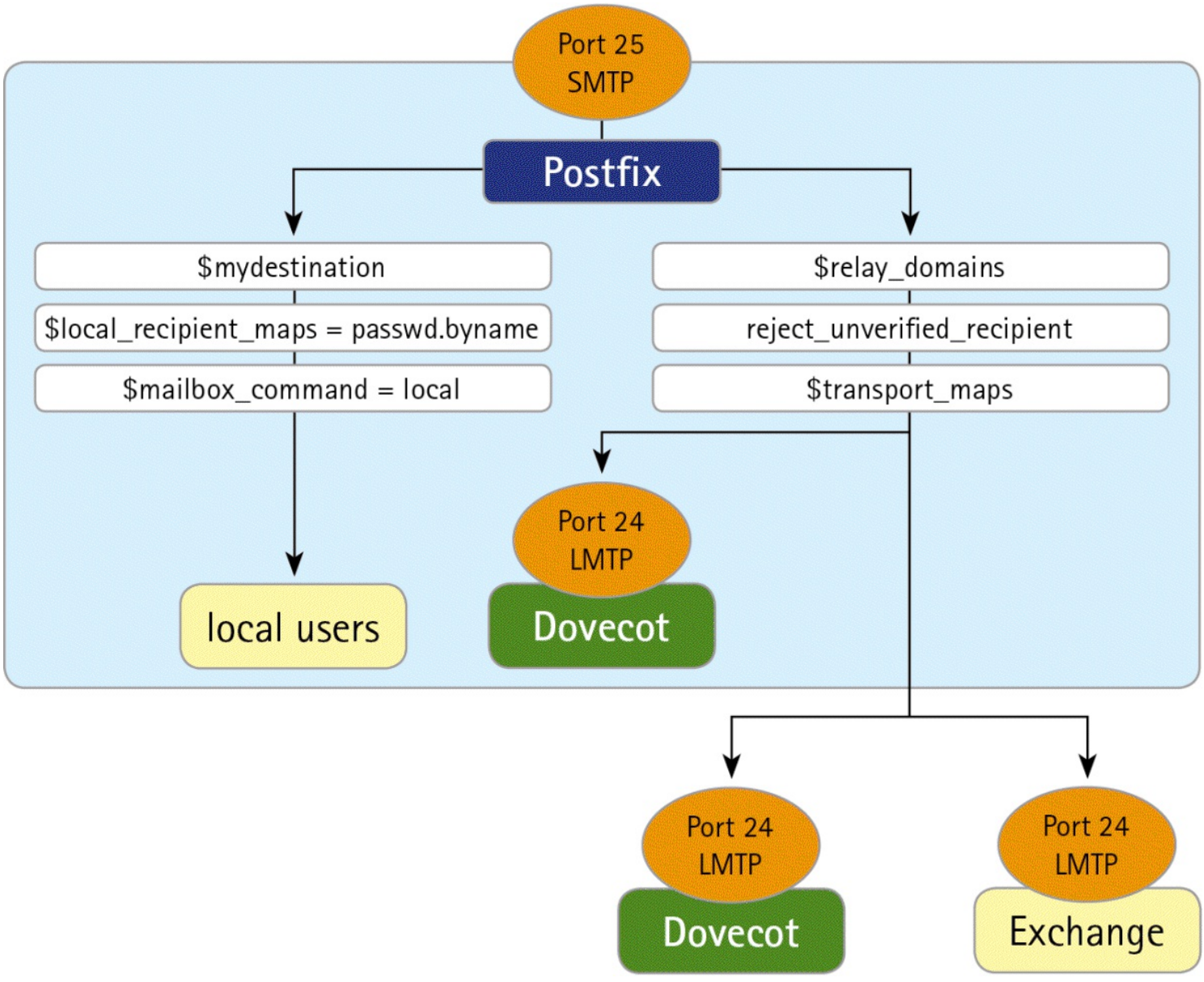


In addition, the distinction whether domains are in `$mydestination`, in `$virtual_alias_domains` or (as shown below) in `$relay_domains` is not unimportant for Postfix. The choices made here in detail will make Postfix behave in rather different ways. You should therefore always use the right "domain class", even if the differences seem unimportant on the face of it.

To put it bluntly: Postfix as a relay in front of Dovecot is not the objective of email routing, so don't brutalize Postfix and don't try to trick it.

Postfix is simply a simple relay in front of a black box called Dovecot. Postfix routes emails to Dovecot via an LMTP delivery or by calling the `dovecot-lda` process. The fact that Dovecot is accidentally running on `localhost` is not relevant here. Where, from the perspective of routing, is the difference between Postfix delivering emails via LMTP to an external exchange server, via LMTP to an external Dovecot server or via LMTP to a locally running `lmtpd` belonging to Dovecot (Figure 6.2)?

Figure 6.2. Whether Dovecot runs on `localhost` or elsewhere, the structure is consistent – the same thing is done in the same way.



Set up the domains you require in Postfix as what they actually are: relay domains. Postfix takes that term to mean domains that it accepts authoritatively from anywhere, but that it then forwards to the final destination – usually using a `transport` map with routing destinations.

Before we move on to the configuration, I would like to explain a preliminary consideration that will save us work and prevent sources of error.

A Postfix table in the hash or btree format always has two columns, because Postfix sends a "question" to this table (*query*) and always requires a response (*result*/*action*). The question "Am I responsible for domain `example.com`?" expects a yes/no response. That means Postfix just needs an answer (=yes), the content of the response is unimportant.

A relay domains table would simply need to list the domains that are later handed on to Dovecot. The second column in this table must exist, but the content is unimportant:

```
example.com     anything
example.org     itdoesntmatter
```

However, Postfix also requires a `transport` map table so it knows where and how to forward the emails for these domains. After all, the DNS-MX records usually refer to this Postfix as an email relay, and the email must then be navigated to its destination by means of manual entries. In this example, a `transport` map would look like this:

```
example.com     lmtp:[127.0.0.1]
example.org     lmtp:[127.0.0.1]
```

It is clear that `$relay_domains` and `$transport_maps` could be combined in a single file in this case. As the `relay_domains` query does not care about the result, it would not do any harm if the response returned to Postfix was `lmtp:[127.0.0.1]`. Postfix is not interested in the content of the response (yet) anyway (at this stage).

So combine entries `transport_maps` and `relay_domains` in `main.cf` as follows:

```
relay_domains = hash:/etc/postfix/relay_domains
transport_maps = hash:/etc/postfix/transport, $relay_domains
```

Warning

As the actual `transport` map may also contain routing entries for domains belonging to other providers, you must under no circumstances integrate the `/etc/postfix/transport` file in `$relay_domains` – in this case, Postfix would also accept emails addressed to the external domains listed there and thereby become an open relay. Instead, the `transport_map` entries in the "own" domains to be relayed are just part of all domains with manual routing entries. For that reason, you should always follow the instructions here. `$transport_maps` may also refer to `$relay_domains` and include the domains listed there in its evaluations – but the reverse is never allowed.

In `/etc/postfix/relay_domains`, list all the domains in your mail cluster as I have just demonstrated:

```
example.com     lmtp:[127.0.0.1]
example.org     lmtp:[127.0.0.1]
example.net     lmtp:[127.0.0.1]
```

If you have decided to perform your LMTP delivery via the Unix socket and not via TCP/IP (Section 6.2), the example would look like this:

```
example.com     lmtp:unix:private/lmtp-dovecot
example.org     lmtp:unix:private/lmtp-dovecot
example.net     lmtp:unix:private/lmtp-dovecot
```

`private/lmtp-dovecot` in this case is the relative file path to `/var/spool/postfix`, so `/var/spool/postfix/private/lmtp-dovecot` (see Section 6.2).

Test your setup. Check that there is an `/etc/postfix/transport` file (whose contents can be empty) as well as an `/etc/postfix/relay_domains` file, and use `postmap` to create the corresponding db file for both tables. After Postfix has been reloaded, Postfix should also accept emails to these domains from external IP addresses (which are therefore not found in `$mynetworks`) and forward these emails properly to Dovecot via LMTP:

```
2013-04-27T18:41:23.651482+02:00 flash postfix/lmtp[16288]: 8ADEA23F2: to=
```

Dovecot itself also logs the receipt of the email by means of its `lmtpd`:

```
2013-04-27T18:41:23.651200+02:00 flash dovecot: lmtp(16295, klaus@example.
```

The best thing is that the local email system remains untouched and continues to work. `root@localhost` remains `root@localhost`.

6.4. What recipients are there? Dynamic recipient verification

Until now, we have not put Postfix in the position where it can reject emails addressed to non-existent recipients. This task is supposed to be performed by the `$relay_recipient_maps` parameter, which is empty at first by default so Postfix accepts emails for all recipients in its domains in `$relay_domains`.

If your users are located in LDAP or SQL, you could consider placing appropriate LDAP or SQL queries in `$relay_recipient_maps`. But if you manage your users in another way, e.g. using an ASCII file in `/etc/dovecot/users`, Postfix cannot query these data sources automatically; as a result, cron scripts are often used that convert something and prepare it as a recipient list for Postfix.

You should also consider that real-time LDAP and SQL queries require corresponding access data on the mail relay and that your Postfix relay requires TCP/IP access to LDAP, SQL or your ActiveDirectory server, something that is often not desirable if there is a clear separation between DMZ and intranet.

But there is a much easier and more convenient way. Use the `dynamic address verification` provided by Postfix. Postfix can learn (and cache) automatically which recipients exist on a downstream system. To do so, Postfix does not need SQL or LDAP; instead, it just needs access via SMTP or LMTP (which exists anyway).

Postfix generates *verify* emails, which means it connects to the backend via SMTP/LMTP and checks whether it accepts emails addressed to the recipient in question. Postfix starts delivering a test email and checks whether the `RCPT TO:` field in the address in question receives a positive or negative result.

However, it never actually sends the email: after the `RCPT TO:` command, Postfix transmits `QUIT` and cancels the connection prematurely. Not much has happened, apart from an entry in the log file, and the recipient never notices that he has been “tested”.

Postfix caches these results in a local Berkeley database, so repeated emails to one address do not result in too many queries.

```
address_verify_map = btree:$data_directory/verify_cache
```

By the way, Postfix cleans up its `verify` database automatically. You do not need to do anything. Non-existent addresses expire after 3 days, but even after 3 hours, Postfix would check again whether the user now exists:

```
address_verify_negative_expire_time = 3d
```

```
address_verify_negative_refresh_time = 3h
```

Postfix retains positive addresses in its cache for 31 days, but it would begin verification again after 7 days and therefore realize early on if a user no longer exists:

```
address_verify_positive_expire_time = 31d
address_verify_positive_refresh_time = 7d
```

That means it does not take long to set up dynamic recipient verification.

Check whether `address_verify_map` is set – by default, old versions of Postfix contain no entry here:

```
flash:~ # postconf address_verify_map
address_verify_map = btree:$data_directory/verify_cache
```

You can now use restriction `reject_unverified_recipient` in your `smtpd_recipient_restrictions`. Postfix will then take a look at its `verify` database at this point in order to determine whether the recipient exists or not. If there is no cache entry for this recipient, Postfix quickly launches a `verify` process. That process usually takes a very short time, so it is possible to inform the delivering client immediately whether the email to this recipient has been accepted.[30]

There is a question that is worth considering: *where* in your `smtpd_recipient_restrictions` is the `reject_unverified_recipient` verification applied? It may only occur *after* `$mynetworks`, `$permit_sasl_authenticated` or `$permit_tls_clientcert`, as there may be no verification with other providers for outgoing emails from your users.[31]

Don’t trick yourself.

As `reject_unverified_recipient` is only executed after `permit_mynetworks`, you are unable to test your setup from a client IP from `$mynetworks`. Remember that and don’t be surprised that it “doesn’t work”.

Here is another piece of advice: I like to set the error code returned for non-existent users during the dynamic verification to `577`. It doesn’t actually matter which 500 code you return. But this unusual number makes it easy to track the results of dynamic recipient verification in the log file, and as an administrator it is easier to see in stressful situations what has caused the different rejections.

```
flash:~ # postconf -e "unverified_recipient_reject_code = 577"
```

6.5. Setting up SMTP-Auth in Postfix

We have thought a lot about the best way for emails to reach Dovecot via Postfix and how these emails can be made available to users via POP3/IMAP. But a full mail server requires another elementary function: the provision of an SMTP relay that end users or other servers can use to deliver authenticated emails to be sent to external destinations.

Postfix of course supports SMTP-Auth for this purpose, i.e. a password authentication option in the SMTP protocol. However, Postfix does not consider itself responsible for the actual verification of the password. Postfix itself has no own database with user names and passwords (at least when receiving emails); instead, it uses the SASL framework[32] to attach itself to other services that then perform password verification.

In the mail server area, that used to be the traditional Cyrus-SASL, a notorious construct made up of an `saslauthd` running with `root` permissions that supported only specific verification methods, or an SASL library named `auxprop` that supported the other methods – or that sometimes had the same abilities as `saslauthd` but used completely different methods and configurations to do so. Those who could think of no other solution finally integrated PAM modules via the `saslauthd` – and so there were three completely different procedures (with numerous parties involved) for verifying passwords on an LDAP server. In short: it was absolutely no fun.[33]

But this horrible situation is over: Dovecot offers its own SASL interface to Postfix, amongst others, and the interface works easily, quickly and reliably. It makes you wonder why you had to suffer for 15 years. Postfix itself has supported Dovecot-SASL since version 2.3.

The procedure is simple:

1. Dovecot by nature contains a full authentication mechanism so it can provide POP3 and IMAP.
2. Dovecot provides an interface for Postfix via a Unix socket.
3. Postfix simply passes authentication queries to Dovecot via that interface and receives a yes/no response.

For the user, authentication via SMTP is no different from the POP3/IMAP areas. That is a good thing, because a normal user does not usually choose between SMTP and POP3/IMAP and is usually unaware that there are huge technical differences between “sending an email” and “fetching an email”.

It does not take long to set up the `auth` socket in Dovecot; you will find the corresponding configurations in the `10-master.conf` file. There, an `auth-userdb` Unix socket is already

prepared in the service, and Dovecot uses that socket for its own purposes and those of its modules:

```
service auth {
  unix_listener auth-userdb {
    #mode = 0666
    #user =
    #group =
  }
[...]
}
```

Under that, an additional `auth` socket is usually prepared for Postfix and equipped with comment characters. Activate the socket there and use the path shown here to ensure that the socket lies securely in the chroot environment of Postfix:

```
service auth {
  unix_listener auth-userdb {
    #mode = 0666
    #user =
    #group =
  }
  #
  # Postfix smtp-auth
  unix_listener /var/spool/postfix/private/auth {
    mode = 0666
  }
}
```

By the way: if you run your SMTP-Auth server separately from the IMAP server, you should install Dovecot there as well and copy the entire Dovecot configuration in `/etc/dovecot` to it. However, you need to set the value `protocols = none` in `dovecot.conf` or, in the case of Debian, install only the corresponding core packages. Dovecot will then start in its new kernel and open the auth socket, but it will not offer any other services.

As soon as you restart Dovecot, the corresponding socket must appear in the file system:

```
flash:~ # ls -la /var/spool/postfix/private/auth
srw-rw-rw- 1 root root 0 Jun  1 22:22 /var/spool/postfix/private/auth
```

Now you just need to instruct Postfix that it has to connect to Dovecot, and tell it how to do so. Enter the following in `main.cf`:

```
smtpd_sasl_auth_enable = yes
smtpd_sasl_type = dovecot
smtpd_sasl_path = private/auth
broken_sasl_auth_clients = yes
```

By the way: the Postfix parameter we just used, `broken_sasl_auth_clients`, is the reason why the auth line is output twice – once with a space (conforms to RFC) and once with an equals sign (conforms to Outlook). Unfortunately, there was an implementation error in

Outlook many years ago, so Outlook could no longer authenticate with RFC-conforming SMTP servers while still being able to do so with Microsoft's own products. This problem with ancient Outlook clients is usually avoided by putting out the corresponding auth line once in each version.

After Postfix has been restarted, it should offer SMTP authentication:

```
flash:~ # telnet localhost 25
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
220 flash.heinlein-support.de ESMTP Postfix
EHLO kk
250-flash.heinlein-support.de
250-PIPELINING
250-SIZE
250-VRFY
250-ETRN
250-AUTH PLAIN
250-AUTH=PLAIN
250-ENHANCEDSTATUSCODES
250-8BITMIME
250 DSN
quit
221 2.0.0 Bye
Connection closed by foreign host.
```

Postfix automatically takes over the authentication procedures that Dovecot supports and that Dovecot itself uses for POP3 and IMAP. Depending on how you save your passwords and on your settings in the `auth_mech` parameter in the `10-auth.conf` file in Dovecot, Postfix can also offer other procedures, which would look like this:

```
250-AUTH PLAIN LOGIN DIGEST-MD5 CRAM-MD5
250-AUTH=PLAIN LOGIN DIGEST-MD5 CRAM-MD5
```

For more information, read the explanation in Section 5.10.5.

One important decision you still need to make is whether the `PLAIN` and `LOGIN` plain text methods can be offered at any time (which a "man in the middle" could tap) or whether `PLAIN` and `LOGIN` are permitted only for SSL/TLS-encrypted connections. The `noplaintext` option allows you to regulate this separately from Postfix for plain text and SSL/TLS connections by making the following entries in `main.cf`:

```
smtpd_sasl_security_options = noanonymous, noplaintext
smtpd_sasl_tls_security_options = noanonymous
```

You could also consider requiring SSL/TLS-secured connections for every form of authentication. A simple entry ensures that Postfix allows PLAIN/LOGIN and CRAM processes only on secure connections:

```
smtpd_tls_auth_only = yes
```

You may also want to take a look at the following two parameters from the area of SASL and authentication, which you can enter in `main.cf` in Postfix:

`smtpd_sasl_authenticated_header = no`
: If you set this value to `yes`, Postfix will log the SASL user name in the `Received:` lines in the email header, which is however visible to anyone.

`smtpd_sasl_exceptions_networks = 192.168.0.0/24 10.0.0.0/8`
: Here you can define IP areas where clients are not offered SMTP-Auth. You could thereby protect yourself slightly from certain areas against brute-force attacks.

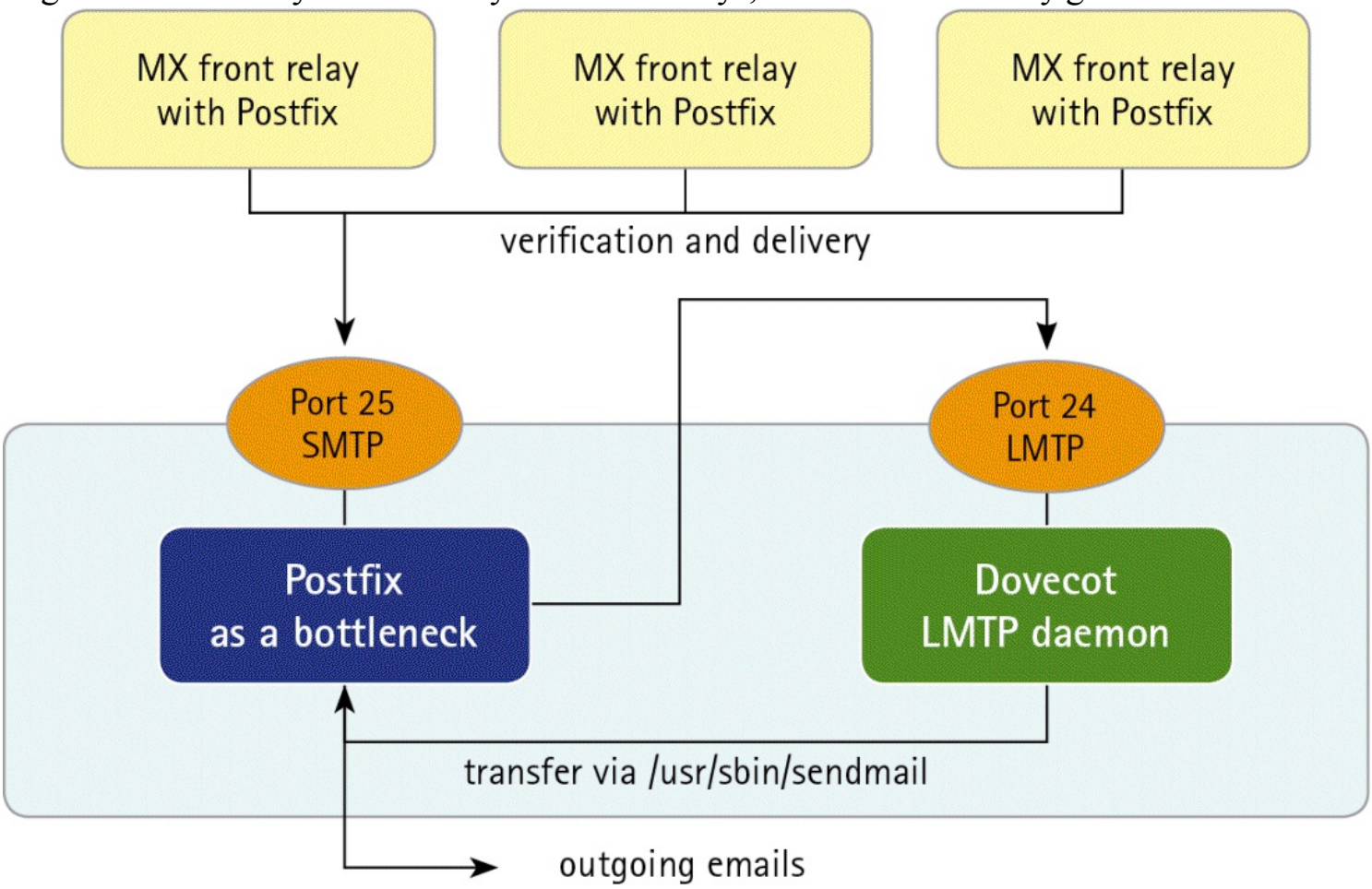
6.6. Less is more: do not go via Postfix

If we summarize all the results of this chapter so far, we can draw two important conclusions:

1. Postfix does not need its own access to the user list and the recipient database.
2. Emails are delivered via LMTP via a TCP-capable network protocol.

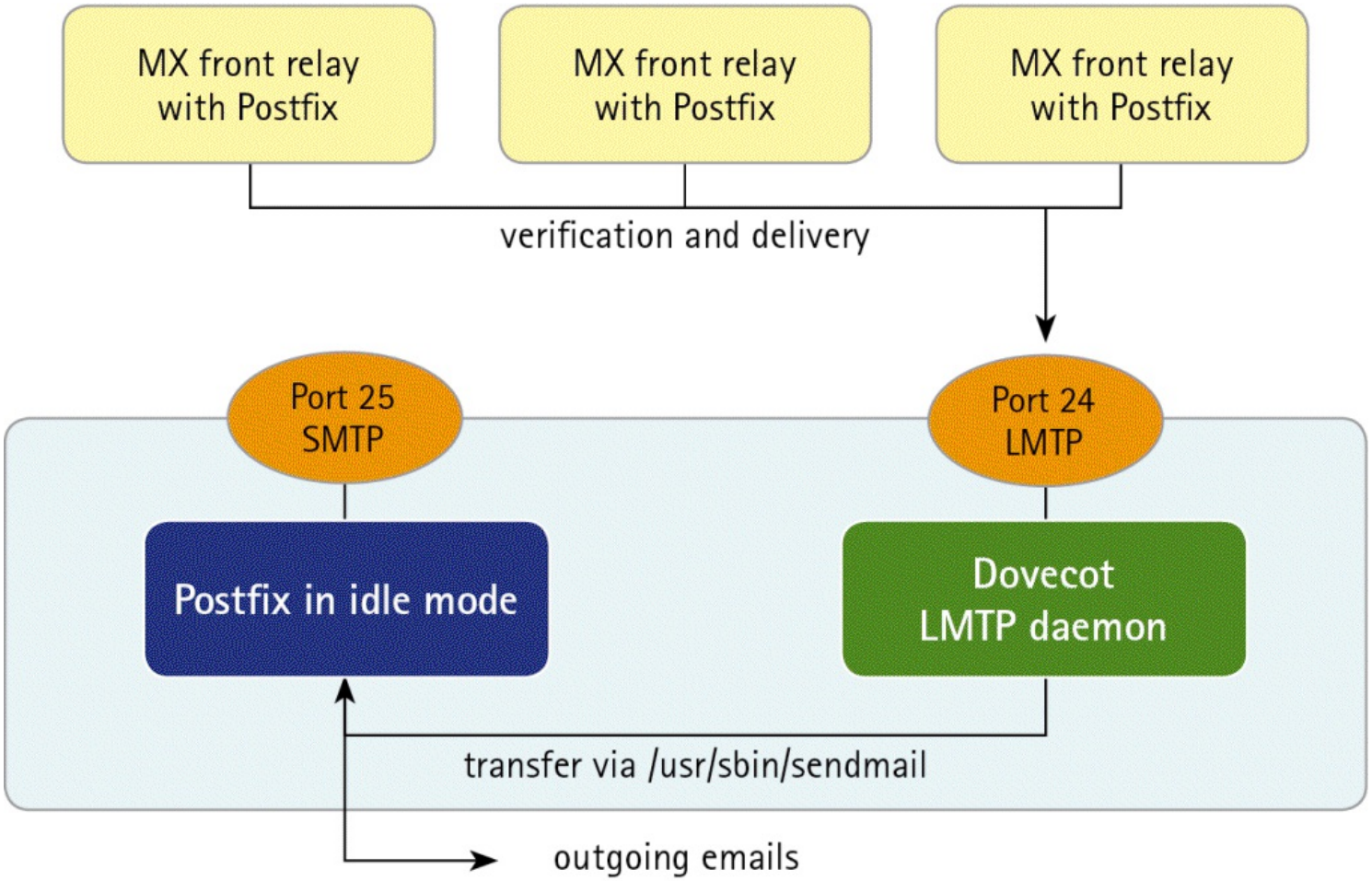
And that means: if the receipt (and spam filtering) of emails takes place separately from the IMAP server on MX front relays in larger setups, it is unnecessary for the MX servers to pass the emails on to the Postfix of the IMAP server first via SMTP, just for Postfix to then deliver the emails to `localhost` via LMTP. In that case, Postfix on the IMAP server is nothing more than a useless percolator, as you can see from Figure 6.4. And it will cost you valuable disk I/O on the IMAP server.

Figure 6.3. Even if you have many MX front relays, the load will finally gather in one Postfix



The MX relays could just as well address the IMAP server directly via LMTP and deliver the emails directly to Dovecot (Figure 6.4). Of course this is only possible if Dovecot makes the LMTP socket available on its external IP addresses as well as on `localhost`.

Figure 6.4. It makes sense to deliver directly via LMTP, particularly if you have many MX servers.



For this purpose, open the LMTP socket as described above – but without restricting it to `localhost`:

```
service lmtp {
  unix_listener lmtp {
    #mode = 0666
  }

  # Create inet listener only if you can't use the above UNIX socket
  inet_listener lmtp {
    # Avoid making LMTP visible for the entire internet
#    address = 127.0.0.1
    port = 24
  }
}
```

Consequently you can now adapt the routing tables on the MX relays. If port 25 (SMTP) was specified in the past in order to deliver the email to the Postfix of the IMAP server

```
example.com     smtp:[imap.example.com]
example.org     smtp:[imap.example.com]
```

you should now choose the direct path to port 24 (LMTP) directly to Dovecot; see the transport method `lmtp:` in this example:

```
example.com     lmtp:[imap.example.com]
example.org     lmtp:[imap.example.com]
```

You will continue to take advantage of the decentralized load distribution by means of multiple MX servers. Dovecot remains a central “bottleneck”, but you no longer need to transport incoming emails through your file system twice, which already takes a considerable load off your IMAP system. The conversion of email addresses by means of `virtual_alias_maps` and similar must of course be performed directly on the MX relays in such setups, but that is a desirable configuration anyway.

After all, you may also have a variety of email backends, e. g. one Dovecot IMAP for students at a university and a traditional groupware server for the administrative department. Thanks to our clean definition of relay domains in Postfix, you can now conveniently accept the different domains from your email relay and route them to different destinations:

```
students.example.com    lmtp:[imap.studserv.example.com]
office.example.com      lmtp:[groupware.example.com]
example.com             lmtp:[groupware.example.com]
```

Just to prevent misunderstandings: you will not be able to do entirely without a running Postfix installation on the IMAP server, as Sieve filters and similar can generate new emails locally that are transferred to Postfix by Dovecot, typically by means of `/usr/sbin/sendmail`. You will have to keep your Postfix simply to send outgoing emails.

6.7. Advanced information on the configuration of Postfix

As far as possible within the scope of this book, you have been supplied with (almost) all the information you need for a Postfix SMTP relay in conjunction with Dovecot. I use the term "almost" because the chapter on quotas (Section 11.12) still provides information on how to set up a matching quota policy daemon.

In addition, there are plenty of topics to write about that relate to Postfix mail relays: SMTP restrictions, MX email reception, spam/virus protection, HA strategies and many more. That would be too much information for a book about Dovecot, which is why you should consult a book about Postfix. I have written a book on the topic that provides all the details about Postfix and is an ideal companion for this book, but unfortunately it has not been translated into English, so you will need to consult a different one unless you read German.

---

[26] If you make intensive use of `procmail`, you should check whether it would be worth it to translate the rules into Sieve scripts (Section 12.3).

[27] If (unlike in this example) you open the socket no just on 127.0.0.1 but also on your server's "real" IPs, the socket could be addressed directly by your Postfix relays and you could shorten the path of the email accordingly while also relieving the server (Section 6.6).

[28] And if that is the case, why have you continued reading this book up to this point?

[29] This generates backscatter mail servers that receive all emails and then bounce them later on. Terrible!

[30] If the target system is unavailable, or if verification is temporarily not possible for other reasons, the delivering client simply receives a temporary error and will try again later on. The same happens if the SQL or LDAP server is unavailable. In fact, there are only advantages: even if your database is not available, Postfix can use its cached verify data to go on accepting emails addressed to existing users and buffer them.

[31] Incorrect recipient addresses could cause other providers to view you as a spam attacker and block you temporarily; in addition, greylisting or other spam checks in the target system would prevent your query.

[32] SASL stands for *Simple Authentication and Security Layer* – though many claim that Cyrus-SASL is neither "simple" nor "secure", as we will see in a moment.

[33] In the Postfix book, the special properties of Cyrus-SASL cover 16 pages, while the special properties of Dovecot-SASL cover exactly *one*.

Chapter 7. mbox, Maildir and mdbox: a comparison of storage formats

Over the years, two formats in particular have become established for the storage of emails: *mbox* and *Maildir*. Both have advantages and disadvantages that will be examined in this chapter. Both formats are RFC standard or long-established standards on the basis of well-known RFC formats and can therefore be read and written by numerous email programs.[34] In addition, Dovecot has developed its own proprietary format, *mdbox*, which has considerable advantages for use on large, powerful mail servers because it combines the advantages of mbox and Maildir. This chapter will also examine mdbox in detail.

7.1. Overview of the three formats

Let's begin by comparing the three formats:

mbox

In the mbox format, a user has an email file for every IMAP folder, where all the emails in his INBOX are saved one after the other. Usually mbox is activated by default in every Linux installation; you will usually find the mbox files for the `INBOX` under `/var/spool/mail` or `/var/mail`. The mbox format itself is actually designed for use with POP3, where emails are fetched regularly and are also deleted from the server, so the mbox files do not grow too large.

Dovecot can also handle IMAP using mbox files, but that is rather problematic: as mbox files increase in size, it becomes ever more difficult and I/O-consuming to work with them; when you delete an email from the middle of an mbox file, the entire mbox file has to be copied. In addition, only *one* process can write in an mbox file at any one time, so you need write locks.

I have seen systems where fewer than 200 users have forced a 15K hard drive to its knees because the admin set up IMAP via mbox files. The system was therefore constantly occupied with copying GByte-sized mbox files. In some cases, Postfix was unable to write a new message to the mbox file for 3-4 hours, because the file was constantly occupied and locked by other processes. I therefore urgently advise you not to use mbox files for mail servers with IMAP access.

We will not discuss mbox and its structure further in this book, because it really makes no sense to offer new systems with this format.

Maildir

Maildir is the most popular storage format for emails. It is simple, robust, easy for admins to handle and also adequate for larger IMAP servers. Every email is placed in a separate file, and a user's entire email directory then consists conveniently of multiple file folders (= IMAP folders) that contain the respective email files. Unlike in mbox, it is therefore easy to delete individual emails, and the different processes do not interfere with one another when working in parallel. Write locks are not required in Maildir, and it is easy to save incrementally, because the individual email files do not change later on.

However, Maildir also has some disadvantages – multiple small files are a strain on any file-based backup. It is far more laborious to check and back up an INBOX with 80,000 email files than an INBOX with an mbox file that comprises 80,000 emails. However,

most IMAP servers at technical colleges and smaller universities are not usually big enough for this situation to cause problems, so Maildir is often the best choice due to its many advantages. However, mail servers with many TBytes of user data will reach their limits if there are billions of small email files.

We will examine Maildir in Section 7.2 in more detail.

mdbox

mdbox is Dovecot's own "professional format" for storing emails. There is neither an mbox file with 80,000 emails nor a Maildir folder with 80,000 small files. Instead, Dovecot creates numerous individual `m` files that are a few MB in size each and contain several emails. In practice, an mdbox memory with 80,000 emails would contain around 800 `m` files, each containing around 100 emails. As Dovecot "closes" an `m` file once it reaches a specific size, the quantity and distribution can vary, depending on the distribution of emails, and also on the maximum size the administrator defines for an `m` file. A size between 5 and 10 MB has proven workable.

Backups therefore have little difficulty with mdbox, because the number of files is sufficiently small; incremental backup works smoothly. Delete operations in the comparatively small `m` files do not result in never-ending I/O copy operations (and are additionally minimized by Dovecot), and parallel working in an mdbox store is no problem, because it is possible to work in multiple `m` files in parallel as the different write processes do not block one another.

However, unlike in mbox or Maildir files, the administrator can no longer copy, delete or edit files at the file system level at will. mdbox is based on a database structure, and any work in the mdbox store must be performed using `doveadm` commands.

We will discuss mdbox in more detail in Section 7.3.

7.2. Maildir as an email storage format

A user's Maildir storage consists of directories containing the email files and of several ASCII files for administration, e.g. for quota information. There are no binary files at first, the administrator can read and edit anything.

Dovecot however adds some of its own binary files to Maildir for better caching. These files are optional and designed only for Dovecot; other processes can continue to work in these directories and with these data in line with the Maildir standard.

Maildir is designed so that numerous programs can work in a Maildir storage in parallel without having to consider one another. Locally running email programs such as pine or KMail read the emails directly from the Maildir without detouring via IMAP. MTAs such as Postfix or Exim can save emails directly in a user's Maildir directory. Maildir by definition requires no write locks and is designed to work smoothly even in NFS environments.[35]

If the mail server is one where all users have shell access (i.e. an account in `/etc/passwd` and a home directory), every user can have his emails delivered to a `Maildir` directory in his home directory (e.g. `/home/tux/Maildir`). Messages are then saved with the permissions of the user.

On a mass mail server, which ideally should know no separate users apart from the administrators, the user database will usually be stored in MySQL, LDAP or similar. In such cases, email users do not have home directories on the server. Instead, a separate directory structure is created (e.g. `/srv/vmail`) where every user is assigned his own email directory (e.g. `/srv/vmail/tux/Maildir`).

7.2.1. Technical structure of Maildir

The Maildir directory of a mailbox contains at least three subfolders

```
flash:~ # ls -la /srv/vmail/example.com/klaus/Maildir
drwx------  7 h users 4096 Jul 27 12:07 .
drwxr-xr-x  8 h users 4096 Jul 27 12:04 ..
drwx------  2 h users 4096 Jul 27 12:04 cur
drwx------  2 h users 4096 Jul 27 12:04 new
drwx------  2 h users 4096 Jul 27 12:04 tmp
```

`cur`
: The `cur` directory contains all the messages that the client already knows.

`new`
: `new` receives all the messages added since the last login so that the server can flag them as `\Recent`. Once a user has logged in and selected the IMAP folder in question, these new emails are immediately moved to `cur`; the `\Recent` flag is only noted in the memory for the duration of this IMAP connect.

`tmp`
: All new files are first created and filled with content in the `tmp` directory, as it can take several (milli)seconds to save a file completely. While the email file is being written, it is still incomplete and may not be viewed by other processes or even delivered to the user (in its incomplete state). Usually the write locks mentioned earlier would be required to lock these files. Instead, Maildir writes everything to `tmp` at first and then moves the complete email file to `cur` or `new` as soon as the file is released.

To summarize: `cur` and `new` form an IMAP folder together, while `tmp` never contains relevant data.

If IMAP users sort their emails to different subfolders that – unlike in the case of POP3 – are stored on the server, the Maildir will contain the IMAP folders in the form of additional directories as well as `cur`, `new` and `tmp` (for the INBOX).

The names of these directories are made up of a point and the actual folder name. If the `INBOX`, `friends` and `company` folders exist, the Maildir listing will look like this:

```
flash:~ # doveadm mailbox list -u klaus@example.com
INBOX
INBOX.friends
INBOX.friends.vacation
INBOX.friends.orchestra
INBOX.company
INBOX.nicethings
flash:~ # ls -lad /srv/vmail/example.com/klaus/Maildir/.INBOX.*
drwx------  5 vmail vmail  4096 Jun  2 23:35 .INBOX.surprise
drwx------  5 vmail vmail  4096 Jun  1 22:26 .INBOX.friends
drwx------  5 vmail vmail  4096 Jun  1 22:26 .INBOX.friends.orchestra
drwx------  5 vmail vmail  4096 Jun  1 22:26 .INBOX.friends.vacation
drwx------  5 vmail vmail  4096 Jun  2 23:34 .INBOX.nicethings
```

The IMAP subfolders in turn are independent directories in the Maildir format and therefore once again contain the `cur`, `new` and `tmp` folders so that emails can be saved in them:

```
flash:~ # ls -la /srv/vmail/example.com/klaus/Maildir/.INBOX.friends
drwx------  5 vmail vmail  4096 Jun  1 22:26  .
drwx------  7 vmail vmail  4096 Jun  1 22:26  ..
drwx------  2 vmail vmail  4096 Jun  1 22:26 cur
-rw-------  1 vmail vmail  4096 Jun  1 22:26 maildirfolder
drwx------  2 vmail vmail  4096 Jun  1 22:26 new
drwx------  2 vmail vmail  4096 Jun  1 22:26 tmp
```

File `maildirfolder` is always empty and indicates that this is already a subfolder that cannot contain any additional Maildirs.

If an IMAP folder called `friends` contains another IMAP subfolder, they are not saved under one another in the file system (i.e. as `Maildir/.friends/.vacation`), but instead all created on the uppermost level of the Maildir, i.e. as `Maildir/.friends.vacation`:[36]

```
/srv/vmail/example.com/klaus/Maildir/
                                  +-- cur
                                  +-- new
                                  +-- tmp
                                  +-- .INBOX.friends
                                         +-- cur
                                         +-- new
                                         +-- tmp
                                         +-- maildirfolder
                                  +-- .INBOX.friends.orchestra
                                         +-- cur
                                         +-- new
                                         +-- tmp
                                         +-- maildirfolder
                                  +-- .INBOX.friends.vacation
                                         +-- cur
                                         +-- new
                                         +-- tmp
                                         +-- maildirfolder
```

The folder structure is always mapped in the file name using the point as a hierarchy separator, even if you have set up `/` as the hierarchy delimiter in the IMAP namespace (see Section 9.2).

If an IMAP client has subscribed to individual IMAP folders at some stage, there is a `subscriptions` file with the subscription list of the account saved line by line:

```
flash:~ # ls -la /srv/vmail/example.com/klaus/Maildir
drwx------  9 vmail vmail  4096 Jun  1 22:26 .
drwxr-xr-x  8 vmail vmail  4096 Jun  1 22:26 ..
drwx------  5 vmail vmail  4096 Jun  2 23:35 .INBOX.surprise
drwx------  5 vmail vmail  4096 Jun  1 22:26 .INBOX.friends
drwx------  5 vmail vmail  4096 Jun  1 22:26 .INBOX.friends.orchestra
drwx------  5 vmail vmail  4096 Jun  1 22:26 .INBOX.friends.vacation
```

```
drwx------  5 vmail vmail  4096 Jun  2 23:34 .INBOX.nicethings
drwx------  2 vmail vmail  4096 Jun  1 22:26 cur
drwx------  2 vmail vmail  4096 Jun  1 22:26 new
drwx------  2 vmail vmail  4096 Jun  1 22:26 tmp
-rw-r--r--  1 vmail vmail  4096 Jun  1 22:26 subscriptions
flash:~ # cat /srv/vmail/example.com/klaus/Maildir/subscriptions
INBOX
INBOX.company
INBOX.friends.orchestra
```

7.2.2. File names of emails

As mentioned before, Maildir does not use databases with information on the stored emails. That makes the server robust and reliable. No index can be shot, no database can be inconsistent or faulty.

On the other hand, it is often not particularly efficient to collect all the information from a variety of different files at every access. Specifically in IMAP emails may have meta information that cannot be saved in the actual email text – e. g. the `\Seen` and `\Flagged` IMAP flags.

For that reason, a few tricks are applied to make the whole thing quicker: many pieces of information are coded in the file name of every individual email. One single directory listing can thereby collect a lot of information very quickly without having to examine every file in detail (which would take a lot of time and therefore be “expensive”).

The best way to illustrate this is to send yourself a test email to the user’s Maildir directory.

There it first appears in the `new` folder of the `tux` Maildir:

```
flash:~ # ls /srv/vmail/example.com/klaus/Maildir/new
1367080883.M622430P16295.flash.heinlein-support.de,S=547,W=559:2,S
```

The file name of the hello-world email consists of a random but unique ID (that often includes the date, time, Inode number and so on) and the host name of the storing server (in this case: `flash.heinlein-support.de`) in order to prevent accidental name conflicts on network drives.

Now log on to the IMAP server in parallel to access this email:

```
flash:~ # telnet localhost 143
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
* OK [CAPABILITY IMAP4rev1 LITERAL+ SASL-IR LOGIN-REFERRALS ID ENABLE IDLE
a login klaus@example.com secret
a OK [CAPABILITY IMAP4rev1 LITERAL+ SASL-IR LOGIN-REFERRALS ID ENABLE IDLE
```

Use the following IMAP command to access the `Inbox` IMAP folder of user `tux`:

```
a select INBOX
* FLAGS (\Answered \Flagged \Deleted \Seen \Draft company)
* OK [PERMANENTFLAGS (\Answered \Flagged \Deleted \Seen \Draft company \*)
* 0 EXISTS
* 1 RECENT
* OK [UNSEEN 1] First unseen.
* OK [UIDVALIDITY 1367080883] UIDs valid
* OK [UIDNEXT 2] Predicted next UID
* OK [NOMODSEQ] No permanent modsequences
a OK [READ-WRITE] Select completed.
```

With `1 RECENT`, Dovecot shows that this folder contains an email that has been received since login.

You can view it as follows:

```
a FETCH 1 RFC822
* 1 FETCH (RFC822 {559}
Return-Path: <root@flash.heinlein-support.de>
Delivered-To: <klaus@example.com>
Received: from flash.heinlein-support.de ([127.0.0.1])
        by flash.heinlein-support.de (Dovecot) with LMTP id g+XPJLP/e1GnPw
        for <klaus@example.com>; Sat, 27 Apr 2013 18:41:23 +0200
Received: by flash.heinlein-support.de (Postfix, from userid 0)
        id 8ADEA23F2; Sat, 27 Apr 2013 18:41:10 +0200 (CEST)
Message-Id: <20130427164123.8ADEA23F2@flash.heinlein-support.de>
Date: Sat, 27 Apr 2013 18:41:10 +0200 (CEST)
From: root@flash.heinlein-support.de (root)

TEST
)
a OK Fetch completed.
```

The `new` folder in the file system is now empty, because the email was moved to the `cur` folder upon first access:

```
flash:~ # ls /srv/vmail/example.com/klaus/Maildir/new

flash:~ # ls /srv/vmail/example.com/klaus/Maildir/cur
1367080883.M622430P16295.flash.heinlein-support.de,S=547,W=559:2,S
```

The file name has also changed: it now has the flags `:2,S` at its end. `S` stands for the IMAP flag `\Seen`, which was set automatically due to the read access.

You can set additional IMAP flags during an IMAP session:

```
a4 STORE 1 +FLAGS (\Flagged \Answered)
* 1 FETCH (FLAGS (\Flagged \Answered \Seen \Recent))
a4 OK STORE completed.
```

`\Answered` marks emails that have already been answered, while `\Flagged` is used to flag important messages.

In the Maildir, the file names are modified accordingly: `\Answered` becomes `R` for *reply* `\Flagged` is mapped as `F`:

```
flash:~ # ls /srv/vmail/example.com/klaus/Maildir/cur
1122460858.V301Ic964.linux:2,FRS
```

Flags *must* be saved in alphabetic order: `2,FRS` is correct, `2,SFR` is not permitted. The following list shows which Maildir code stands for which IMAP flag.

`F` (`\Flagged`)

The message is flagged (this is usually how clients remember *important* messages)

`D` (`\Draft`)
: Email is flagged as a draft

`P` (not implemented in the IMAP protocol)
: Email has been forwarded

`R` (`\Answered`)
: Email has been answered

`S` (`\Seen`)
: Email has been read

`T` (`\Deleted`)
: Email is flagged for deletion ("trashed")

By the way, the file name does not have a precise definition; the respective documents usually use the term "should".[37] Often it will correspond to the following pattern:

```
<time>.<microseconds>P<pid>V<dev>I<inode>.<host>,S=<bytes>:2,<flags>
```

Up to the optional specification of the file size (`S=<bytes>`) or the mandatory colon and therefore the information on an email's flag, the precise file name is not important – it must simply be ensured that the same file name cannot be created twice. For this reason, a variety of elements are combined in order to produce a unique name even in cluster operation, e. g. time, host name, process ID, hard drive and Inode number:

- The time in seconds since 1.1.1970 (in the case of the file mentioned above, `1122460858.V301Ic964.linux:2,FRS` `1122460858` seconds).
- Instead of the `<microseconds>` placeholder, the time can optionally be made more precise by specifying microseconds. Postfix does not do so.
- The process ID (`<pid>`) of the saving process. This information also avoids name conflicts. Postfix leaves out this information along with the leading `F` – as you can see from the example file.
- The `<dev>` placeholder is replaced with the device number of the device containing the email file (in the example file: `301`).
- The `I` is followed by the hexadecimal number of the Inode with which the email file begins, in this case `c964`. It is still possible to copy these files at will and thereby save them in other Inodes, as this information is only one of many tricks to ensure that the file name is unique.
- The host name that has stored the email file.
- Many programs leave out the information on the file size in the form `S=<bytes>` (`S` stands for *size*). Dovecot will add this parameter to the file name if it has the opportunity.

The current quota load is logged in the separate `maildirsize` file anyway, but saving the length of the email in bytes helps the server to calculate the consumed storage volume quickly.

As a result, even large directories containing thousands of emails will not result in performance problems. For details on quotas, see Chapter 11.

- Separated by a colon, the file name specifies if the subsequent flags are defined in an RFC (`:2,<flags>`) or are experimental flags (`:1,<flags>`).[38]

But as I have said, the actual file name of the Maildir email up to the colon is unimportant. You could also create an email file and name it `test`.

If you create a new email file (with matching file permissions) in `cur`, `new` or an IMAP subfolder with RFC-2822-conforming content, it will appear as an email in the user's mailbox. When such an email file is deleted, it disappears from the mailbox. You can also manipulate the content of the file using an ASCII text editor as long as the structure of the file continues to conform to RFC.

It is only quotas (if you are even using any) that become imprecise if manipulations are performed in these Maildir directories. But that is not a big problem, as Dovecot regularly checks these directories and recalculates the `maildirsize` files.

7.2.3. Keywords: custom IMAP flags

Next to the five official IMAP flags described above, IMAP clients have the ability to use other custom flags.[39] They are also referred to as *keywords*; they only differ from system flags because they lack a preceding backslash in their name: `\Seen` is an official IMAP flag, `hey` is a custom one. Apart from this detail, custom and system flags are addressed and set in the same way.

Keywords can therefore be temporary in nature just like normal IMAP flags (in which case they are lost at every folder change) or permanently stored on the server (in which case they are retained even when the user logs out).

Dovecot manages individual keywords in the `dovecot-keywords` file, and keywords are numbered for administration purposes:

```
0 $Junk
1 $NonJunk
2 $Test
3 $Company
```

At the same time, Dovecot saves the keywords of an email in the Maildir file name – not as a number, but as a corresponding lower-case (!) letter, i.e. `0=a`, `1=b`, `2=c`, `3=d` and so on. An email with the `$Company` flag would therefore be given flag `d`:

```
-rw------- 1 vmail vmail  298 Apr 27 18:41 1367080883.M622430P16295.flash.
```

That means Dovecot can manage no more than 26 flags in the Maildir name (`a` to `z`, i.e. `0` to `25`). If more keywords are set, they are written to `dovecot.index.cache`. That means that Dovecot is able to store more keywords, but other programs can only recognize 26 flags from the Maildir files – which should be sufficient in practice.

You can set any flag you like via IMAP without any further preparation:

```
flash:~ # telnet mail.example.com 143
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
* OK [CAPABILITY IMAP4rev1 LITERAL+ SASL-IR LOGIN-REFERRALS ID ENABLE IDLE
a login klaus@example.com secret
a OK [CAPABILITY IMAP4rev1 LITERAL+ SASL-IR LOGIN-REFERRALS ID ENABLE IDLE
a select .Sent
a NO [CANNOT] Invalid mailbox name
a select INBOX.Test
* FLAGS (\Answered \Flagged \Deleted \Seen \Draft)
* OK [PERMANENTFLAGS (\Answered \Flagged \Deleted \Seen \Draft \*)] Flags
* 1 EXISTS
* 1 RECENT
* OK [UNSEEN 1] First unseen.
* OK [UIDVALIDITY 1367080883] UIDs valid
* OK [UIDNEXT 2] Predicted next UID
```

```
* OK [NOMODSEQ] No permanent modsequences
a OK [READ-WRITE] Select completed.
a STORE 1 +FLAGS Company
* FLAGS (\Answered \Flagged \Deleted \Seen \Draft company)
* OK [PERMANENTFLAGS (\Answered \Flagged \Deleted \Seen \Draft company \*)
* 1 FETCH (FLAGS (\Recent Company))
a OK Store completed.
a FETCH 1 FLAGS
* 1 FETCH (FLAGS (\Recent Company))
a OK Fetch completed.
a logout
* BYE Logging out
a OK Logout completed.
Connection closed by foreign host.
```

You should now find the file `dovecot-keywords` in this user's Maildir (or in the folder used here, `INBOX.Test`):

```
flash:~ # ls -la /srv/vmail/example.com/klaus/Maildir/.INBOX.Test
total 44
drwx------ 5 vmail vmail  4096 Feb  5 11:28 .
drwx------ 6 vmail vmail  4096 Feb  5 11:29 ..
drwx------ 2 vmail vmail  4096 Feb  5 11:28 cur
-rw------- 1 vmail vmail     8 Jun  1 16:04 dovecot-keywords
-rw------- 1 vmail vmail   115 Feb  5 11:28 dovecot-uidlist
-rw------- 1 vmail vmail 16384 Feb  5 11:28 dovecot.index.cache
-rw------- 1 vmail vmail   500 Feb  5 11:28 dovecot.index.log
-rw------- 1 vmail vmail     0 Feb  5 11:28 maildirfolder
drwx------ 2 vmail vmail  4096 Feb  5 11:28 new
drwx------ 2 vmail vmail  4096 Feb  5 11:28 tmp
```

And of course you will also find flag `d` in the Maildir file name now:

```
flash:~ # ls /srv/vmail/example.com/klaus/Maildir/.INBOX.Test/cur
1367080883.M622430P16295.flash.heinlein-support.de,S=547,W=559:2,Sd
```

7.3. mdbox: the new format for larger setups

The structure of an mdbox storage is very different from the familiar Maildir storage. It is subject to the following rules:

- The actual emails are stored in the `storage` directory in any number of numbered files with the name `m.<number>`, i.e. `m.1`, `m.2`, `m.3` etc.
- An `m` file can contain any number of emails but is only a few MB in size
- Every IMAP folder has its own file folder under the `mailboxes` directory; it contains files `dovecot.index.cache` and `dovecot.index.log`. Unlike in Maildir, these serve not purely for caching, but also contain essential information on the `m` file and part of the `m` file the email is located in. While it was not particularly dangerous to delete the `dovecot.index.*` files in the Maildir, doing the same here would damage the mdbox storage.
- While keywords and IMAP flags are saved in auxiliary files or in file names by means of tricks in the Maildir format, they are now saved directly in the index files in mdbox, because these have to exist as elementary components.

```
mdbox
├── dovecot.mailbox.log
├── subscriptions
├── mailboxes
│   └── INBOX
│       ├── dbox-Mails
│       │   ├── dovecot.index.cache
│       │   └── dovecot.index.log
│       └── friends
│           ├── dbox-Mails
│           │   ├── dovecot.index.cache
│           │   └── dovecot.index.log
│           ├── orchestra
│           │   └── dbox-Mails
│           │       ├── dovecot.index.cache
│           │       └── dovecot.index.log
│           └── vacation
│               └── dbox-Mails
│                   ├── dovecot.index.cache
│                   └── dovecot.index.log
└── storage
    ├── dovecot.map.index.log
    ├── m.1
    ├── m.2
    └── m.3
```

Manual manipulations are not permitted in an mdbox store. Whether you want to create and delete IMAP folders or delete and restore emails, you will always require the `doveadm` command when working on a user's mailbox:

`doveadm mailbox (create|list|delete|subscribe|unsubscribe)`
: for operations on the folders

`doveadm search`
: for searching emails

`doveadm expunge`
: for deleting emails

`doveadm fetch`
: for viewing or exporting individual emails from mdbox

The extensive `doveadm` documentation in Chapter 14 explains all this with concrete examples.

That is no problem for a "full-time" Dovecot administrator, but an all-rounder administrator who only takes care of the mail server from time to time will find that this takes some getting used to. It is therefore worth asking whether the advantages of mdbox are actually required and whether you want to burden yourself with its more complex operation.

I advise many of my customers to continue working with Maildir. As long as the backup process does not become a sustained problem due to numerous small files, and as long as you have no noticeable performance problems, it is not really worth switching to mdbox.

It is difficult to say in which situations mdbox is worth using; it depends on the (hard drive) hardware in use, the backup procedure and the degree of user activity. For myself, I have developed the (rough) rule of thumb that file-based backup processes from 3-4 TB of Maildir volume become unpleasant – and that the use of mdbox should be considered from this point onwards.

7.3.1. Why mdbox is faster

In larger setups, where drive I/O is the limiting factor in the scaling of the email system, mdbox is highly suitable as a storage format if you can accept the slightly more complex administration:

Backups are fun again

As mentioned before, mdbox avoids the major drawback of the Maildir format, which is large numbers of small files. With mdbox you can perform file-based backups quickly and easily. Many `m` files containing old emails are often no longer modified, so incremental backups function exceptionally well and space-savingly in mdbox. Careful: the `m` files and `dovecot.index.*` files now depend on one another, so it must be ensured that there is a consistent state when the backup takes place.

Email deletion is I/O-optimized

Deleting individual emails generates much less I/O in mdbox than in mbox. For this reason alone, mdbox is far more efficient than mbox.

In addition, emails are not actually deleted in mdbox; instead, they are flagged for deletion. The actual expunging is performed by the `doveadm purge` command, which you have to launch regularly using a cron job (for more information, see Section 7.3.3). That allows you to schedule the I/O load caused by the deletion of emails late at night – or even during the low-load times at the weekend.

At the same time, an expunge run clears up the user's mailbox in one go, which also optimizes a large chunk of I/O load.

Optimized file access

Upon request, mdbox can preventively reserve the maximum file size for new `m` files in the file system. That is intended to prevent fragmentation of the file system, as files will not grow incrementally. On the other hand, this also requires far more space on the hard drive.

7.3.2. Configuration of mdbox

There is not much to set up for mdbox: file `10-mail.conf` knows only three mdbox-specific parameters:

```
##
## mdbox-specific settings
##

# Maximum dbox file size until it's rotated.
#mdbox_rotate_size = 2M

# Maximum dbox file age until it's rotated. Typically in days. Day begins
# from midnight, so 1d = today, 2d = yesterday, etc. 0 = check disabled.
#mdbox_rotate_interval = 0

# When creating new mdbox files, immediately preallocate their size to
# mdbox_rotate_size. This setting currently works only in Linux with some
# filesystems (ext4, xfs).
#mdbox_preallocate_space = no
```

The following recommendations are based on my practical experience:

- A file size of 2 MB is rather conservative by today's standards, and leads to an unnecessarily large number of files being created. I use the setting `mdbox_rotate_size=10M` and have found it a good balance between performance and quantity of files.
- You should consider whether you want to have your mdbox files rotated automatically according to age. I do not use this feature. There are not many advantages in terms of backup; instead, I keep the number of files to a low(er) number.
- If you set `mdbox_preallocate_space=yes`, Dovecot will automatically create every new m file in the size specified in `mdbox_rotate_size`, i.e. 2 MB or (if you follow my suggestion) 10 MB. That should reduce file system fragmentation, but it does take up noticeably more hard drive space at first. I don't use this feature in practice; instead, I rely on the administrative power of a good file system.

7.3.3. Making space: doveadm purge

When a user uses IMAP to delete an email, it is equipped with the `\Deleted` IMAP flag but not actually deleted. The mail client uses the `EXPUNGE` IMAP command to inform the server that the flagged messages are now to be physically deleted. Some mail clients therefore offer the option of continuing to display deleted emails (as marked for deletion), while others simply flag and delete the emails in one go – that depends on the settings.

The mdbox format (and it is the only one) offers an additional stage before the email is actually deleted for ever. In mdbox an `EXPUNGE` command marks the email as deleted in the Dovecot index but does not release it from the `m` files. That means that the hard drive space continues to be occupied. Only the special `doveadm purge` command cleans up a user's `m` files, copies everything and then releases the hard drive memory.

Just to repeat, so that there are no misunderstandings: in mdbox, there are *three* levels before an email is actually deleted:

1. The `\Deleted` flag – the user is still able to see and restore the email.
2. The `EXPUNGE` IMAP command – the email is irretrievably lost for the mail client, but still exists in the `m` files as a "dead data block".[40]
3. The `doveadm` command `purge` compresses the `m` files and actually removes the dead data.

If you use mdbox, you *must* set up a cron job that executes `doveadm expunge` for all users. Run it at a time when there is not much going on – so usually at night. You do not have to run `doveadm expunge` every day; better run it late at night at the weekend, or whenever it is least in the way.

If your user authentication is set up so that the `doveadm user '*'` command returns a list of all available users (Section 5.13), you only need the following little shell script to instruct `doveadm` to execute the `expunge` command for all users:

```
logger "Start Dovecot-Purge"
doveadm purge -A
logger "End Dovecot-Purge"
```

If you are unable to access the list of all users by means of the `doveadm` parameter `A`, take a look at the file system instead and have the `doveadm` command executed for all available email directories:

```
# Adapt path to your email directories
MAILPATH=/srv/vmail

logger "Start Dovecot-Purge"
cd $MAILPATH
```

```
# The script assumes /srv/vmail/example.com/susi
for DOMAIN in * ; do
    cd $MAILPATH/$DOMAIN
    for USER in * ; do
         if [ -e $USER/mdbox ] ; then
             /usr/bin/doveadm purge -u $USER@$DOMAIN
         fi
    done
done

logger "End Dovecot-Purge"
```

Create the script as `/usr/local/sbin/dovecot-purge` and use a cron tab or an entry in `/etc/cron.weekly` to regulate the daily or weekly call.

By the way, the `logger` command ensures a suitable entry in `/var/log/messages` or `/var/log/syslog`, depending on the distribution. You can use the `logger -p mail.info <text>` to have the log entered directly in your email log file if you prefer.

7.3.4. ALT storage: fast and slow data storage combined

On large mail servers , you have to competing interests:

Up-to-date data require speed
: You need fast storage as all the delivered, retrieved and in some cases deleted emails involve a heavy I/O load. However, fast storage usually means small and fast drives (= expensive), possibly even in a fast RAID such as RAID-10 (=inefficient).

Archived data require volume
: At the same time, you need a lot of memory, as many users keep their emails for years and mailboxes several GB in size are not unusual. These ancient emails slumber in the depths of some IMAP folders and are very rarely retrieved thanks to caching IMAP clients (in fact, they are usually retrieved only when the user sets up a new mail client). You would prefer to use large hard drives (= cheap) for these data, ideally in an effective (but slower) RAID-5.

In practice, the required I/O load forces us to equip the entire email storage of an IMAP server with fast small hard drives.

But if you use the mdbox format, Dovecot has an intriguing answer that can quickly save tens or hundreds of thousands of dollars. Dovecot is able to use the fast main memory (index files and current data) as well as a second memory, the ALT storage (*alternative storage*), i. e. a second file path that ends up on a slower/larger/cheaper storage system.

As explained in Section 5.9, it is a good idea to assign every user a `$HOME` in the authentication process and use it as the path in `mail_location` – whether you refer to it using a swung dash `~` or the variable `%h`:

```
mail_location = mdbox:%h/mdbox
```

I advise you to simply prefix this `$HOME` path for the definition of the ALT storage part, which will allow you to continue working with `~` or `%h`. In this case, the user would store his emails under `/srv/vmail/example.com/klaus` and under `/altstorage/srv/vmail/example.com/klaus`:

```
mail_location = mdbox:%h/mdbox:ALT=/altstorage/%h/mdbox
```

Alternatively, you could of course use the familiar placeholders `%d`, `%n` or `%u`:

```
mail_location = mdbox:/srv/vmail/%d/%n/mdbox:ALT=/altstorage/vmail/%d/%n/m
```

As an `m` file may by definition exist in only one of the two paths, the administration is conceivably easy for Dovecot. It determines from its index files what the `m` file is called where the email is stored. It does not need to know where the `m` file is stored. It simply looks first in the main storage and then in the ALT storage. For this reason, Dovecot does not need to record in its index storage whether an email is in the fast or slow storage.

You as the administrator can determine which emails are saved in the (slow) ALT storage. The `doveadm altmove` command moves all emails to the ALT storage that match a specified search pattern. You want to swap out emails to there that are retrieved only rarely, so you will typically include a) read and b) older emails – in this example, the emails included were saved more than two weeks ago and have already been read:

```
flash:~ # doveadm altmove -u klaus@example.com seen savedbefore 2w
```

If you run this command on an existing mdbox storage of a test user, you will see that Dovecot has created new `m` files in the ALT storage and moved the affected emails to there.

By the way, the `r` parameter is used to instruct `doveadm` to move the emails matching the search filter from the ALT storage back to the main storage. That means that you can dissolve your ALT storage and move all emails back to the main storage with a single command:

```
flash:~ # doveadm altmove -A -r all
```

You should seriously consider whether the use of ALT storage makes sense for you. The advantage is that you can use fast and slow storage at the same time; in addition, it is nice to be able to check two smaller partitions than one large one when performing a file system check.

On the other hand, you have to allow for this special constellation in all your management and evaluation scripts, during backups and in all other everyday administrative matters – unless you perform your tasks using `doveadm` commands: in that case, Dovecot handles the separation of the two storage locations in an unnoticeable and fully transparent manner.

7.4. zlib compression on the fly: faster and more space-saving

For storage formats Maildir and mdbox, Dovecot has the ability to save and read all emails in a compressed state via a zlib library.[41] One thing is very interesting: from the outside, you do not notice anything, as Dovecot takes care of the compression and decompression in a transparent fashion – which means that email clients continue to have access to their mailboxes as normal.

I consider the use of zlib on mail servers of all sizes a matter of course; there is no reason why you should not use it, and there are many advantages:

- zlib compression saves space. The amount of space it saves depends on how the email data can be compressed. Experience has shown, however, that you can reduce space consumption by around 40% – which also has financial effects in larger systems due to savings in storage.
- You will also notice this space saving during backup, which will also run more quickly if there are fewer data to transfer.
- As the data are read and written in compressed form, the hard drive system is relieved – and the I/O in the storage is usually the most important bottleneck that limits the performance of an IMAP server.
- The hard drive cache of the Linux kernel, RAID controller and/or the hard drive itself is also used far more efficiently, because it now contains data that have already been compressed. To put it another way: you have practically doubled the size of the cache in one go. Improved caching speeds up access – and, as it is possible to keep more data in the cache, a higher hit rate in the cache will also relieve the hard drive system from the strain of read operations.
- You could object that compression in turn leads to a corresponding CPU load. That is true, but modern CPUs (unlike hard drives) are so powerful that this small amount of CPU load is a worthy investment in order to noticeably relieve the comparatively slow storage system.

It is pretty simple to activate zlib compression. First, add the `zlib` plugin to the plugin section in `10-mail.conf`:

```
mail_plugins= [...] zlib
```

Then configure the `zlib` plugin by making an entry in the plugin section of file `90-plugin.conf`:

```
plugin {
  zlib_save_level = 6 # 1..9
```

```
  zlib_save = gz # or bz2
}
```

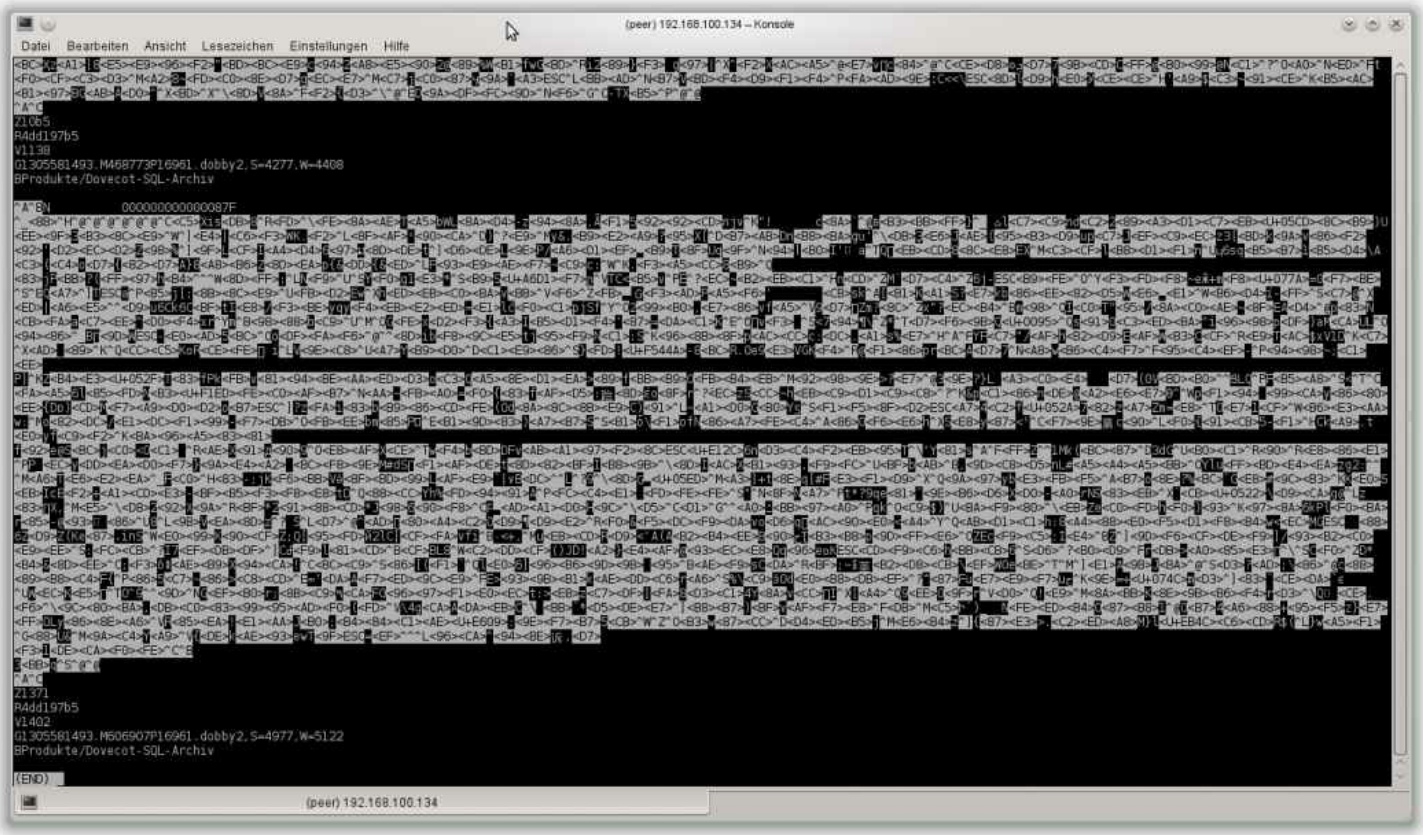
You can use the `bzib2` and `gzip` procedures; in my experience, the (older) `gzip` is three times more efficient while the compression it provides is not really any worse. I therefore urgently advise you to use `gzip` (as shown here) at compression level 6, because I believe it provides the best balance between performance and benefit.

As soon as Dovecot has been restarted, newly saved emails will automatically be compressed using zlib. By the way, mixed operation is harmless in all storage formats. It is not a problem if some emails are compressed while the existing data remain uncompressed. Dovecot always reads the first bytes of an email and can then decide whether decompression is necessary. I will show you how to compress existing data during live operation in Section 7.5.

If emails are saved in the Maildir format, every email is already available as an individual file – and it takes no effort to compress it with gzip. That means you could compress it retroactively with gzip – or use `zless`, `zcat` or `zgrep` to view or search it, for example.

In mbox and mdbox that is slightly more complicated. Here, multiple emails are saved in sequence in one file, and instead of compressing the entire email file, every individual email is compressed and saved in the email file. If you use mdbox, you will notice this very quickly when you open an `m` file with `less`: here the zipped data of the emails are saved as BLOB – visibly separated by a Dovecot plain text header provided in ASCII (see Figure 7.1).

Figure 7.1. The emails are compressed individually and saved as BLOB, interposed by Dovecot headers in plain text



Maildir, zlib and Debian 7 Wheezy

I urgently advise you not to use zlib if your emails are stored in the Maildir format in Dovecot versions around 2.1.17, like those supplied by Debian 7 Wheezy. A bug that has not been fully explained to date results in destroyed index files and even lost emails.[42] In some independent installations, I had huge problems in Debian-7 systems when using Maildir/zlib that made the affected systems unusable because empty emails were delivered to users. The use of Maildir/zlib works for Dovecot version 2.2.x and higher, so make sure you are using a recent version. By the way, use of zlib under mdbox was not affected; it worked smoothly even with version 2.1.17.

If you have activated `zlib` in general, you can activate the additional plugin `imap_zlib` in `20-imap.conf`. This plugin has nothing to do with the saving of emails, but it does enable the client and server to agree on an on-the-fly compression of the IMAP transmission in order to minimize the required bandwidth:

```
protocol imap {
  # Space separated list of plugins to load (default is global
  # mail_plugins).
  mail_plugins = $mail_plugins imap_zlib
}
```

7.5. auto: migration of storage formats during live operation

It is no problem to convert data in the mbox, Maildir or mdbox formats to other formats while the system is live. The `dsync` tool performs this task and also retains the UIDs and flags of the emails, so that the conversions are performed in a manner that is transparent and unnoticeable for the email client.

Later zlib compression of an existing mailbox is also no problem if the method described here is used. You do not need to change the storage format; simply convert Maildir to Maildir or mdbox to mdbox. When writing the email data to the new directory, Dovecot will convert the data on the fly (if the zlib plugin is activated). If the data are being compressed for the first time, this will involve a fair CPU load, and the conversion process will take a while. But there is nothing wrong with that, as you can run the conversion script below for days while the system is running and are therefore in no hurry.

7.5.1. auto: parallel operation of multiple storage formats

Migration while the system is live also means that different storage formats are in operation at the same time and fixed definition by means of the `mail_location` parameter in `10.mail.conf` no longer works. I am therefore grateful to Timo Sirainen for enabling Dovecot from version 2.0.16 onwards to automatically recognize the storage format of a user's email directory when that user logs in.

How? It searches the user's home directory in the order described here for directories named `mdbox`, `sdbox`, `Maildir`, `mail`, `Mail` or `mbox` and then assumes the corresponding storage format if any directory or file structure it contains also looks meaningful.

For this purpose, simply define `auto:` instead of the `Maildir:` or `mdbox` prefix in `mail_location` – but be careful: `auto` may not be followed by any further entries, and the definition of a path is now *not* allowed any longer:

```
# mail_location=Maildir:~/Maildir
mail_location=auto:
```

It is therefore essential that a clean home directory is set for the user as the result of the authentication process (Section 5.9).

That way, it is easy to operate various formats for different users in parallel and therefore also perform the format conversion described below during running operation as long as you ensure that the email directories are named properly. A large import of existing email directories from another system in the "wrong" format is therefore not a problem.

But there is one problem you need to consider. If the `auto` mode is active, Dovecot is unable to determine a format for new users without an email directory – and is also unable to simply create non-existent email directories. Emails addressed to new users without a home directory therefore fail in LMTP delivery with a temporary error and remain in the Postfix queue. If you have set `mail_debug=yes`, you can see how Dovecot searches for directories and finally gives up:

```
mdbox: access(/srv/vmail/example.com/klaus/mdbox, rwx): failed: No such fi
mdbox: couldn't find root dir
sdbox: access(/srv/vmail/example.com/klaus/sdbox, rwx): failed: No such fi
sdbox: couldn't find root dir
maildir: access(/srv/vmail/example.com/klaus/Maildir, rwx): failed: No suc
maildir: couldn't find root dir
mbox autodetect: has .imap/: stat(/srv/vmail/example.com/klaus/mail/.imap)
mbox autodetect: has inbox: stat(/srv/vmail/example.com/klaus/mail/inbox)
mbox autodetect: has mbox: stat(/srv/vmail/example.com/klaus/mail/mbox) fa
mbox autodetect: has .imap/: stat(/srv/vmail/example.com/klaus/Mail/.imap)
mbox autodetect: has inbox: stat(/srv/vmail/example.com/klaus/Mail/inbox)
mbox autodetect: has mbox: stat(/srv/vmail/example.com/klaus/Mail/mbox) fa
mbox: couldn't find root dir
sdbox: access(/srv/vmail/example.com/klaus/sdbox, rwx): failed: No such fi
```

```
sdbox: couldn't find root dir
Error: user klaus@example.com: Initialization failed: Namespace 'INBOX.':
```

If your migration takes several days or even weeks, you have to consider how to handle that. I have gotten into the habit of checking the email queue for such failed deliveries every minute by means of a cron job and then extracting the email addresses involved.

Once you have done that, it is easy to create an `~/mdbox/storage` directory with matching file permissions for these users. These two empty directories are enough for Dovecot to recognize an `mdbox` structure for this user and create the missing directories and files automatically. Postfix can then successfully deliver the emails shortly afterwards.

Alternatively, you can also initiate a process that creates the required email directory when the user is created in LDAP or SQL.

7.5.2. A solid migration script as an example

Once the `auto:` mode is active, you can adapt my migration script and tackle the conversion of your email directories.

The script assumes that the email directories are structured in the style of `/srv/vmail/example.com/user/Maildir`. With two nested `for` loops, it dips into the list of all domains and then into the list of all existing users, so that it runs once for every email directory. This example is designed to convert data in the Maildir format to the new mdbox format:

```
cd /srv/vmail

for DOMAIN in * ; do
 cd /srv/vmail/$DOMAIN

 for USER in * ; do
  cd /srv/vmail/$DOMAIN/$USER

  if [ -e /root/STOP ] ; then exit 0 ; fi
  if [ -e /root/SLEEP ] ; then sleep 1h ; fi

  if [ -d Maildir ] ; then
   if [! -e sync.LOCK ] ; then
    touch sync.LOCK

    dsync backup -u $USER@$DOMAIN mdbox:/srv/vmail/$DOMAIN/$USER/mdbox-NEU
    dsync backup -u $USER@$DOMAIN mdbox:/srv/vmail/$DOMAIN/$USER/mdbox-NEU

    mv Maildir Maildir-OLD
    mv mdbox-NEW mdbox

    doveadm kick $USER@$DOMAIN
    dsync mirror -u $USER@$DOMAIN maildir:/srv/vmail/$DOMAIN/$USER/Maildir

    doveadm quota recalc -u $USER@$DOMAIN

    doveadm kick $USER@$DOMAIN

   fi
  fi

 done
done
```

I have gotten into the habit of including some protective measures into such migration scripts so that the script can continue to run safely for several days or even be called up several times in a row without any danger:

- You can use `touch /root/STOP` to create an (empty) STOP file so that the shell script ends safely (that may take a few minutes, and the conversion of the account currently

being processed is of course completed).

- If the `/root/SLEEP` file exists, the script will fall asleep for an hour. You can use a cron job to create the `SLEEP` file in the morning and have it deleted again in the evening to prevent any conversions being performed during high-load times and thereby placing unnecessary strain on the system.
- A check is first performed in every email directory to determine whether there is a `sync.LOCK` file. If there is, the user directory in question is skipped. That way, there are no problems if the script is started several times or restarted.

The actual conversion process then takes place as follows:

- `dsync backup` writes the entire email storage of the user to `mdbox-NEW` in the mdbox format, i.e. parallel to the existing mbox or Maildir folder of the user (if you want to convert to a format other than mdbox, you have to adapt that here).
- The first run of `dsync backup` can take several minutes – particularly if zlib compression is performed for the first time. That is why a second run of `dsync backup` is performed immediately after the first one – this time, `dsync` works incrementally on the data it has already converted, therefore has very little to do and takes just a few seconds.
- Then the current `Maildir` directory is renamed `Maildir-OLD` and the new `mdbox-NEW` directory is renamed `mdbox`. That means that the active directory and the inactive directory change places. It does no harm if the user in question is logged in at this point in time. With the file handles that are already open, he continues to access his email files in `Maildir-NEW`.
- In order to complete the migration, `doveadm kick` terminates the POP3/IMAP connection to the email client. If the user logs in again a few seconds later, he will then access the new `mdbox` directory from that point onwards.
- Theoretically a new email could have entered the `Maildir` directory at the last second and therefore not have been converted to the new `mdbox` directory by `dsync backup`. So we launch a new `dsync` run, which, however, is performed as `dsync mirror` and transmits all changes in both directions. That means that changes that are still in `Maildir-OLD` are now transferred back to the active `mdbox` directory.
- The conversion usually has the result that any active quota calculations erroneously calculate the user's volume twice, so emails may be rejected because of an apparently full mailbox. It has proven useful to recalculate the user's quota on principle once the migration has ended.
- Finally a `doveadm kick` takes place at the end; if the user has logged in again (which is usually not the case after such a short time), his connection is terminated once more. That ensures that the user sees the most recent status of his mailbox properly and in full the next time he logs in.

Once the migration has been completed:

- The `Maildir-OLD` directory remains on the hard drive as data trash – that means that

more storage space is required than before. Wait a few days before deleting the `Maildir-NEW` directories of the users. That way, you can fall back on the original data in the event of user complaints or any detected conversion problems (after all, the backup may already be quite old). Once the dust has settled and no problems have emerged after several days, you can delete all `Maildir-NEW` directories; some time later, you can also delete the `sync.LOCK` files and thereby conclude the migration.

- It may be necessary for you to adapt the configuration of `mail_location=` to your defined storage format, i.e. use `mail_location=maildir:[...]` or `mail_location=mdbox:[...]` instead of `mail_location=auto:`. That way, Dovecot will once again be able to create missing home directories automatically (see above).

By the way, you can easily perform a test run of this script with a few chosen accounts. In the two `for` loops, replace the `*` with your domain or a list of test users:

```
cd /srv/vmail

for DOMAIN in  example.com  ; do
   cd /srv/vmail/$DOMAIN

   for USER in   klaus admin test dummy user  ; do
      cd /srv/vmail/$DOMAIN/$USER

[...]
```

That way, you can also approach the migration incrementally, e.g. beginning with all domains that start with `a` to `h`:

```
cd /srv/vmail

for DOMAIN in  [a-h]* ; do
   cd /srv/vmail/$DOMAIN

[...]
```

There is one last piece of advice you should follow: it is always possible for something to go wrong during migrations. Make sure that you can read up what has happened, as you will be looking for a needle in a haystack – particularly if the migration process involves many users and days:

- First, launch a `screen` session so that the script will continue to run properly once you have logged off.[43]
- Then launch the migration script in such a way that all STDOUT and STDERR messages are written to a log file, for example by means of a call in the style of `migrate-maildir2mdox.sh &> /root/migrate-20131023.log &`. You can start the script in the background by means of `&` (and still stop it conveniently by means of `touch /root/STOP` (see above)). A `tail -f /root/migrate-20131023.log` allows you to read along and monitor the migration process.
- At the end, use `grep -i error` and `grep -i fatal` to find anything that has gone

wrong.

---

[34] mbox is defined in RFC 4155, Maildir is based on RFC 2822 email files.

[35] A long time ago, write locks in NFS were a common problem and were sometimes simply not implemented.

[36] You could use the `LAYOUT=fs` parameter to persuade Dovecot to actually nest the folders.. On some ancient mail servers, the Maildir folders are set up in that way, and in that case you can adapt Dovecot to the existing file system layout. However, I think it is better to rename the folders by means of a shell script and then set up a traditional Maildir structure as described in this chapter.

[37] See http://cr.yp.to/proto/maildir.html and http://www.qmail.org/man/man5/maildir.html

[38] At least, that is the RFC specification of Maildir; I have never actually encountered experimental flags and therefore never found Maildir emails containing `:1,` in their file name.

[39] Unfortunately, not many clients support these keywords as yet, so they lead a rather shadowy existence.

[40] From Dovecot 2.2.x onwards, the `doveadm import` provides the option of importing these deleted messages to a separate IMAP folder and therefore restore them.

[41] In the mbox storage format, only read access to compressed files is possible, so it is rarely useful for practical application. But who wants to use mbox…

[42] After compression, Dovecot no longer determines the correct size of the emails correctly, declares incorrectly that the index files are corrupt and therefore repairs them incorrectly. The message in the log file is `Error: Maildir filename has wrong S value`, see our bug report on http://www.dovecot.org/list/dovecot/2013-March/089232.html

[43] `screen` is a brilliant tool: it starts a shell session that keeps running along with all the programs running in it even if you log off from the screen session using `[Ctrl]+[A]` and then `[D]` or if your underlying SSH session was terminated. Unlike in the `nohup` command, all programs can also run in the foreground within the `screen` session and you can rejoin existing screen sessions at any stage by using `screen -x`. That means you can start shell sessions that you can rejoin from a different location and even days later. Finally, it is possible for several people to log in to a `screen` session at the same time. It is a kind of Teamviewer for shell sessions. Excellent for telephone conferences and cooperative administration.

Chapter 8. Partitions, file systems and downtimes during the file system check

In this chapter, I will describe the best way to create the user data partition of your mail server and secure it against errors. I will also highlight the benefits and drawbacks of a variety of file systems. One main consideration will be how long your system is offline when you perform a file system check. There are some shocking results.

8.1. /srv/mail as data partition

In a larger system, it is very desirable to have a separate data partition for the storage of emails. The mbox or Maildir files of a local shell user are typically stored under `/var/spool/mail` or the user's home directory. For virtual users from SQL databases, LDAP or local `passwd` files you will prefer a separate storage path to prevent them getting mixed up accidentally.

You could easily argue in favor of a storage location or mount point `/var/spool/vmail` (in contrast to `/var/spool/mail`). However, I prefer path `/srv/vmail` (which has just as many reasons in favor of it) because I believe it provides better system monitoring as it is better at keeping the email volume separate from `/var`. LVM is appropriate for simple extension purposes (the volume of emails usually increases steadily). If you do not want to use LVM, you should, in virtual systems, at least swap out the email partition to a second (virtual) hard drive so that the partition is easier to extend or integrate in another system later on.

The following examples assume that you use a single user and group ID named `vmail` for your email users as described in Section 5.8.

8.1.1. Email data in a separate subdirectory

If you do not want to provide a separate partition for your email data, the setup is pretty simple:

1. Use `mkdir /srv/vmail` to create the future storage location.
2. Now give Dovecot write permissions for the directory so that it can create the home directories for new users; at the same time, make sure that other users are not unintentionally given access to it.

```
flash:~ # mkdir /srv/vmail
flash:~ # chown vmail:vmail /srv/vmail
flash:~ # chmod 770 /srv/vmail
```

8.1.2. Email data on a separate data partition

A separate data partition requires a little more effort than a simple data directory. Whether you use NFS or just a separate partition: you now have to assume that the data partition could be unmounted at some stage and the data directory could therefore be empty. Therefore:

- When the data partition is mounted, Dovecot should be able to automatically create new email directories for users. Dovecot requires corresponding write permissions for this purpose.
- If you use a separate partition, Dovecot should be unable to read or write to the mount point in the event that the data partition is *not* mounted. If the email data are missing, it is better for Dovecot to terminate the connection to the client than to present the user with an (apparently) empty directory. Or (worse) begin to save new emails to the empty data directory, so that you have to bring this *split brain* together again. Or (even worse) if Dovecot finds that numerous `m` files are missing and it therefore assumes it has to clear up its index and consequently deletes the non-existent emails permanently from the index.

In such cases, you should therefore proceed as follows:

1. Use `mkdir /srv/vmail` to create the future mount point.
2. Make sure that Dovecot has no access to the mount point in unmounted condition by using `chown root:root /srv/vmail` and `chmod 700 /srv/vmail` to configure the directory so that only `root` can write to it.
3. Set up the file system of your choice on your data partition and mount it to `/srv/vmail`.
4. Now give Dovecot access permissions to the mounted(!) directory by executing `chown vmail:vmail /srv/vmail` and `chmod 750 /srv/vmail` in mounted condition.

If the partition is mounted, the access permissions specified in the partition will overwrite those of the mount point directory from the root partition. If it is not mounted, the underlying access permissions will appear and permit only `root` access while locking out Dovecot.

Here is an example for a ext4 partition in the LVM; you will of course have to modify the `mfks` command to suit your setup:

```
flash:~ # mkdir /srv/vmail
flash:~ # chown root:root /srv/vmail
flash:~ # chmod 700 /srv/vmail
flash:~ # mkfs.ext4 /dev/mapper/data-vmail
[...]
flash:~ # mount /dev/mapper/data-vmail
flash:~ # chown vmail:vmail /srv/vmail
flash:~ # chmod 770 /srv/vmail
```

You can easily see the difference if you view the file permissions in mounted and unmounted condition:

```
flash:~ # ls -lad /srv/vmail
drwxr-x--- 7 vmail vmail 4096 Jan 26 16:22 /srv/vmail
flash:~ # umount /srv/vmail
flash:~ # ls -lad /srv/vmail
drwx------ 7 root root   4096 Jan 26 16:22 /srv/vmail
```

Don’t forget to add your data partition to `/etc/fstab` in order to prevent any surpises after the next reboot.

8.1.3. doveadm mount: in case something is missing

I should not forget to mention that Dovecot also provides a mechanism that detects missing (non-mounted) email partitions. Dovecot logs in `/var/lib/dovecot/mounts` which email directories it has already seen in its lifetime on separate partitions:

```
flash:~ #  cat /var/lib/dovecot/mounts
online /
online /mail
online /altstorage
```

Do not maintain this file manually; you can use the `doveadm mount` commands here:

```
flash:~ # doveadm mount list
  path                                  state
  /                                     online
  /altstorage                           online
  /mail                                 online
  /oldmail                              online
```

If a mount point is not available, Dovecot indicates that fact using an exclamation point, as it does for `altstorage` in this case:

```
flash:~ # doveadm mount list
  path                                  state
  /                                     online
! /altstorage                           online
  /mail                                 online
  /oldmail                              online
```

Use `doveadm mount add` to manually add a path and `doveadm mount delete` to remove any mount points from monitoring if they no longer exist.

```
flash:~ # doveadm mount remove /oldmail
```

Even though Dovecot provides its own security level here, I recommend that you prevent access to an empty mount point by means of suitable file permissions. Just to be on the safe side.

8.2. Choosing the right file system

In small mail servers for 20 users, the way in which the operating system organizes data on the hard drive does not really affect performance. You can in that case skip the rest of this chapter and just use your favorite file system.

But if you have to react quickly to many thousands of users simultaneously as an internet or email service provider, you will want to get the best out of your file system as well.

Above all lies the format where Dovecot saves its email files, so conveniently the choice is between Maildir (see Section 7.2) and mdbox (see Section 7.3). But the file system layer underneath also provides potential for improvement – and you have to decide which file system you want to entrust your email pool to.

That takes you to a faith war that will probably never be decided: what is "the best" file system?

ReiserFS
: About 10 years ago, ReiserFS was praised as a highly efficient system, but there is no longer any true development. ReiserFS was easily overtaken by its competitors in terms of speed a few years ago; in my file system benchmark, which you will find below, it takes last place and is far behind all its competitors. There is really no longer any reason to use ReiserFS. Regardless of that, I have repeatedly encountered blatant instabilities in ReiserFS and would never trust this wannabe file system with data that are important to me. I think ReiserFS is out of the question, no matter on which system, and I am always surprised when I encounter this file system on a customer's productive system.

XFS
: Quite a few people swear by XFS. It seems you only learn to appreciate it when you spend some time with it. It comes first in terms of speed and is considered very robust. However, I have experienced XFS file systems in three different customer projects that were reproducibly corrupt after a file system check even though XFS is deemed to be very stable. Features such as reducing the size of an existing file system are unfamiliar to XFS (unlike ext3/ext4) – only an increase in size is possible.

ext3/ext4
: ext4 is now part of all distributions, and it is my favorite file system. It is robust and fast – but above all, it is easy to debug and repair in the event of a crisis; we have often been able to save the majority of the data in file systems that have been severely destroyed. Its performance is very good, even for small files, and has improved considerably on ext3, as the ext4 code in the kernel has been rewritten and can work in parallel in many areas. Just mounting an existing ext3 partition as an ext4 partition (which is easy to do) can improve the speed. Nothing has changed in the actual structure of the file system, but now the (downwardly compatible) ext4 module of the Linux kernel comes into play.

btrfs
: This latest file system is currently available only in unstable beta versions. It has some very interesting properties, but our performance test demonstrates clearly that it does not currently have the performance (yet) to be used as a server file system with its default values.

NFS
: NFS is not a file system in this sense, because it does not save the data to the hard drive. Instead, NFS exports a file system to the network – and the question of the file system you run locally on the NFS server remains unanswered. If you plan to use NFS, make sure you read my explanations and especially my explicit warnings in Section 16.3.

There are plenty of different assessments and tests of the speeds and advantages of various file systems, and often enough they are not just confusing but actually contradictory. Different kernel versions can result in considerable differences. In addition, different distributions configure the Linux kernel and the file system drivers in very different ways, so the results also vary widely.

Reason enough to set up my own benchmark under the conditions that usually occur on a mail server.

8.3. Measuring performance

The well-known measuring tools `iozone` and `bonnie` or `bonnie++` are only of limited use when testing file system performance on mail servers, as the read and write accesses to the data carrier do not alternate enough. The `postmark`[44] tool written by NetApp, on the other hand, simulates the way that mail and news servers work. During the measurement, multiple small files are read, written and deleted in alternating order as is common in the Maildir format. Of course, the test results also depend on the way the test is designed, but their language is clear, as Table 8.1 shows.

Table 8.1. A comparison of various file systems

10,000 RPM SATA	ReiserFS	btrfs	ext3	ext4	XFS
transactions/sec.	129	202	330	679	877
files created/sec.	87	180	241	458	665
files deleted/sec.	64	101	164	339	437
read accesses/sec.	64	101	164	339	438
write accesses/sec.	87	180	241	458	665
read throughput MB/sec.	0.288	0.59	0.799	1.45	2.15
write throughput MB/sec.	0.538	1.09	1.46	2.77	4.02
duration of test run in min.	86:00	41:40	31:13	16:22	11:17
CPU consumption “user” in min.	0:10	0:09	0:08	0:07	0:08
CPU consumption “sys” in min.	2:06	1:24	0:46	0:41	1:00

The measurements from Table 8.1 were created in version 1.51 with `postmark` and took place under the following conditions:

- The operating system was the default kernel of openSUSE 13.1 in version `3.11.6-4-default`.
- The measurements took place on a standard server with an Intel QuadCore-CPU L5410 2.33 GHz with 32 GB RAM.
- The hard drive system consisted of two hard drives WD500HHTZ 500 GB SATA with 10,000 RPM on the local RAID-1 controller.
- All file systems had just been formated in the standard configurations of openSUSE 13.1.
- The measurements were performed by the `postmark` tool in version 1.51 in the following test design: 200,000 files between 5 and 50 KB in size in 10,000 directories with 500,000 transactions.

The bad test results should not be held against btrfs, because it is currently in the beta status (here: version 0.2 RC1). You should not apply these measurements to btrfs for ever, but rather

update with a newer version when you get the opportunity. However, ReiserFS fell a long way behind – it is clear to see that this file system has not been in development for years. The rumor that ReiserFS is very powerful was true 10 years ago, but the opposite is true today.

8.4. Performance tuning the file system

Speed is affected not just the choice of file system, but also by *how* that file system is used. A change to the configuration can improve speed slightly, and that is always a good thing, especially if there are no drawbacks.

8.4.1. atime

By default, every read access to a file also involves a write access, as Linux saves not one but *three* date inputs per file:

Modification time (`mtime`)
: this date is also displayed in the file list and describes the time when the file content was last changed.

Change time (`ctime`)
: saves the date of the last change to file permissions or file ownership. A `chmod` or `chown` command will ensure that a new `ctime` is set while leaving the `mtime` unchanged.

Access time (`atime`)
: shows the date of the most recent *read* access. Every read access, even a simple `cat`, causes a new `atime` to be saved.

That is a small waste of performance. Even though the `atime` is usually just placed in the write cache, it involves unnecessary administration and hard drive I/O, even though the client simply retrieved a few emails.

Table 8.2. The positive effects of `noatime` are clear to see in any file system

10,000 RPM SATA	**xfs**	**xfs**	**ext4**	**ext4**
	`noatime`	`atime`	`noatime`	`atime`
transactions/sec.	892	877	800	679
files created/sec.	693	665	513	458
files deleted/sec.	445	437	399	339
read accesses/sec.	446	438	400	339
write accesses/sec.	693	665	513	458
read throughput MB/sec.	2.24	2.15	1.66	1.45
write throughput MB/sec.	4.18	4.02	3.10	2.77
Duration of test run in min.	10:49	11:17	14:37	16:22
CPU consumption “user” in min.	0:08	0:08	0:07	0:07
CPU consumption “system” in min.	1:00	1:00	0:41	0:41

On servers you are usually not interested in the `atime`, so there is no reason not to deactivate it.

Whether you are using ext3, ext4, xfs, btrfs, NFS or another file system, mount the data partition for it with the `noatime` option and/or enter this option in `/etc/fstab`. The Linux kernel will then leave the `atime` unchanged when read access takes place. There are no drawbacks.[45]

```
/dev/mapper/data-vmail    /srv/vmail    ext4    defaults,noatime  0 0
```

8.4.2. The journal mode in ext3/ext4

The ext3/ext4 file system knows three different journaling modes:

`journal`
: This mode guarantees the greatest data security, as the data are first written to the journal and then to their final destination. Naturally this is the slowest mode, as the test proves.

`ordered` (default)
: The kernel writes the data to the destination immediately and then enters the information on the completed action in the journal.

`writeback`
: Unlike in the other two methods, the order in which the data are written to the journal and the usage data area is not specified.

  This method in principle provides just as much protection from data loss as `journal`, but deleted files could reappear in the file system after a crash; you could say it's the opposite of data loss. In a Maildir system that would not really be a problem: the worst case is that some recently deleted emails reappear.

Table 8.3 demonstrates the differences in speed.

Table 8.3. The `writeback` mode does not make ext4 faster, `ordered` wins by a slight margin

10,000 RPM SATA	ordered	writeback	journal
transactions/sec.	679	667	333
files created/sec.	458	453	248
files deleted/sec.	339	333	166
read accesses/sec.	339	333	166
write accesses/sec.	458	453	248
read throughput MB/sec.	1.45	1.47	0.88
write throughput MB/sec	2.77	2.74	1.5
Duration of test run in min.	16:22	16:35	30:12
CPU consumption "user" in min.	0:07	0:08	0:07
CPU consumption "system" in min.	0:41	0:42	0:53

General opinion holds that `writeback` increases performance by about 10% on `journal`, so it should have emerged from the test as the clear winner. I have to admit that I did not expect `writeback` to do so badly here.

However, according to Ted Tso, the differences measured in the `journal` and `writeback` modes can be explained: in principle, `writeback` is slightly faster during normal operation, but not if the running software frequently calls `fsync()`, by which it forces the system to write the data definitively from the cache to the hard drive.

These frequent sync actions have very different effects. In the `writeback` mode, the user data have to be written to the hard drive. That requires numerous individual accesses and drive head movements, as the data blocks are spread over a wide area. In `journal` mode, the data only have to be written to the journal when `fsync()` is called. The file system can do this quickly and without moving the hard drive head very much, so it takes considerably less time.

Mail servers do call `fsync()` constantly and regularly in order to ensure the security of the emails. Otherwise they would lose all the emails still located in the cache in the event of a crash or a reset during operation.

In this case, the desktop PC and the mail server have different requirements for the file system. So it is fair to say that in an ext3/ext4 system, the `ordered` journal mode that is active by default is not just the safest but also the fastest.

8.4.3. Optimized fstab entries

The mount options in `/etc/fstab` determine whether `atime`/`noatime` is used and which method is used for ext3/ext4.

If you combine all the options suggested here, an entry in `/etc/fstab` for ext3/ext4 will look like this:

```
/dev/sda5 /mail ext4 defaults,noatime,data=ordered 1 2
```

As XFS does not offer different journal modes, the entry in `/etc/fstab` would look like this:

```
/dev/sda5 /mail xfs defaults,noatime 1 2
```

8.5. Out of service thanks to fsck

The volume of emails grows and grows, which is not a problem in itself; after all, adequately sized hard drives in the TB range are available, and storage systems with dozens of hard drives offer almost unlimited options. But there is a hidden problem that does not emerge for a long time: the file system check (`fsck`).

It takes a long time to run a proper `fsck` across large file systems. In partitions with many TB, it can take an hour or more. In this time, your server and therefore your email service are not available, and you can do nothing but wait and apologize to users. The volume of hard drives has increased enormously in recent years – while speed has not risen much in comparison. A regular `fsck` that takes place from time to time can be scheduled as a downtime during the night, but what happens after an uncontrolled reboot in the middle of the day, or after a crash, power outage or other problem?

Journaling file systems use their journal to ensure that the entire hard drive no longer needs to be subjected to a file system check; but situations can nevertheless occur where a full scan is required.

Do you know how long your server is out of service at such times? Have you included this situation in your availability planning? You should think about it – if just so you can accept (and plan) this as a calculated and accepted residual risk. You may decide that dealing with this problem will require more cost and effort than accepting the damage that such a failure can produce. That may be acceptable for a "normal" (personal) mailbox and an outage of a few hours, but it could have serious business consequences if your business is at a standstill, meaning that hundreds or thousands of employees can no longer work and customers can not use your platforms.

Several factors determine how long a `fsck` actually takes:

- the hardware of the hard drive(s), so the connection system (SATA, SAS u. a.) and I/O performance due to the rotation speed (7200 rpm, 10,000 rpm, 15,000 rpm)
- the RAID setup (RAID-1/5/6/10)
- the fill level of the file system (volume)
- the number of files (so it is also very important whether you use Maildir or mdbox as your email storage)
- the age of the file system, i. e. the fragmentation and gaps of the individual blocks
- above all the type of file system, because there are huge differences once again between ext3, ext4, xfs and btrfs

In Table 8.4 I compared the times of a file system check in the "SATA hard drive" and "SAN storage" scenarios. These measurements differ widely, depending on the hardware; they were

also performed on a recently formated file system. Nevertheless, the measurements clearly show how outage times of several hours can develop here and how the file systems behave in comparison.

Table 8.4. Duration of a normal file system check in minutes

	ext3	ext4	XFS	btrfs	ReiserFS
500 GB SATA 10K RPM	1:45	0:02	0:01	<0:01	–
empty					
500 GB SATA 10K RPM	3:22	0:27	0:49	2:43	–
filled with 420 GB					
750 GB SATA 7.2K RPM	15:00	1:12	–	–	–
filled with 300G					
750 GB SATA 7.2K RPM	21:45	1:15	–	–	–
filled with 500G					
3.5 TB SAN 15K RPM Raid-5	32:45	0:07	0:21	0:00	2:54
empty					
3.5 TB SAN 15K RPM Raid-5	48:54	2:15	0:44	12:54	189:08
filled with 3,2 TB					

It is shocking to see how even the completely empty (!) 3.5 TB partition on the basis of ext3 in SAN causes a break of more than 30 minutes before the system is ready to go again – and interesting to see that ext4 or xfs are not really bothered. As for ReiserFS, I hope it has been made clear already that there are many reasons not to use it. Once again, the clear winner is XFS.

Consider this: the full partitions were necessarily tested on a fully formated partition with clean contents. On full file systems that have fragmented over time, the file system check will take correspondingly longer.

The following solutions are possible:

- use ext4 or XFS as the file system – ext3 and ReiserFS are clearly out of the running
- more I/O in the hardware
- smaller partitions, so a file system check can be performed on parts or parallelized
    - A partitioned cluster allows the pool of emails to be distributed among several autarkic smaller files (Section 16.2)
    - If mdbox is used, an *alternate storage* at least ensures that the volume is distributed among two partitions (Section 7.3.4)
    - Object storage effectively resolves the problem of email partitions that are too large and is a dream choice under these aspects (Section 20.1)
- If you use Dovecot with replication in the cluster (Section 16.4), every node has its own autarkic file system and can be removed easily for maintenance purposes for a

comprehensive file system check. That makes the situation far more relaxed.

---

[44] It used to be http://www.netapp.com/tech_library/postmark.html, but there seems to be no more official website anymore.

[45] In general you can mount all partitions in your system with the `noatime` option, even the root partition.

Part II. Advanced Dovecot Installation

Chapter 9. The IMAP namespace and shared folders

Shared folders are an extremely useful way of enabling a team to collaborate on a project. You don't need an elaborate groupware solution or any other tricks – IMAP provides everything you need.

A user can use IMAP ACLs (*access control lists*) to grant another user permission to access one of his folders. He can make detailed decisions on the permissions that the other user should be granted:

`l` (lookup)
: The folder is visible and the user can subscribe to it.

`r` (read)
: The user can select the folder for reading.

`w` (write)
: The user can save/change message flags and keywords except for `\Seen` and `\Deleted`.

`s` (write-seen)
: The `\Seen` flag can be set/modified.

`t` (write-deleted)
: The `\Deleted` flag can be set/modified.

`i` (insert)
: Messages can be written or copied to this folder.

`p` (post)
: Messages can be delivered to this folder via `dovecot-lda` or LMTP, for example as the result of Sieve filtering.

`e` (expunge)
: Messages can be marked for deletion in this folder.

`k` (create)
: Under this folder, the user can create new folders and rename existing ones. Renaming is only possible with delete permissions.

`x` (delete)
: The user can delete this folder.

`a` (admin)
: The user has administration permissions for this folder (and is therefore allowed to set ACLs).

The great thing is that the user can set ACLs in his email client without assistance from the administrator. Older email clients do not support ACLs, and Thunderbird still requires installation of the `Imap-ACL-Extension` from menu item *Extras+Add-ons*. In principle, however, modern email clients can handle ACLs for reading and writing operations. With an

`acl plugin`, current versions of the Roundcube webmailer also support the reading and setting of ACLs in the IMAP plugin.

ACL settings are not always easy to find – in Thunderbird, for example, you can find them by right-clicking on the folder in question and then choosing *Properties*+*Sharing*+*Privileges*.

Please note that the user should specify the login name of the user who is to be granted access to the directories. In setups where login name and email address are identical, this is simple and easy to do: you provide access to a different email address, and that's all there is to it.

If, however, short names are used as login names, the person granting access must first ask the recipient for his login name in order to enter the ACL rule correctly. I have encountered quite a few data protection issues at universities and colleges where the login name also contained the student's matriculation number; of course, that is a fundamental problem anyway. Exam results are often posted "anonymously" on notice boards under students' matriculation numbers.

All in all, shared folders are another reason why I think users should log in with their email address. Unfortunately that is not always possible, and in some setups it does not even make sense.

9.1. Necessary preparation

Shared folders work in Dovecot in the following way:

1. If a user logs in and the `acl` plugin is active, Dovecot consults a *dictionary* to determine whether the other users have granted access to this user.
2. The IMAP process of the user who is logged in uses a `userdb` query to determine the home directories of the other users.
3. Then Dovecot looks at the `dovecot-acl-list` file of these users, which lists the folders for which ACL permissions have been set.
4. In the last step, Dovecot evaluates the `dovecot-acl` file in the respective folders; this file actually contains the list of ACLs for the folder in question.

There is a simple reason for this rather complicated method. If you have a system where many tens of thousands of users each own thousands of folders, Dovecot is unable to search all folders for existing ACLs when a user logs in. It has to use central content directories to determine quickly which folders are even a possible option, i.e. where an ACL permission for the user who is logged in may be hidden.

When a user revokes the permissions he has granted to another user, the original entry in the central directory remains. Dovecot does not tidy up there, even if that means it has to search the home directory of the user who originally shared the folder. That is (slightly) inefficient but not really a problem, because a hundred email directories do not really make a difference during login. The important thing is that this mechanism excludes hundreds of thousands of other directories!

First you need to prepare Dovecot for the central dictionary `/var/lib/dovecot/db/shared-mailboxes.db`. The (unprivileged) IMAP process of the logged-in user must also be able to read and write to it. As a temporary file has to be set up when this file is changed, you have to modify the file permissions of the whole `/var/lib/dovecot/db` folder. If, as recommended in this book, you use the cental uid `vmail` for all users, the procedure is simple:

```
flash:~ # mkdir /var/lib/dovecot/db
flash:~ # chown vmail:vmail /var/lib/dovecot/db
```

During operation, make sure that this file is kept in sync on all cluster nodes.[46]

You need to set up another `auth` socket with unprivileged access permissions so that the IMAP process of the logged-in user can use a `userdb` query to determine other home directories. Modify the existing `auth-userdb` socket in `10-master.conf` as follows:

```
service auth {
```

```
  unix_listener auth-userdb {
    #mode=0660
    user = vmail
    group = vmail
  }
}
```

Now you need to make sure that the `acl` plugin is loaded for all modules. Add the `acl` plugin to `mail_plugins` in the `10-mail.conf` file:

```
mail_plugins = [...] acl
```

Then add the `imap_acl` plugin to the IMAP module in the `20-imap.conf` file. After all, you want the user to be able to manage his access permissions via the IMAP protocol:

```
protocol imap {
  mail_plugins = $mail_plugins [...] imap_acl
}
```

Finally, tell the `acl` plugin in the `90-acl.conf` file where to create the dictionary:

```
plugin {
   acl = vfile
   acl_shared_dict = file:/var/lib/dovecot/db/shared-mailboxes.db
}
```

Later on there will be entries according to the following pattern:

```
shared/shared-boxes/user/peer@example.com/ivonne@example.com
```

The `shared-mailboxes` file should be read from right to left: Ivonne has granted Peer access. The remaining entry, `shared/shared-boxes/user/`, is always the same and does not have any deeper meaning at the moment.

If the `acl` plugin is active, `doveadm` in version 2.2 or higher is also familiar with corresponding `acl` commands (see Section 9.6.1).

9.2. Definition of a shared namespace

Now you can set up a special IMAP namespace under which the shared folders belonging to other users are displayed. In addition to the namespace for the INBOX, the `/etc/dovecot/10-mail.conf` file contains a `shared` type namespace that you have to activate and modify as shown here:

```
namespace inbox {
  type = private
  hidden = no
  ignore_on_failure = no
  inbox = yes
  list = yes
  location =
  prefix =
  separator = .
  subscriptions = yes
}

namespace {
  type = shared
  hidden = no
  ignore_on_failure = no
  inbox = no
  list = children
  location = maildir:%%h/Maildir:INDEX=%h/shared/%%u:CONTROL=%h/shared/%%u
  prefix = shared/%%u/
  separator = .
  subscriptions = yes
}
```

In Dovecot, `%h` refers to a user's home directory and `%u` refers to his user name, while double percentage signs in a shared namespace always refer to the data of the other user, i.e. the one sharing the folder. `%%h/Maildir` therefore refers to the Maildir directory containing the shared email data.

If user Peer grants user Ivonne access to his folder `INBOX.Personal.Vacation`, the folder would be displayed to Ivonne in this configuration as

```
shared
 +-- peer
     +-- INBOX
         +-- Personal
             +-- Vacation
```

and access to this folder would be redirected to Peer's home directory. However, the index files and the control files (such as the `subscriptions` file) remain with user Peer, stored in a directory named `shared/<otherusername>`.

Once you have completed all these steps and reloaded Dovecot after completing the changes to the configuration, your users can grant access to their IMAP folders. But there are still a few traps to remember and a few decisions to make.

9.3. The right hierarchy separator for a shared namespace

In practice, the configuration shown above will cause problems if the users' login names contain a point, as in `p.heinlein@example.com`. As the point is also used as a hierarchy separator, the resulting hierarchy for the email client would be

```
shared
 +-- p
     +-- heinlein@example
         +-- com
             +-- INBOX
                 +-- Personal
                     +-- Vacation
```

If you cannot ensure that user names will never contain a point, I strongly advise you to change the hierarchy separator from a point to "`/`".

`INBOX.Personal.Vacation` becomes `INBOX/Personal/Vacation`, and the email client is able to display a shared-folder release `shared/p.heinlein@example.com/INBOX/Personal/Vacation` with the right hierarchies. [47]

If you are just setting up your email system, it is easy to change the hierarchy separator. Email clients will recognize that you have done so and adapt to that fact from the start. The IMAP protocol is set up so that an IMAP server can choose its own hierarchy separator (and the client can decide which separator it will use in its final display). In our example, you would specify `separator = /` in both namespaces (!):

```
namespace inbox {
  type = private
  hidden = no
  ignore_on_failure = no
  inbox = yes
  list = yes
  location =
  prefix =
  separator = /
  subscriptions = yes
}

namespace {
  type = shared
  hidden = no
  ignore_on_failure = no
  inbox = no
  list = children
```

```
  location = maildir:%%h/Maildir:INDEX=%h/shared/%%u:CONTROL=%h/shared/%%u
  prefix = shared/%%u/
  separator = /
  subscriptions = yes
}
```

If you make this change on a system that is already running and contains relevant email data, make sure you observe the following points:

- In the `subscription` files, where Dovecot lists the folders the user has subscribed to, the folder names can still contain a point. Dovecot can understand and translate that.
- However, you need to adapt the folder names in Sieve filtering rules. `fileinto INBOX.Mailinglists.Dovecot` now becomes `fileinto INBOX/Mailinglists/Dovecot`. Don't replace all points in Sieve scripts, otherwise you may destroy email addresses and autoresponder texts. Make sure that you only replace points in lines that also contain the `fileinto` command.

The following `sed` command performs this task, but it outputs the modified files only on the screen:

```
flash:~ # sed '/fileinto/   s/\./\//g'  sievescript.txt
```

Check the output of `sed` and then have the actual changes performed in the files via parameter `-i´.

```
flash:~ # sed -i '/fileinto/   s/\./\//g'  sievescript.txt
```

9.4. Shared folders in mdbox or the auto:-mode

But be careful. Unlike in the Maildir format, the `dovecot.index.*` index files are essential with mdbox so that Dovecot can identify the position of the emails in the `m` files again (see Section 7.3).

Unlike Maildir, the `mdbox` format does not allow `INDEX` and `CONTROL` to be twisted towards a directory of the share recipient in the `shared` namespace.

So the option below would be *wrong* for mdbox:

```
namespace {
  type = shared
  prefix = shared/%%u/
  location = mdbox:%%h/mdbox:INDEX=%h/shared/%%u:CONTROL=%h/shared/%%u
}
```

The right version for mdbox is:

```
namespace {
  type = shared
  prefix = shared/%%u/
  location = mdbox:%%h/mdbox
}
```

Dovecot can then use the share giver's `dovecot.cache.*` and find the emails 's mdbox directory.

A similar principle applies if you allow Dovecot to determine the type of the target directory automatically, so if you are using the `auto:` mode (see Section 7.5):

```
mail_location=auto:
```

There can be no parameter after `auto:`. You therefore cannot use `auto:%%h` to specify the remote home directory. Even shared namespaces require nothing other than `auto` for the auto mode:

```
namespace {
  type = shared
  separator = /
  prefix = shared/%%u/
  location = auto:
  subscriptions = yes
  list = children
}
```

9.5. Folders parallel to the INBOX

If you have not done so before, you should think carefully about the structure of the `inbox` namespace when you introduce shared folders. The IMAP protocol itself does not specify whether IMAP folders may exist in parallel to the `INBOX` or have to be underneath it in the hierarchy. This subject was discussed in Section 4.4.1, so you may want to read up on it again there.

Considering the two points made there, I would always advise you to allow folders only underneath the `INBOX`. Particularly if you are building a new system from scratch, you should set it up to be future-proof: there is no real reason not to, and you do not need to perform any migrations later on.

Set the suitable prefix in `namespace inbox`. Remember that the prefix must end with the hierarchy separator. Depending on the setup, the entry should be `INBOX.` or `INBOX/` – and not just `INBOX`:

```
namespace inbox {
  type = private
  hidden = no
  ignore_on_failure = no
  inbox = yes
  list = yes
  location =
  prefix = INBOX/
  separator = /
  subscriptions = yes
}
```

If you decide to adapt the namespace on a system that has grown over time, make sure you observe the instructions in Chapter 19.

9.6. Public folders

While *shared folders* involve individual users granting one another access, *public folders* are not tied to a single user as the share giver, but are instead set up and shared centrally by an administrator at file level.

At first they are accessible to all users in the system, almost like a notice board, but the administrator can also use ACLs to determine which users have access to which public folders, and what kind of permissions they have for these folders.

In addition to the namespaces described earlier, `inbox` and `shared`, you can enter the `public` namespace in `10-mail.conf` (`public` is currently not yet prepared in the configuration files):

```
namespace {
  type = public
  separator = /
  prefix = Public/
  location = maildir:/srv/vmail/public
  subscriptions = yes
}
```

Make sure that `separator` and `prefix` match the rest of your namespace. If you are operating Dovecot in an active/active replication, please also observe Section 16.4.2.

Now create the directory for your public namespace. Make sure that everything is accessible to user `vmail` and this path does not accidentally cross that of another user.

```
flash:/srv/vmail # mkdir public
flash:/srv/vmail # cd public
flash:/srv/vmail/public # mkdir cur new tmp
flash:/srv/vmail/public # mkdir ".Newsletters"
flash:/srv/vmail/public # mkdir ".Menu for canteen"
flash:/srv/vmail/public # chown vmail:vmail .*
flash:/srv/vmail/public # ls -la
total 16
drwxr-xr-x 4 vmail vmail 4096 Aug  2 21:16 .
drwxr-xr-x 4 vmail vmail 4096 Aug  2 21:15 ..
drwxr-xr-x 2 vmail vmail 4096 Aug  2 21:16 .Menu for canteen
drwxr-xr-x 2 vmail vmail 4096 Aug  2 21:15 .Newsletters
drwxr-xr-x 2 vmail vmail 4096 Aug  2 21:16 cur
drwxr-xr-x 2 vmail vmail 4096 Aug  2 21:16 new
drwxr-xr-x 2 vmail vmail 4096 Aug  2 21:16 tmp
```

All users now have read, write and delete permissions for this public folder. That may not be quite what you want.

You can use standard ACLs to manage restrictions, so your next step is to manually modify the ACL permissions in the `dovecot-acl` file in the Maildir. Use a special keyword,

`authenticated`, to grant read permissions to all users, and then grant some privileged users write permissions for the public folders so they can publish messages there, for example by using drag & drop.

```
flash:/srv/vmail/public # vi dovecot-acl
authenticated lrs
user=teamassistant@example.com akxeilprwts

flash:/srv/vmail/public # chown vmail:vmail dovecot-acl
```

Remember to modify the ACLs for subfolders accordingly as well. You can simply copy the `dovecot-acl` file to do so.

9.6.1. Managing the public namespace with a dummy user

In certain circumstances, you can also deliberately have the public namespace displayed in the path of a dummy user reserved for that purpose. That allows you as the administrator to integrate this special mailbox in your email client and design the content of the public namespace. You can also use the email client or `doveadm` commands to manage the ACLs of the folder in order to control who has read access to these IMAP folders and who has write access as well.

Only use this trick if you have *no* other shared namespace. Otherwise your users would see your public namespace twice, once as a share by the dummy user and once as a normal public namespace, due to the ACL shares you have set.[48]

To use this version, set the `location` of the public namespace to the Maildir of the dummy user:

```
namespace {
  type = public
  separator = /
  prefix = Public/
  location = maildir:/srv/vmail/example.com/publicuser/Maildir
  subscriptions = yes
}
```

Access the mailbox in question via your email client or webmailer and use it to assign the required ACL permissions easily and conveniently.

By default, Dovecot does not allow ACL permissions to be set for `anyone` (all users, even anonymous ones) or `authenticated` (all authenticated users) via the IMAP protocol. There is a great risk that users will accidentally grant far-reaching access.

However, you can activate the keywords `anyone` and `authenticated` in `90-plugin.conf` in the following way if you want to manage permissions via your email client:

```
plugin {
  acl_anyone = allow
}
```

From Dovecot 2.2.x onwards, you can also set ACL permissions in the console using the Dovecot command `doveadm acl`:

```
flash:~ # doveadm acl set -u publicuser@example.com INBOX owner all
flash:~ # doveadm acl set -u publicuser@example.com INBOX authenticated lo
flash:~ # doveadm acl get -u publicuser@example.com INBOX
ID                  Global Rights
authenticated       lookup read write-seen
owner               admin create delete expunge insert lookup post read w
```

You can then also create folders conveniently by means of the `doveadm` command:

```
flash:~ # doveadm mailbox create -u publicuser@example.com "INBOX/Newslett
flash:~ # doveadm mailbox create -u publicuser@example.com "INBOX/Menu for
```

Hint: before you start creating folders, you should first sort out the ACLs in the public namespace. Newly created folders will then “inherit” the right ACLs, and you avoid having to repeat various tasks.

9.6.2. Display in the folder listing

You can use your own email client to check whether the public folders are available to users, but you can also use the `doveadm` command to view the folders for each individual user:

```
flash:~ # doveadm mailbox list -u klaus@example.com
INBOX
INBOX/Vacation
Public
Public/Menu for canteen
Public/Newsletters
```

9.7. User-specific \Seen flags in shared folders and in the public namespace

In the Maildir format, the `\Seen` flag is coded in the file name and is therefore the same for everyone in folders used by multiple users. In the mdbox format, the flag is saved in the index file rather than the file name, but it is still the case that all users access the same index file and so will see the same `\Seen` flags.

From version 2.2, Dovecot allows you to save the `\Seen` flag in a separate index file (in contrast to the normal process), which can be located in the home directory of every user and separate from the actual email files. That way, the `\Seen` flag is saved individually for every user.

You have to decide on a case-by-case basis whether individual `\Seen` flags are useful or inconvenient. In a team mailbox or for holiday cover, it is very useful to know what the other colleagues have already read, so individual `\Seen` flags are more of a drawback. If, however, you use public or shared folders as a team distribution list to provide central information to all users, individual `\Seen` flags for each user are a good idea.

Whether you use Maildir or mdbox, you need to add the `INDEXPVT` parameter to the `location` entry in the shared or public namespace for the individual `\Seen` flags, because that parameter refers to the user's local home directory.

For a shared namespace:

```
namespace {
  type = shared

# Maildir:
  location = maildir:%%h/Maildir:INDEX=%h/shared/%%u:CONTROL=%h/shared/%%u
# For mdbox:
# location = mdbox:%%h/mdbox:INDEXPVT=%h/mdbox/shared

[...]
}
```

And for the public namespace:

```
namespace {
  type = public

# Maildir:
  location = maildir:/srv/vmail/public:INDEXPVT=%h/Maildir/public

# For mdbox:
# location = mdbox:%%h/mdbox:INDEXPVT=%h/mdbox/shared
```

```
[...]

}
```

---

[46] http://wiki2.dovecot.org/SharedMailboxes/Shared shows how to provide the dictionary on an SQL server.

[47] There is a plugin called `listescape`, which can escape dots that are not supposed to be hierarchy separators in such a way that the client shows them as a dot and does not use them as a hierarchy separator: http://wiki.dovecot.org/Plugins/Listescape. However, I still encountered some strange problems with various email clients, so I do not consider this a feasible option.

[48] A crafty method would be to manually delete the necessary entry from the shared dictionary in `/var/lib/dovecot/db/shared-mailboxes.db` and thereby prevent the share again.

Chapter 10. Setting up SSL and TLS

SSL/TLS-encrypted connections are old hat; after all, the TCP/IP connections of a variety of protocols have been transmitted via an encrypted tunnel using this procedure for about 20 years. Next to `https`, email sending and mailbox access for SMTP, POP3 or IMAP servers are among the most popular applications.

There has been a lot of discussion in recent years about the quality and safety of the SSL/TLS system. It is based on the idea that some *Certification Authorities* (CA) can sign keys and are trustworthy enough for everyone to believe them. If a single CA signs keys for unauthorized parties, the entire system fails, because attackers can now use trustworthy keys to break into connections.

SSL/TLS is certainly not the Fort Knox of data security, but it does provide solid everyday encryption – particularly for mass use. The strength of SSL/TLS is that it can transport any encrypted TCP/IP-based connection and therefore faces potential attackers or spies with the needle-in-a-haystack problem: thanks to the sheer mass of encrypted connections, it is impossible to tell which connections are worth concentrating on.[49]

Edward Snowden's revelations about the extensive NSA spying programs[50] have (finally) made people realize the shocking extent to which data traffic on the internet is read, recorded and stored by others. In technical terms, it is obviously easy for secret service organizations (and others) to gain access to data traffic. So mass is not a problem. What is a problem for eavesdroppers is if this data traffic is protected by an overlying encryption. This can be broken in individual cases – but not in the sum of all connections. As a result, the eavesdroppers are blinded.

Edward Snowden provided this advice in his interviews: "Strong encryption is the only thing you can rely on."[51]

SSL/TLS is easy to set up and offers great benefits. Even more surprising, then, that of 15 German providers tested in July 2013, only Freenet, mail.de and Arcor offered complete and secure SSL/TLS encryption; Googlemail, emailn.de, 1und1.de and kabelmail.de did advertise SSL/TLS , but unfortunately their certificates did not work. Yahoo, AOL, web.de, gmx, hotmail.com, outlook.com, facebookmail, me.com and Strato do not advertise an SSL/TLS option on their MX servers.[52] From my experience as a consultant, I know that German providers are often in this situation because some sellers of technology for recording emails demand an unencrypted SMTP data stream in line with the German telecommunications monitoring act (TKÜV); if the providers prefer not to replace their chosen solution, they simply go without SSL/TLS.

Snowden's revelations have also shown that the NSA can eavesdrop on TSL encryptions in a major way – there is a suspicion that the ancient RC4 algorithm, which is used on many websites, no longer offers any security at all. So using current versions of `openssl` and setting up perfect forward secrecy will considerably improve security.[53]

10.1. How SSL/TLS works

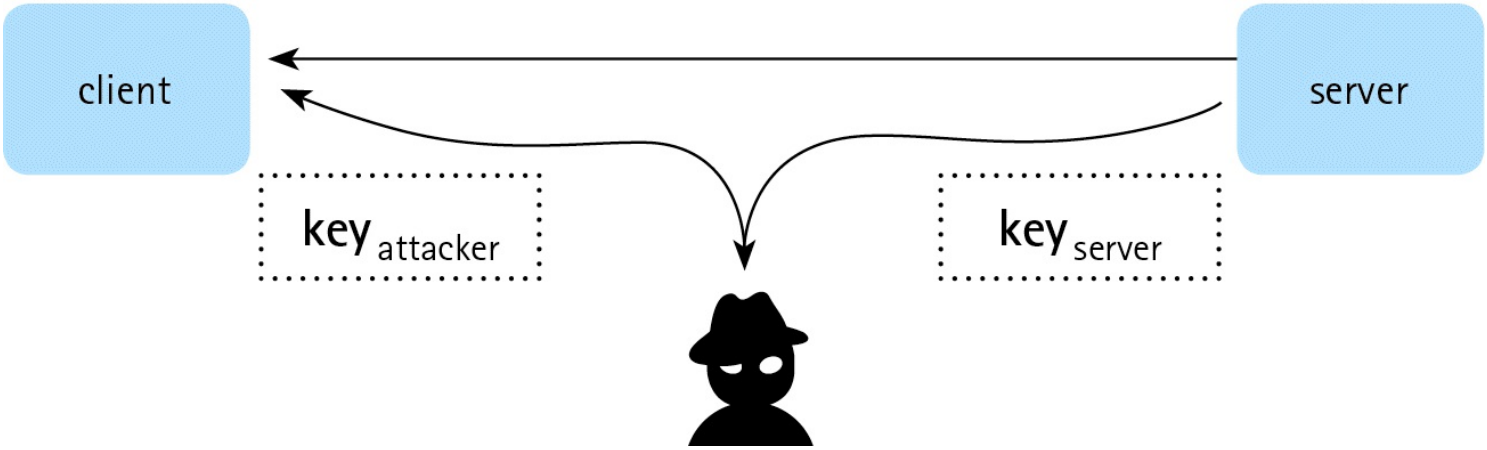
SSL/TLS is based on an asymmetric pair of keys and a principle familiar from PGP or S/MIME. There are two key parts:

Public key
: The public key allows you to encrypt things but not decrypt them again. As the name indicates, it is neither secret nor worth protecting; everyone can and should know it, and that is why the server sends the public key to the client upon request.

Private key (or *secret key*)
: It allows data to be decrypted that were previously encrypted using the public key. It *must* remain secret and may not fall into the hands of an attacker.

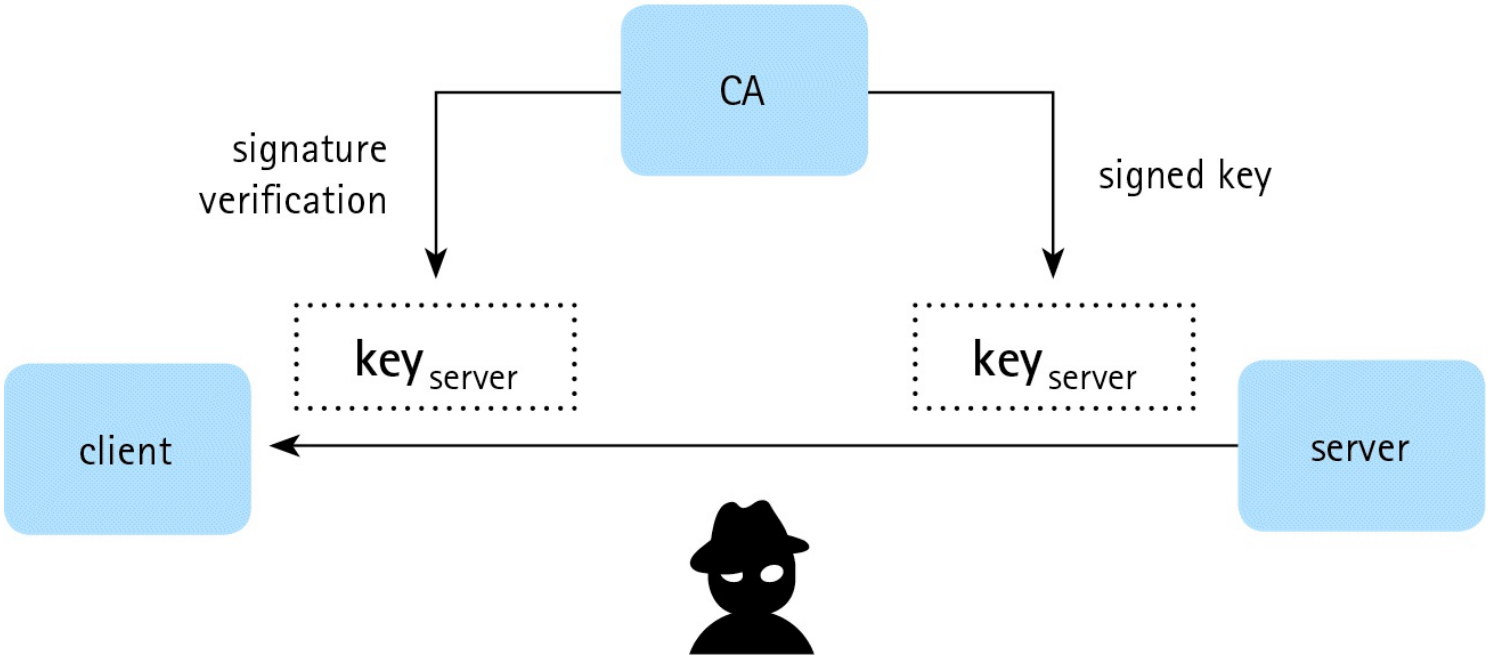
If a man in the middle exchanges the certificate, he can conduct all subsequent communications via himself as a relay, encrypt/decrypt everything for both sides and read all the data (see Figure 10.1).

Figure 10.1. If a man in the middle replaces the key, he can read the subsequent encryption



In order to prevent this, the public key is usually signed by a (commercial or self-built) *Certification Authority* (CA) (see Figure 10.2). For this reason, the term in SSL/TLS is not public key but *certificate*, which is nothing other than a signed public key.

Figure 10.2. The signature of a well-known CA prevents the key from being exchanged secretly



A functioning SSL/TLS encryption based only on asymmetric keys would require a key pair on the client as well as a key pair on the server, because that is the only way the server would be able to send the client encrypted data. As a rule, clients do not have their own key pair, and they certainly do not feature a CA-signed key.

SSL/TLS is actually a little more sophisticated:

1. The client requests the public key from the server and checks the authenticity of the key by means of the accompanying signature (*certificate*) from a hopefully familiar CA.
2. The client then generates a symmetric (!) key for the subsequent data transmission. This key should not be overheard by any man in the middle, so the client encrypts the symmetric key using the server's public key and thereby sends the key to the server in a secure manner.
3. The server can unzip the contents of the data package and so safely obtain the symmetric key.
4. Now the client and the server can switch to purely symmetric encryption and subsequently communicate in encrypted fashion.

Once the connection has been built up, the asymmetric keys are no longer used; this is particularly good for the server because symmetric encryption requires far less computing than asymmetric encryption.

With this knowledge, you should now realize that a public key signed by a "proper" CA does not provide better encryption, but simply prevents the key being swapped by a man in the middle. You can also sign your keys yourself --the quality of the encryption will not be affected.

When first connecting to a server with a self-signed key, the client is warned about the lack of authentication, but can accept the key regardless – now there is a danger that a man in the middle has already interposed himself.

If you assume, however, that the connection is (still) secure at that time, subsequent communications are also trustworthy. If a man in the middle does undercut the key exchange at a later stage and introduces his own key, the client will see a corresonding warning again, whether the original key was signed by a CA or self-signed. The main thing is to teach users not to simply agree to *later* key warnings.

You have to decide for yourself whether to use self-signed keys or obtain a certificate from a CA (which is no longer particularly expensive). It does not affect the set-up of SSL/TLS, and you can replace the key files at any time.

10.2. How to generate a self-signed key

A script named `CA.pl` is usually used to generate the SSL/TLS key; it is provided as part of the distributions and performs the matching `openssl` calls.

Various instructions describe how to use `CA.pl` to create your own key pair (`CA.pl -newreq`), create your own CA (`CA.pl -newca`) and then sign your own key with your own CA (`CA.pl -sign`). This method works, and is both correct and feasible, but there is a much simpler method involving just one step.

With the following `openssl` call you generate a self-signed key pair in one single process that you can immediately use on all your servers – even for Apache web servers or other services. Remember that your key is top secret and use `chmod` to make the file permissions correspondingly secure:

```
flash:/etc/ssl/private # openssl req -new -newkey rsa:2048 -days 3650 \
   -nodes -x509 -subj /CN=mail.example.com  \
   -keyout /etc/ssl/private/mail.example.com.key \
   -out /etc/ssl/certs/mail.example.com.crt
flash:/etc/ssl/private # chmod 400 mail.example.com.key
```

When generating your keys, always make sure that you issue you them for precisely the host name that your users have entered in the client software as the server to be used. Replace `mail.example.com` in this example with the host name that you have supplied to your users. Professionally signed keys will not do you any good if the key and host names are not exactly the same.

A proliferation of host names will come back to haunt you

This is another reason to make sure that your machines are not available in parallel under different names. Make sure that your sever is available *either* under mail.example.com *or* under imap.example.com. It is easy for your users to accumulate different host names over the years; if you want to (or have to) standardize everything, you will get a nasty surprise (and require a lot of support work).

Here is another piece of advice: always name the files of the SSL keys exactly for the host name in the key, e. g. `mail.example.com.key`. That is the only way to keep order if you have 20, 200 or even 2,000 keys, and the only way to manage and generate them automatically.

Once you have generated the keys and set the file permissions securely, you can make modifications in `10-ssl.conf`. `ssl=yes` switches SSL/TLS on as an option, `ssl=required` makes SSL/TLS mandatory for the clients and prevents plain text connections from the users.

Of course you also have to provide Dovecot with the correct paths to the key and cert files:

```
# Possible values: yes | no | required
ssl = yes

ssl_cert = </etc/ssl/certs/mail.example.com.crt
ssl_key = </etc/ssl/private/mail.example.com.key
```

By the way, the pointed brackets in front of the file names of the keys are not a typo; they actually have to be here. In this case, Dovecot interprets the pointed brackets as an include and includes the contents of the key files as a value in the parameters when parsing the config file. Dovecot modules `pop3d` and `imapd` initially read the config file with `root` permissions (which is how they obtain the keys) and can then downgrade themselves to become unprivileged processes without read permissions for the key files.

If your CA uses intermediate certificates (in the German Research Network, or DFN, for example), you must enter all certificates in the `cert` file in the order of the certificate chain. Start with the certificate for your server, and then add the next-higher intermediate certificate. End with the "highest" certificate from the CA.

Once everything has been entered correctly, Dovecot has to announce the `STARTTLS` command after a restart. A manual SSL/TLS connection via `telnet` is most likely beyond the mental arithmetic abilities of most readers of this book. But if you give your server the `STARTTLS` command and the server then begins to initiate TLS, you can very easily test whether everything is set up correctly for Dovecot and all the keys are available. If it doesn't work, you will find information on the errors in the Dovecot log file:

```
flash:~ # telnet localhost 143
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
* OK [CAPABILITY IMAP4rev1 LITERAL+ SASL-IR LOGIN-REFERRALS ID ENABLE IDLE
a STARTTLS
a OK Begin TLS negotiation now.
^C
Connection closed by foreign host.
```

If you like, you can use the `openssl` command to create a working SSL/TLS-secured connection and use it as you would with `telnet`:

```
flash:~ # openssl s_client -starttls pop3 -connect mail.sample.com:110
flash:~ # openssl s_client -starttls imap -connect mail.sample.com:143
```

10.3. Different keys for different IPs

If you have to make your Dovecot server available under different names via SSL/TLS, maybe because different customers know you by different names or you have to allow for historic profileration, there are three procedures:

10.3.1. Different keys for different services

In the previous example, the keys in `10-ssl.conf` were generically defined. But you can also provide different definitions for parameters `ssl_cert` and `ssl_key` within a `protocol` section:

```
protocol imap {
    ssl_cert = </etc/ssl/certs/imap.example.com.crt
    ssl_key  = </etc/ssl/private/imap.example.com.key
}

protocol pop3 {
    ssl_cert = </etc/ssl/certs/pop.example.com.crt
    ssl_key  = </etc/ssl/private/pop.example.com.key
}
```

10.3.2. Different keys on different IPs

If your server runs multiple IPs with different names, you can use them to provide different keys and certificates:

```
# Customer 1
local 192.0.2.10 {
    ssl_cert = </etc/ssl/certs/imap.example.com.crt
    ssl_key  = </etc/ssl/private/imap.example.com.key
}

# Customer 2
local 192.0.2.20 {
    ssl_cert = </etc/ssl/certs/mail.example.net.crt
    ssl_key  = </etc/ssl/private/mail.example.net.key
}
```

And of course combinations are possible, so you can provide different keys for different protocols on different IPs:

```
local 192.0.2.10 {
  protocol imap {
    ssl_cert = </etc/ssl/certs/imap.example.com.crt
    ssl_key  = </etc/ssl/private/imap.example.com.key
  }

  protocol pop3 {
    ssl_cert = </etc/ssl/certs/pop.example.com.crt
    ssl_key  = </etc/ssl/private/pop.example.com.key
  }
}

local 192.0.2.20 {
  [...]
}
```

10.3.3. Server Name Indication (SNI)

If a server is available under more than one name, it is unable to know at first which name the client used to find it in the DNS – and which SSL certificate the server needs to present to the client so everything fits together. For this reason, it has always been common to equip web and mail servers with multiple IPs that could clearly separate host names and certificates.

Server Name Indication provides a solution: this protocol extension causes the client to transmit the host name it considers correct to the server when creating the TLS connection – and the server can now present the client with the matching certificate from its fund.

That allows you to provide individual certificates for different customers under their own host names on a server with one single IP address.

If you configure the necessary keys in a separate `local_name` section, Dovecot will react to the names transmitted by the clients:

```
local_name imap.example.org {
  ssl_cert = </etc/ssl/certs/imap.example.org.crt
  ssl_key = </etc/ssl/private/imap.example.org.key
}
local_name mail.example.net {
  ssl_cert = </etc/ssl/certs/mail.example.net.crt
  ssl_key = </etc/ssl/private/mail.example.net.key
}
```

Of course the clients also need to support SNI, which is increasingly the case in versions released in recent years. Older email clients without SNI will otherwise receive certificates that do not match the requested host names, which in turn will generate a corresponding warning for the client. So you have to decide whether you can save yourself some work and IP addresses with the aid of SNI, or whether you should provide a separate IP address for each host name and certificate, just to be on the safe side.

10.4. SSL/TLS and authentication

Once you have set up SSL/TLS on your servers, you should think about making encrypted connections mandatory for your users. The advantages and increased safety are clear.

You can achieve that by entering `ssl=required` in opsfile:10-ssl.conf[] as shown above. By forcing your users to use SSL/TLS-secured connections in this way, you protect their passwords as well as their transmitted emails from unauthorized lurkers.

But even if you have to permit unencrypted connections for some reason, you should at least attempt to prevent the unsecured transmission of passwords in plain text. You will find the `disable_plaintext_auth` parameter for that purpose in `10-auth.conf`:

```
disable_plaintext_auth = yes
```

The `PLAIN` and `LOGIN` procedures are then permitted only for connections via SSL/TLS, but also for local connections. So don't be surprised if `PLAIN` and `LOGIN` are still offered to you as the administrator in the context of a `localhost` connect.

However, there is no rule without an exception: individual IP addresses or network areas that are listed in `login_trusted_networks` are considered secure even without a SSL/TLS connection. Plain text login continues to be permitted for connections from these IP addresses. That allows you to create exceptions for special clients such as webmailers, internal monitoring or Dovecot proxy servers.

```
login_trusted_networks = 192.168.50.22  192.168.1.0/24
```

---

[49] It is easy to open a sealed envelope, read the contents of the letter and reseal the envelope without leaving any damage. But it is undeniably an effective way to prevent, for example, the mailman from systematically reading the letters he delivers.

[50] http://www.nsa.gov

[51] Snowden's actual words in the Guardian live chat were: "Encryption works. Properly implemented strong crypto systems are one of the few things that you can rely on. Unfortunately, endpoint security is so terrifically weak that NSA can frequently find ways around it." The whole situation changes when you consider how far manufacturers of operating systems are cooperating with the secret service organizations and providing interfaces to the unencrypted data traffic on the user's desktop. http://www.theguardian.com/world/2013/jun/17/edward-snowden-nsa-files-whistleblower

[52] As of July 2013, see http://www.golem.de/news/e-mail-provider-luecken-in-der-verschluesselungskette-1307-100429.html

[53] https://bettercrypto.org

Chapter 11. Quotas

It is hardly worth mentioning, and every administrator knows: users are data hoarders – and users are creative in the way they handle emails. Users are often ingenious in the way they set up their mailbox via IMAP to manage their personal files and data. They copy local files to themselves, sort them, forward them, and, above all, keep them.

User mailboxes grow, and, as hard drive space appears cheap and freely available to users, users do not even feel guilty if their mailboxes are several GB in size. Just recently, an employee of one of my customers was having a spirited argument with the admin team about whether it was necessary and appropriate to expect her to tidy up her 60 GB `INBOX`.

Users like to forget they are not alone in the world, and 10,000 users with a mailbox of 10 GB each amounts to about 100 terabytes. This is expensive as well as being difficult to handle.

Quotas are the technicians' answer to this problem; they simply stop the recipient from receiving new emails if the inbox has reached an unacceptable size.

For me, quotas are a two-edged sword: they are necessary as soon as you have more than just a few power users. But quotas also affect the wrong people first, i. e. the innocent sender to whom the laboriously typed email is returned just because the recipient is unable to curb his thirst for data. In the worst case, the recipient does not even notice his quota block and continues to send emails (that generate responses) and is simply surprised that he receives so few answers.

It is hardly possible to do without quotas in free email systems for private users, and most providers won’t really care that a quota block also affects the recipient. You can’t simply let emails proliferate – just think of the old unused accounts that continue to gather spam and newsletters.

From a business perspective, you could question the use of quotas – after all, the sender is often a customer or business partner, and it does not leave a great impression if orders get stuck behind a quota block.

Tough quota rules on a mail server with plenty of space would even be described provocatively as a preemptive denial of service: though I could technically receive further emails, I will deactivate mailboxes that are in active use – even though availability is supposedly so important. It may be better in such cases to make quotas stricter only once the available space is reaching its limits.

Bear that in mind when setting up quotas in your system.

11.1. Structure of a quota system

A functioning quota system under Dovecot consists of the following components:

Quota roots
: areas that are combined for a shared quota calculation. As a rule, a user will have only one quota root: his mailbox with all sub-folders. You could also set up a quota root for all users in a domain and thereby control a customer's consumption across all his mailboxes.

Quota backends
: various procedures that Dovecot uses to record its current utilization. Every quota root is assigned a quota backend that Dovecot should use for that purpose.

Quota rules
: define in more detail which rules apply in which directories.

Quota warnings
: warn users when they cross specific thresholds.

11.2. Quota backends

First, you need to detemine where and how Dovecot should keep an account of the generated volume. Four quota backends are available at the moment, but only `maildir` and `dict` are relevant:

`dirsize` – a look at the file system
: sums up the volume of all files in the file system. As this happens frequently, and as problematic mailboxes are typically large and contain many files, this method is inefficient and not advisable, despite its simplicity.

`fs` – use of the file system quotas
: If your email users are really still shell users and are assigned individual user IDs in `/etc/passwd`, you can have the file system quotas of the Linux system evaluated by Dovecot. That is only really interesting if your users log on to the mail server with a real shell, or if they have stored other data in their home directory that you want to include in the quota calculation. If that is not the case, I recommend you use other quota backends.

`maildir` – Maildir++ quota
: logs in the incoming and outgoing quota volume in a file named `maildirsize` in the user's home directory. From time to time, Dovecot consolidates the file automatically, adds up all existing journal orders and empties the file again. As there is no need to recalculate everything every time an email is received or deleted, the log is comparatively efficient to keep. But be careful. This quota backend only works if storage is in Maildir, not for `mdbox`!

`dict` – quotas in a database (e. g. MySQL)
: this quota backend allows storage in an SQL database, which is probably rather helpful for automatic evaluation at a different point (web GUI). I personally prefer to have the dictionary saved in a simple file in the user's home directory. The data are then always kept together and consistent, and you are not dependent on a database. Basically this is no different from the Maildir quotas in the `maildirsize` file – just that the file dictionaries also work for mailboxes that store their emails in `mdbox`.

11.3. Activating the quota plugin

Now it's time to activate the quota plugins in Dovecot. Add the following to the generic plugin list in `10-mail.conf`:

```
mail_plugins= [...] quota
```

And add the specific `imap_quota` plugin to the `imap` module in `20-imap.conf`:

```
mail_plugins= [...] imap_quota
```

11.4. Configuring quota roots and the quota backend

All parameters below are ideally entered in `/etc/dovecot/conf.d/90-quota.conf`, or are even prepared there in a commented-out state, just waiting for you to modify and activate them.

You define the quota backend of your choice in the plugin section via the `quota =` parameter. Different backends may contain additional parameters:

```
plugin {
  #quota = dirsize:User quota
  #quota = fs:User quota
  #
  # Quotas in the file system, but only for Maildir:
  quota = maildir:User quota
  #
  # Quotas in SQL database:
  #quota = dict:User quota::proxy::quota
  #
  # Quotas in the file system, also for mdbox:
  #quota = dict:User quota::file:%h/dovecot-quota
}
```

As mentioned before, dictionaries can also be used to measure the quotas of all users in a domain in total:

```
plugin {
  #
  # Domain-wide quotas in the file system, also for mdbox:
  quota = dict:User quota:%d:file:/srv/vmail/%Ld/dovecot-quota
}
```

The term `user quota` is in this case the name of the quota area. When the email client later uses the `QUOTA` command in the IMAP protocol to query the current status, this name is also transmitted to the client (and displayed by it if applicable). You need to choose a name because there may be several quota roots, and you have to keep them apart somehow. If you have only normal quotas in the user's mailbox, just stick with `user quota`.

The quota configuration offers several other options:

`noenforcing`
: log and output quotas, but do not block when exceeded.

`ignoreunlimited`
: users with unlimited quotas are no longer tracked.

`ns=<X>`
: evaluate quotas only for namespace `<X>`.

The `noenforcing` option has been particularly useful. On the one hand, you can ensure that their quota utilization is calculated and displayed to users, but without the mailbox being blocked from receiving emails. Simply displaying the quota state in the mail client or webmailer often ensures that users clear up in reaction to the 100% fill bar.

`noenforcing` is also useful when quotas are first introduced, because it prevents users with large mailboxes from having their email reception blocked without warning. With `noenforcing`, users can be given a transition period during which they can see their quota status and clean their mailbox before `noenforcing` is removed from the configuration and the block is actually activated.

Dictionaries can be provided in the following formats:

- SQL (MySQL, PostgreSQL)
- Redis (Remote Dictionary Server)
- Flat File

In a system consisting of a single host (or an active/passive cluster with shared storage), I would always advise you to have quotas saved in a simple file. Not only is it faster, it is also more stable, and has fewer dependencies in terms of robust system operation; that is why I have chosen to describe this version in this chapter.

But there is no recommendation without restrictions. An SQL database is often easier to read out via (PHP) web frontends if you want to show end users their current quota utilization in a GUI. Alternatively, you can consider having the quota value read out by the GUI via the `doveadm quota get` command – or rely on the fact that end users are shown their quota utilization by the IMAP protocol in their email client and webmailer.

So you need to decide whether you need a database. If not, use the simple file.

11.5. Definition of the quota rules

Quota rules have the following syntax:

```
quota_rule = <mailbox name>:<limit configuration>
```

Here is a simple case for all folders of all users:

```
quota_rule = *:storage=1G
```

You can evaluate quotas by looking at the data volume (`storage=`) and by looking at the number of emails (`messages=`). A mailbox with 500,000 small emails could present a strain on the system or indicate an inactive mailbox, in which case you will want to block it from receiving any more emails.

The following inputs are permitted as quota rules:

`storage`
: quota volume in KB – `0` means unlimited.

`bytes`
: quota volume in bytes – `0` means unlimited.

`messages`
: number of messages – `0` means unlimited.

`ignore`
: the folder is ignored when the quotas are calculated.

You can maintain rules that apply in parallel if you number them at the end:

```
plugin {
  quota_rule = *:storage=1G
  quota_rule1 = INBOX/Trash:storage=+100M
  quota_rule2 = INBOX/Archive:storage=+20%%
  quota_rule3 = INBOX/Sent:ignore
  quota_rule4 = INBOX:messages=+100K
}
```

The suffixes for specifying size are `b`, `k`, `M`, `G` and `T`, with the calculations using factor 1024. So 1K means 1024, not 1000. You can also use `%%` to enter a percentage.

The rules shown here therefore define the following:

- The user can collect a data volume of 1.0 GB in this quota root (= his own mailbox).
- For emails stored in the `INBOX/Trash` folder, the quota limit is 1.0 GB + 100 MB = 1.1 GB.
- For emails stored in the `INBOX/Archive` folder, the quota limit is 1.0 GB + 20% = 1.2 GB.

- The `INBOX/Sent` folder is completely excluded from the quota calculations. The user can therefore save everything he has ever written.
- By the way, the maximum number of messages in his `INBOX` is not 100,000, but 102,400 (as 1K = 1024).

The quota rules for sub-folders no not provide additional separate memory, but simply specify the quota limits that should apply when messages are saved *in this folder*. Example: if a user has stored 1.1 GB in his trash folder while all other folders are empty, he will still be unable to save messages in his `INBOX` because the applicable size of 1 GB has been exceeded by the entire mailbox. He would, however, be able to save emails in the `INBOX/Archive` folder, as the maximum size of the entire mailbox is 1.2 GB there at the moment. At the same time, the user can keep 5 GB of his own emails in `INBOX/Sent`, as this volume is also not included when the size of the mailbox is calculated.

It is often sensible to let a user exceed his quota once. If a mailbox is "almost" full, a large message would have to be rejected while a small message could still be received. That is confusing. Use the `quota_grace` parameter to specify how far a user may exceed the quota limit when saving the "most recent email":

```
plugin {
  # allow user to become max 10% over quota
  quota_grace = 10%%
  # allow user to become max 50 MB over quota
  # quota_grace = 50 M
}
```

If you have activated detailed debugging via `mail_debug=yes`, you can check in the log file whether Dovecot has evaluated everything correctly the next time the user logs in or receives an email:

```
2013-09-20T20:50:34.745817+02:00 flash dovecot: lmtp(20009, test2@example.
2013-09-20T20:50:34.746276+02:00 flash dovecot: lmtp(20009, test2@example.
2013-09-20T20:50:34.746600+02:00 flash dovecot: lmtp(20009, test2@example.
2013-09-20T20:50:34.746956+02:00 flash dovecot: lmtp(20009, test2@example.
2013-09-20T20:50:34.747306+02:00 flash dovecot: lmtp(20009, test2@example.
2013-09-20T20:50:34.747564+02:00 flash dovecot: lmtp(20009, test2@example.
```

11.6. How to create quota warnings for users

Quota warnings in Dovecot from version 2.0 are completely different from those in the old 1.x series. So don't believe any outdated how-to guides!

First, a service `quota-warning` is set up that starts a shell script as soon as the service is addressed. Dovecot has already prepared this in file `90-quota.conf`:

```
# Example quota-warning service. The unix listener's permissions should be
# set in a way that mail processes can connect to it. Below example
# assumes that mail processes run as vmail user. If you use mode=0666, all
# system users can generate quota warnings to anyone.
#service quota-warning {
#  executable = script /usr/local/bin/quota-warning.sh
#  user = dovecot
#  unix_listener quota-warning {
#    user = vmail
#  }
#}
```

If you take my advice and save all email users under the central user ID `vmail`, you should not just activate this service, but also set the executing user to `vmail`:

```
service quota-warning {
  executable = script /usr/local/bin/quota-warning.sh
  user = vmail
  unix_listener quota-warning {
      user = vmail
  }
}
```

Once the service is set up, define one or more quota warnings:

```
plugin {
  quota_warning = storage=95%% quota-warning 95 %u
  quota_warning2 = storage=80%% quota-warning 80 %u
}
```

If the percentage specified here is exceeded, the `quota-warning` service is called up with the value (`95`) and the corresponding user name (`%u`). Dovecot transfers these two parameters to the shell script as call parameters, where they can be used as `$1` and `$2`.

Please note: in this example script, one call parameter is used to ignore quota monitoring so that quota warnings can be delivered even if a mailbox is completely full:

```
#!/bin/sh
PERCENT=$1
USER=$2
cat <<EOF | /usr/lib/dovecot/dovecot-lda -d $USER -o "plugin/quota=maildir
```

```
From: postmaster@example.com
Subject: Quota warning

Your mailbox is currently $PERCENT% full.
EOF
```

If you have difficulties implementing quota warnings, consider this:

- Warnings are only triggered if the specified threshold is actually *exceeded*.
- If the user remains above the threshold, no additional warning is generated until the next threshold is reached or the user falls below the limit and then exceeds it again “from below”.
- Check the execution permissions for the `quota_warning.sh` script. Did it set the `x` bit for the specified user?

11.7. Individual quota messages

You can specify the text that Dovecot outputs when quotas are exceeded. It can be helpful to add a URL with help/FAQs in order to minimize the support required:

```
plugin {
   quota_exceeded_message = Quota exceeded, please go to http://www.exampl
e.com/over_quota_help for instructions on how to fix this.
}
```

11.8. User-specific quotas

As shown in Section 11.5, you can set the quota rules in the Dovecot configurations flexibly for different folders. In practice, there will always be exceptions where a user or user group is granted a higher quota.

In this case, individual quota rules apply to a user; they are read out from the `passwd` file, from LDAP or from SQL, depending on the authentication source. If no individual quota value is set for the user, i.e. if the LDAP attribute or SQL column for a user is empty, the default settings from the Dovecot configuration apply. That means you can save only the exceptions and leave the fields empty for all users with default values.

11.8.1. passwd file

In the `passwd` file, the last column offers space for the *Userdb extra fields* (Section 5.12). There, you enter a user's `quota_rule`(s) directly, but marked as `userdb_quota_rule`:

```
bla@example.org:{PLAIN}test:10000:10000::/srv/vmail/example.org/bla::userd
b_quota_rule=*:storage=2G
```

11.8.2. LDAP

Individual quota rules are easy to set for an LDAP query. The condition is that you have a suitable LDAP attribute available – in this example, the `userquota` LDAP attribute.

You can now enter the full quota rule `*:storage=2G` into LDAP and then have the value of this attribute written to the Dovecot configuration `quota_rule` when the `userdb` lookup is executed:

```
user_attrs =  =home=/srv/vmail/%Ld/%Ln/,=uid=vmail,=gid=vmail,userquota=qu
ota_rule
```

This is a flexible way to specify the quota rule on an individual basis, but it is also a source of errors, because it is easy to make syntax and spelling mistakes in an LDAP entry when you modify it by hand.

It is therefore more practical to save only the number value (e.g. `2`) in the LDAP attribute and then have it entered in the `quota_rule`. Character combination `%$` is available as a placeholder to indicate where the attribute should be entered.

```
user_attrs =  =home=/srv/vmail/%Ld/%Ln/,=uid=vmail,=gid=vmail,userquota=qu
ota_rule=*:storage=%$G
```

As you can see, I have integrated the `G` for value `2G` straight back into the `quota_rule` so that the LDAP contains only pure numbers. Of course you can handle this differently; in that case, you would be responsible for adding the quantity suffix in the LDAP attribute. But be careful: if you accidentally save `2` instead of `2G`, Dovecot will interpret this as 2 KB…

To use `userdb` prefetching (Section 5.7), it makes sense to enter this rule in `pass_attrs` as well, of course with the `userdb_` prefix:

```
pass_attrs = uid=user,userPassword=password,=userdb_home=/srv/vmail/%Ld/%L
n/,=userdb_uid=vmail,=userdb_gid=vmail,userquota=userdb_quota_rule=*:stora
ge=%$G
```

11.8.3. MySQL

Even if you authenticate your users against a MySQL database, you can save the quota values in a column and uses `concat` to construct a `quota_rule` with completed syntax:

```
user_query = SELECT uid, gid, home, \
  concat('*:bytes=', quota_limit_bytes) AS quota_rule \
  FROM users WHERE userid = '%u'

password_query = SELECT userid AS user, password, \
  uid AS userdb_uid, gid AS userdb_gid, \
  concat('*:bytes=', quota_limit_bytes) AS userdb_quota_rule \
  FROM users WHERE userid = '%u'
```

11.8.4. PostgreSQL or SQLite

A similar method is used to generate a `quota_rule` in PostgreSQL:

```
user_query = SELECT uid, gid, home, \
  '*:bytes=' || quota_limit_bytes AS quota_rule \
  FROM users WHERE userid = '%u'

password_query = SELECT userid AS user, password, \
  uid AS userdb_uid, gid AS userdb_gid, \
  '*:bytes=' || quota_limit_bytes AS userdb_quota_rule \
  FROM users WHERE userid = '%u'
```

11.9. Multiple quota roots

If you have one of those rare setups where multiple quota roots are used, attach a number to the `quota` parameter in the manner familiar from the `quota_rule`.

```
plugin {
# Quotas for the individual mailbox of every user...
# 1 Gbyte per mailbox and 100,000 emails
#
  quota = dict:User quota::file:%h/dovecot-quota

  quota_rule = *:storage=1G
  quota_rule1 = INBOX/Trash:storage=+100M
  quota_rule2 = INBOX/Spam:storage=+20%%
  quota_rule3 = INBOX/Sent:ignore
  quota_rule4 = INBOX:messages=100000

  quota_warning = storage=95%% quota-warning 95 %u
  quota_warning2 = storage=80%% quota-warning 80 %u


# And then another quota level for all mailboxes in one domain
# 150 Gbyte for the domain, unlimited number of emails
#
  quota2 = dict:User quota::file:%h/dovecot-quota

  quota2_rule = *:storage=150G
  quota2_rule1 = INBOX/Sent:ignore

  quota2_warning = storage=95%% quota-warning 95 %u
  quota2_warning2 = storage=80%% quota-warning 80 %u
}
```

11.10. Working with quotas: doveadm quota

If the `quota` plugin is active, `doveadm` is also familiar with the commands `quota get` and `quota recalc`. They allow you as an administrator to read out or recalculate a user's quotas from the command line or in shell scripts:

```
flash:~ # doveadm quota get -u k.test@example.com
Quota name   Type         Value     Limit    %
user         STORAGE    1786844   2097152   85
user         MESSAGE       6613         -    0
```

In normal operation it is not necessary to recalculate quotas, as Dovecot programs update the quota calculation every time an access takes place.

It is a different case if you have performed major manual manipulations on the pool of emails in the file system, e.g. by manually deleting email files in the Maildir format or restoring them from a backup.

Also, if you have used the `doveadm` commands `backup` or `sync` to execute conversions, Dovecot may count all emails twice in its quota protocols for a short time, because it appears for Dovecot as if the entire pool of emails had been added again. Migrated users could then incorrectly encounter a quota block for a while.

For this reason, it is advisable to execute a `quota recalc` for the user in question after any major migration activities. It does not consume many resources, and you are on the safe side.

```
flash:~ # doveadm quota recalc -u k.test@example.com
```

11.11. Introducing quotas in large systems

When you activate the quota plugin for the first time, Dovecot encounters the problem that there is no information on the size and quantity of emails in the mailboxes as yet. Dovecot first needs to determine this information when the mailbox receives its first email or the user logs in for the first time.

In larger mailboxes, this involves a little work and takes several seconds. So you can imagine what your mail server would look like if, after the conversion, the first 100 or 1000 users log in at nine in the morning at more or less the same time and Dovecot executes the necessary volume calculations *now* for all 100 or 1000 mailboxes.

There are two ways to introduce quotas safely.

11.11.1. Medium-sized systems with no more than 500 users

Activate the quotas on Friday evening, when most users have already logged off and are on their way home. The calculation of quotas then mainly takes place over the weekend. There are fewer logins at the weekend, and they are usually distributed over time. At the same time, the emails received during those two-and-a-half days until Monday morning will trigger an initial quota calculation in the mailboxes. During the peak on Monday, the calculations then only need to be performed for mailboxes whose users did not log on during the weekend or that did not receive any emails during that time. Typically, these mailboxes are smaller and do not belong to power users.

There will probably be a storm in the teacup at 9 on Monday morning until everyone has logged in for the first time, but then everything will calm down.

Alternatively, you can take advantage of the weekend quiet and run the `doveadm quota recalc` command described in Section 11.10 on all the accounts.

11.11.2. Large systems with more than 500 users

If you have too many simultaneous logins, and many people log in simultaneously even in the evening and at weekends, you have to find a different solution.

Set up the quotas completely, but do not yet activate the quota plugin so that Dovecot does not perform any quota calculations yet.

Then run the command

```
flash:~ # doveadm -o mail_plugins=quota quota recalc -u <username>
```

or more simply

```
flash:~ # doveadm -o mail_plugins=quota quota recalc -A
```

for all users. `doveadm` (unlike in the real configuration) loads the `quota` plugin, and the `quota recalc` command ensures that the current quota of the mailbox is recalculated. The takes just a few seconds for every mailbox – you can calculate 10,000 or more mailboxes *in a row* in a single night. After all, you are unable to perform these calculations *simultaneously* for all mailboxes on Monday at 9 in the morning.

Once the current quota has been calculated for all mailboxes, activate the quota plugin (in the evening or at the weekend) in the regular Dovecot configuration as well. Dovecot will immediately continue keeping the quota journals without any strain worth mentioning. The fact that some small changes may have occurred in the meantime in some mailboxes is not particularly important. Dovecot recognizes the changes. If in doubt, you can run the `doveadm quota recalc` command for all mailboxes again after the quotas have been activated. You will see that this will take far less time in all later runs.

One more thing: `doveadm` only knows the `quota` command if the `quota` plugin is loaded. If it is missing, the `quota` command is not even output in the `doveadm` help.

11.12. The Quota policy server for Postfix

A typical problem for quotas on the IMAP server is that SMTP front relays usually know nothing about the quota condition of a mailbox. They would always accept an email addressed to an (existing) recipient – even if this email cannot be delivered to Dovecot shortly afterwards because the quota has been exceeded. In the end, the recipient would receive a *late bound* – with all the consequences and effects in the event of forged senders in spam or virus emails.

But there is a quick solution to this problem in Dovecot: in version 2.2 and higher, Dovecot provides a `quota-status` policy server for Postfix. Postfix email relays can use this to query the quota status of a mailbox from the Dovecot server when receiving the email – and, if the quota has been exceeded, they can refuse to accept the email and reject it in real time.

The setup does not take long. These configuration examples are not yet part of the sample configurations in Dovecot, so you will need to type in these lines yourself. You could add them to `90-quota.conf`, for example; just temporarily create `91-quota-status.conf` to prevent overwriting of anything in the event of an update.

First, the `quota-status` module must be available from a TCP/IP port so that the other servers can query the service. In this case, it is TCP port 12340:

```
service quota-status {
    executable = quota-status -p postfix
    inet_listener {
        port = 12340
    }
    client_limit = 1
}
```

You should then configure the module in a `plugin {}` section and specify which feedback is returned to Postfix and evaluated there in the `smtpd_recipient_restrictions`:

```
plugin {
    quota_status_success = DUNNO
    quota_status_nouser = DUNNO
    quota_status_overquota = "552 5.2.2 Mailbox is full"
}
```

Even if a mailbox is able to receive emails, you must stick to `DUNNO`; you may not set `OK` or `PERMIT`. Otherwise Postfix would always accept the email, even if other spam protection rules would prevent it from doing so otherwise.

The response should always be `DUNNO` (“no decision”), so the email is neither rejected nor accepted, but simply subjected to further verification by Postfix. `DUNNO` should also be

returned if the user is unknown, in case you process emails for multiple IMAP backends in parallel.

Once your configurations are complete, you have to install the policy check in your Postfix relays in the right part of `smtpd_recipient_restrictions`.

```
smtpd_recipient_restrictions =
    [...]
    check_policy_service inet:imap.example.com:12340
    [...]
```

This call should take place *after* `permit_mynetworks` or `permit_sasl_authenticated` so your Postfix relay is not tempted to subject emails to external recipients to an internal quota check.[54]

The quota service can take place *before* or *after* your spam protection measures, e.g. RBL, Greylisting and/or Policyd-Weight, depending on which method is more *expensive*. RBL, Greylisting and Policyd-Weight are comparatively *cheap* checks that already reject the majority of spam emails that then cannot trigger a quota check, so this method conserves resouces. My recommendation is to use `quota-service` *after* the spam protection measures and therefore directly before the end of `smtpd_recipient_restrictions`.

Based on chapter 8.8 in my Postfix book, the best `smtpd_recipient_restrictions` would look like this:

```
#
# Sample solution for smtpd_recipient_restrictions in line with the
# Postfix book -- Heinlein Support GmbH, http://www.heinlein-support.de
#
smtpd_recipient_restrictions =
# Whitelist recipient?
        check_recipient_access hash:/etc/postfix/access_recipient-rfc
# Blacklist host and sender?
        check_client_access cidr:/etc/postfix/access_client,
        check_helo_access hash:/etc/postfix/access_helo,
        check_sender_access hash:/etc/postfix/access_sender,
        check_recipient_access hash:/etc/postfix/access_recipient
# Do not accept dodgy emails!
        reject_non_fqdn_sender,
        reject_non_fqdn_recipient,
        reject_unknown_sender_domain,
        reject_unknown_recipient_domain,
        reject_invalid_hostname,
# Permit our children!
        permit_sasl_authenticated,
        permit_mynetworks,
# Prohibit all other relaying!
        reject_unauth_destination,
# Check RBL!
        reject_rbl_client zen.spamhaus.org,
        reject_rbl_client ix.dnsbl.manitu.net,
```

```
# Check Policyd-weight!
        check_policy_service inet:127.0.0.1:12525
# Check Greylisting!
        check_policy_service inet:127.0.0.1:10023
# Dynamic recipient verification
        reject_unverified_recipient,
# Check quota status of the user on the IMAP server
        check_policy_service inet:imap.example.com:12340
# Everything that is left can be let through!
        permit
```

---

[54] A small gap remains: internal users can still send emails to users who have exceeded their quotas. These emails will bounce shortly afterwards. However, as the emails come from your own users and are not spam messages, the effects are small and not problematic.

Chapter 12. Server-side email filtering with Sieve

Support for server-side email filtering using Sieve scripts used to be the domain of Cyrus IMAP, and one which users of other IMAP servers eyed enviously. But these times have passed. Dovecot of course offers comprehensive Sieve support that goes far beyond that of Cyrus Sieve in the scope of functions and quality of implementation.[55]

Many administrators still perform filtering in the old manner, using scripts and the `procmail` MDA, which is no longer usable if emails are transferred to Dovecot via LMTP as shown in Section 6.2. `procmail` scripts also require shell access to the mail server; on old email systems, that was possible because users were still actually entered in `/etc/passwd`, but it no longer works in current email systems with "virtual users" entered in `/etc/dovecot/users`, in SQL or in LDAP.

So it is definitely worth switching to Sieve – and administrators should go to the effort of transferring existing Procmail scripts to Sieve. Less support will be required, because the `managesieve` protocol allows users to adapt and activate their Sieve scripts independently via their email client or a webmailer such as Roundcube. And, last but not least, Sieve makes it more convenient to sort emails from work, private contacts or mailing lists, forward certain emails automatically to other users, or use the `vacation` module in Sieve to deal with out-of-office notifications. Even if administrators are usually not particularly fond of the last option.

Here is a list of the advantages:

1. Dovecot contains a `managesieve` daemon that used to listen on TCP port 2000 (now obsolete), and now listens on port 4190. The Managesieve protocol is a bit like a mixture of IMAP and FTP: users can log in and upload or download script files from their Sieve directory on the server.
2. Various script files can contain one or more Sieve rules. Only one script file can be active at any one time.[56] But users can use this method to prepare specific sets of rules and switch between rules for when they are in the office and when they are on vacation. Of course the Managesieve protocol also offers the client the ability to choose which file of rules is to be active at any one time.
3. When an email is saved via Dovecot's LMTP daemon or Dovecot's `dovecot-lda` program, the Sieve scripts are executed if the `sieve` plugin was included via `mail_plugins`.

12.1. Setting up Sieve

This is how you make Sieve available to your users: as Managesieve is developed by Stephan Bosch under the project name “Pigeonhole”, it is packaged separately by some distributions. Users of Debian/Ubuntu should first ensure that the `dovecot-managesieve` package is installed.

But don’t worry, Stephan Bosch and his project are firmly integrated in Dovecot. Pigeonhole/Managesieve is as safe and reliable as if Managesieve had been created directly by Timo Sirainen.

In order for Managesieve to be packaged separately, the settings for the `managesieve` daemon are located not in `10-master.cf`, but in a file named `20-managesieve.conf`.

Activate the `managesieve` daemon there. In the past, port 2000 was used for that purpose, but it has now been allocated to Cisco, while official port 4190 has been reserved for Managesieve. Ancient clients may still address port 2000 by default. If in doubt, simply activate Managesieve on both ports; there is no reason not to.

```
service managesieve-login {
  inet_listener sieve {
    port = 4190
  }

  inet_listener sieve_deprecated {
    port = 2000
  }
}
```

Debian activates the `managesieve` daemon automatically as soon as the corresponding package has been installed. On SUSE systems, where Managesieve is already part of the Dovecot package, you have to activate the `managesieve` daemon separately. To do so, change the `protocols` entry in `/etc/dovecot/dovecot.conf`. But be careful. Confusingly, the corresponding entry is not `managesieve` but `sieve`:

```
# Protocols we want to be serving.
protocols = imap pop3 lmtp sieve
```

After a Dovecot restart, you should use `lsof` to view the open ports in IPv4 and, if applicable, IPv6, before continuing.

```
flash:~ # lsof -i :2000
COMMAND  PID USER   FD   TYPE  DEVICE SIZE/OFF NODE NAME
dovecot 9735 root   17u  IPv4 1818920      0t0  TCP *:sieve (LISTEN)
dovecot 9735 root   18u  IPv6 1818921      0t0  TCP *:sieve (LISTEN)
```

```
flash:~ # lsof -i :4190
COMMAND  PID USER   FD   TYPE  DEVICE SIZE/OFF NODE NAME
dovecot 9735 root   15u  IPv4 1818918      0t0  TCP *:4190 (LISTEN)
dovecot 9735 root   16u  IPv6 1818919      0t0  TCP *:4190 (LISTEN)
```

The Sieve scripts are executed by the LMTP daemon (or the `dovecot-lda` program) if the emails are saved in the user mailboxes. For the other parts of Dovecot, i.e. the `pop3` or `imap` daemon, `sieve` currently plays no role.[57]

Go to `20-lmtp.conf` and add the `sieve` plugin to the `mail_pugins` parameter described elsewhere in this book:

```
protocol lmtp {
  # Space separated list of plugins to load (default is global
  # mail_plugins).
  mail_plugins = $mail_plugins sieve
}
```

If you deliver your emails not via LMTP but via the `dovecot-lda` program, you naturally have to expand the `mail-plugins` in `15-lda.conf` as well:

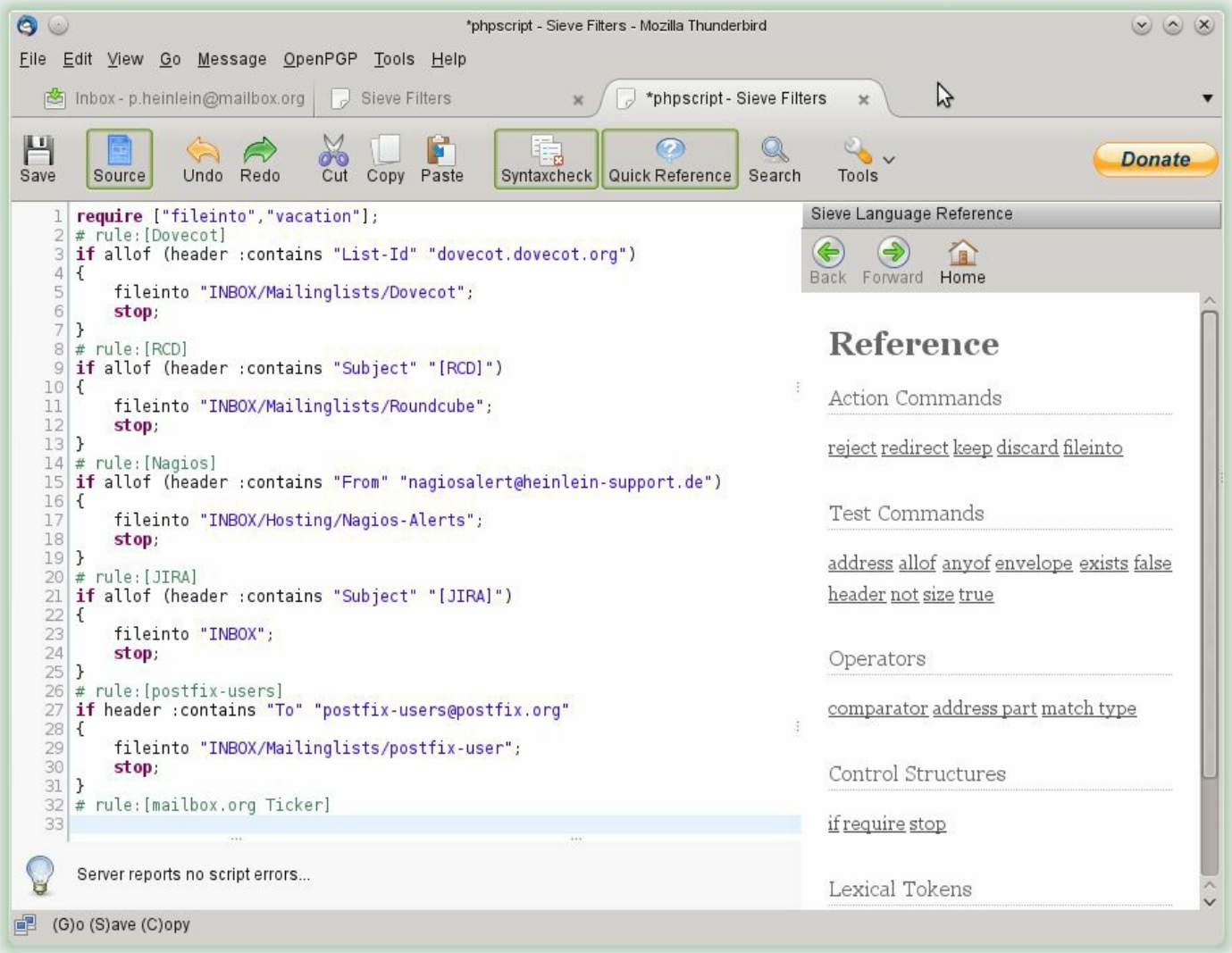
```
protocol lda {
  # Space separated list of plugins to load (default is global
  # mail_plugins).
  mail_plugins = $mail_plugins sieve
}
```

The `90-sieve.conf` file contains the settings for said `sieve` plugin. These are all sensible settings, so you should not usually modify them. From the start of `90-sieve.conf` you can see that Dovecot saves Sieve scripts by default in the "sieve" subfolder in the user's (virtual) home directory, and that the currently active Sieve script is searched for under `.dovecot.sieve`:

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

Now you should create, upload and activate your own test script in Sieve. Many email clients offer (unfortunately often rudimentary) Sieve support, as Figure 12.1 clearly shows.

Figure 12.1. ASCII editor with syntax highlighting – that is all Thunderbird could come up with when it comes to “Sieve”.



For testing purposes, I recommend the *Roundcube* webmailer, which is very good in general and contains an excellent Sieve plugin (Figure 12.2). In Section 18.2.2 I also demonstrate how to activate the `managesieve` plugin in Roundcube.

If everything works (read the log file), you will see on the file level that an ASCII-readable Sieve file has been saved in the `~/sieve` directory. The name of the file is unimportant and is often assigned “at random” by the email client in question.

You should also see that this Sieve script is active; this is achieved rather elegantly by `managesieve` creating a symlink from `~/.dovecot.sieve` to the required Sieve script in the `~/sieve` directory:

```
root@flash:/srv/vmail/heinlein-support.de/p.heinlein# ls -l
60 in total
lrwxrwxrwx  1 vmail vmail    21 31. Aug 2011  .dovecot.sieve -> sieve/phps
drwx------  4 vmail vmail  4096 26. Apr 11:30 mdbox
drwx------  3 vmail vmail  4096 26. Apr 22:20 sieve
```

When an email is delivered via LMTP or `dovecot-lda`, the log file should definitely contain a reference to the `sieve` plugin, whether sets of rules have triggered filtering or not:

```
Apr 21 06:28:11 mail dovecot: lmtp(28023, klaus@example.com): S96TFGNqc1F3
```

A simple example script for filtering emails in the Dovecot mailing list could look like this: [58]

```
require ["fileinto"];
# rule:[Dovecot]
if allof (header :contains "List-Id" "dovecot@dovecot.org")
{
        fileinto "INBOX/mailinglists/Dovecot";
        stop;
}
```

The `#` comment line at the beginning is not part of the syntax itself; it comes from Roundcube in this case, which uses this method to save the displayed name of the filter rule.

If the filter rule is applied, the whole process can be found in the log file:

```
Apr 21 15:35:37 dobby4 dovecot: lmtp(29531, p.heinlein@heinlein-support.de
```

By the way, a precompiled script is generated from the ASCII version of the Sieve script and placed in the user's home directory as `~/.dovecot.svbin`. You don't need to do anything about it – Dovecot detects automatically if the underlying Sieve script has changed and `.dovecot.svbin` has to be compiled again.

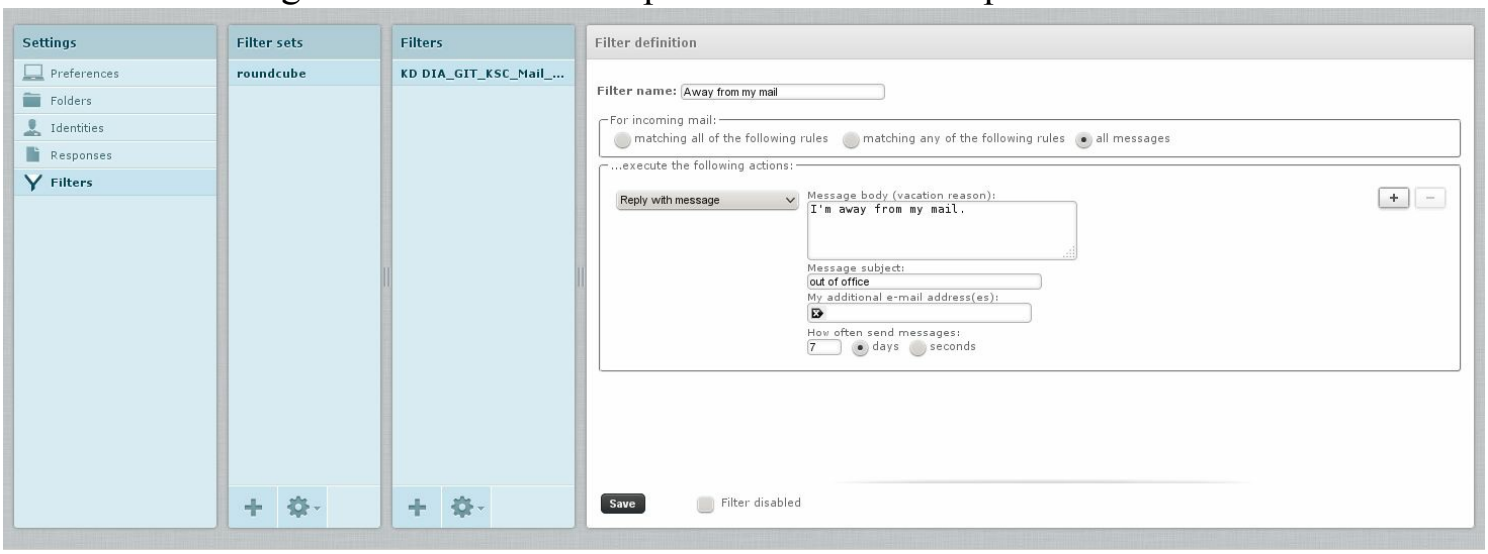
If you read the `90-sieve.conf` file, you will see that you can use `sieve_before` and `sieve_after` to define global Sieve scripts that are executed automatically for every user. This is a convenient way to map old central `procmail` rules in particular (for example those used to filter by spam markings). As the other Sieve parameters are documented rather well there, we won't provide a translation here.

And that is almost all there is to say about Sieve – but not quite…

12.2. Out of office response via Sieve script

Of course, it is not quite as simple as that. Autoresponder actions in Sieve, so the use of the `vacation` module, has two important unusual features.

Figure 12.2. Vacation responder via Sieve script in Roundcube



A typical out of office reply like that generated with Roundcube in Figure 12.2 could look like this, for example:

```
require ["vacation"];
# rule:[vacation]
if true
{
        vacation :days 4 :subject "I'm on vacation... :-)  " text:
Now I've finished the Dovecot book I will be away
and offline for a while.

For urgent matters, please contact my colleagues:
support@heinlein-support.de, or phone +49 30 40 50 51 - 0.
.
;
}
```

In order to prevent email loops between two autoresponders, responses should not be sent to every email, but only every <*n*> days. If no other entry is made, Sieve assumes 7 days – in this example, the limit was reduced to 4 days by entering `days 4`.

So, when you are testing Sieve, remember that everything might be working fine and you are simply not receiving a response because you have already received one during your tests.

Sieve uses the `.dovecot.lda-dupes` in the home directory of the user to log who has received an answer and when. If in doubt, delete the `.dovecot.lda-dupes` file – all

response blocks are removed and Sieve starts from the beginning.

```
root@flash:/srv/vmail/heinlein-support.de/p.heinlein# ls -la
60 in total
drwx------  6 vmail vmail  4096 26. Apr 21:49 .
drwxr-xr-x 84 vmail root   4096 17. Apr 11:08 ..
-rw-------  1 vmail vmail  2750 26. Apr 18:35 .dovecot.lda-dupes
lrwxrwxrwx  1 vmail vmail    21 31. Aug 2011  .dovecot.sieve -> sieve/phps
-rw-------  1 vmail vmail  9073 26. Apr 09:13 .dovecot.sieve.log
-rw-------  1 vmail vmail  1234 28. Mar 08:01 .dovecot.svbin
drwx------  4 vmail vmail  4096 26. Apr 11:30 mdbox
drwx------  3 vmail vmail  4096 26. Apr 22:20 sieve
```

In the `dovecot.sieve.log` file you can also see here, errors occurring during the interpretation of the Sieve script are logged – but Dovecot also always enters a warning in the normal email log file as well.

Careful, if your login name is different from the email addresses used (see Section 5.10.6), Sieve will always "know" what user name it is currently running under. At the same time, Sieve will for safety reasons only return autoresponder/vacation messages if it finds the current user name in a `Header-To` or a `Header-CC` field. That is intended to prevent mailing loops or unintentional responses to BCC or mailing list emails.

If you do not use your email address as your login name (= Sieve name), Sieve will not find what it is looking for in the email header and will therefore not respond. In such cases, you can define alternative email addresses in the Sieve script by entering `addresses`, and Sieve will search for these addresses in the email header as well:

```
# rule:[vacation]
if true
{
  vacation :days 4 :addresses ["p.heinlein@heinlein-support.de",
"p.heinlein@dovecot-book.com"] :subject "I'm on vacation... :-)   " text:
Now I've finished the Dovecot book I will be away and offline for a while.

For urgent matters, please contact my colleagues:  support@heinlein-suppor
.
;
}
```

Figure 12.2 shows how Roundcube queries the alternative addresses in a separate field. Instruct your users to enter their actual email addresses here.[59]

The most proper way is to route emails to Dovecot during the LMTP transfer using the actual email addresses rather than the login names. Section 5.10.6 describes how to configure the `userdb` query in Dovecot in such a way that login name and email address both work.

Sieve scripts are then always executed in the context of the email address, and the dilemma described here does not even develop.

12.3. Converting old procmail scripts

If you use system-wide or user-specific filter rules in `procmail`, it may be worth taking a look at http://www.dovecot.org/tools/procmail2sieve.pl. The script converts `procmail` scripts into Sieve syntax.

12.4. Applying Sieve scripts to emails that have already been received

With release version 0.3, Pigeonhole maintainer Stephan Bosch also released the `sieve-filter` script that allows emails to be filtered with Sieve even if they have already been received. Previously, you would have needed a rather adventurous construction where emails are read out of the mailbox and returned to Dovecot again via SMTP and LMTP/`dovecot-lda`.[60]

`sieve-filter` is an elegant solution and is familiar with the following call parameters, which are explained in detail in `man sieve-filter`.

```
flash:~ # sieve-filter
Usage: sieve-filter [-c <config-file>] [-C] [-D] [-e] [-m <default-mailbox
            [-P <plugin>] [-q <output-mailbox>] [-Q <mail-command>]
            [-s <script-file>] [-u <user>] [-v] [-W] [-x <extensions>]
            <script-file> <source-mailbox> [<discard-action>]
```

The important and relevant options and arguments are:

`-v`
: Verbose output, a detailed log of all activities.

`-u <username>`
: As with `doveadm`, you have to transfer the user name of the user whose mailbox is to be filtered.

`-e`
: For security reasons, `sieve-filter` first accesses the entire mailbox in read mode only. That allows you to check whether everything is working. It is option `-e` (*Execution Mode*) that then performs this filtering.

`-W`
: Even if `-e` is active, `sieve-filter` would not delete the emails it finds in the INBOX (if the Sieve action is `discard`), and it would only copy emails to other folders, not move them there. In other words: access to the actual INBOX would continue to be read-only, resulting in plenty of duplicate emails. `-W` (*Write*) grants `sieve-filter` write access to the INBOX, which could result in deleted emails.

`<script-file>`
: Path to the Sieve script.

`<source-mailbox>`
: The name of the mailbox of the inbox (= IMAP folder) to which the rules are to be applied. As a rule this will be `INBOX`.

Here is an example call for a test run:

```
flash:~ # sieve-filter -v -u klaus@example.com /srv/vmail/example.com/klau
```

If the test is successful, `-e -W` is used to let the script go “live”:

```
flash:~ # sieve-filter -e -W -v -u klaus@example.com /srv/vmail/example.co
```

There is one more thing you need to remember if you use the `discard` Sieve action in your scripts. Usually, Sieve would delete the affected emails – but `sieve-filter` does not do so for reasons of safety. Instead, you can optionally enter `<discard-option>` as the last call option to instruct `sieve-filter` how to proceed.

The following options are possible:

`keep`
: ignores the `discard` action and leaves the email folders untouched (default).

`move <foldernname>`
: moves the email to the `<foldername>` folder, e.g. `move INBOX.Test`.

`delete`
: marks the messages as deleted, i.e. it sets the `\Deleted` IMAP flag.

`expunge`
: irretrievably deletes the message.

Here is an example call that actually deletes discarded emails:

```
flash:~ # sieve-filter -e -W -v -u klaus@example.com /srv/vmail/example.co
```

Do a test run

If you want to be on the safe side, copy the affected mailbox to a test account first – at the file level or using `dsync`. You can then safely run the `sieve-filter` script and check the result.

---

[55] Some Cyrus Sieve commands were not even RFC-compliant.

[56] There are some complicated ways to include additional script files.

[57] That may change: RFC 6785 specifies how to integrate Sieve actions in the IMAP protocol as well: https://tools.ietf.org/html/rfc6785

[58] Here with `/` as the namespace separator

[59] In a few projects we simply hard-coded a suitable LDAP query and patched it into the Roundcube source code. A pragmatic and effective solution, at least for all Roundcube users.

[60] http://wiki2.dovecot.org/HowTo/RefilterMail

Chapter 13. Email extensions

Email extensions are a little-known method for handling email addresses creatively. They are appreciated by everyone who is familiar with them.

13.1. The recipient delimiter extends email addresses

By introducing a *delimiter*, i.e. a special character such as `+`, an additional text can be added to an email address between the user name and the `@` sign (the *extension*). `user@example.com` can now become `user+personal@example.com`. As soon as the involved mail servers (so Postfix and Dovecot) know that the `+` is to be understood as an extension delimiter, they will treat the email address accordingly from that moment.

`+` is rarely used in user names or domains, so it is a common delimiter.

This makes it easy to mark email addresses for individual purposes – for example to sort mailing list emails or private email correspondence more easily later on.

However, Postfix and Dovecot do not ignore the extension completely. Instead, they first perform their lookups for the full email address with the extension. In our example, Postfix looks up `user+personal@example.com` in the `access`, `virtual`, `canonical` or `transport` Postfix tables. That means that email addresses with an extension can be handled differently from pure email addresses. Dovecot also first looks for a user with that name.

As the first query for the full email address does not usually return a hit, Postfix and Dovecot use their knowledge of the delimiter in a second step and separate off the email extension. Now they just look for `user@example.com`. That means the emails is delivered to the actual user as normal.

In order to activate email extensions, you have to add the following parameter in Postfix in `main.cf`:

```
recipient_delimiter = +
```

The parameter has the same name in Dovecot; it is prepared twice, once in `15-lda.conf` and once in `90-sieve.conf`. Of course you only need to activate it once, for example in `15-lda.conf`:

```
# Delimiter character between local-part and detail in email address.
recipient_delimiter = +
```

There are some pretty smart applications:

- You can allow email addresses with a specific extension in the Postfix `access` maps and block those without an extension (and vice versa).
- You can forward email addresses differently via the Postfix `virtual` map – and reroute them using the `transport` map (test emails, for example).
- As a user, you can use Sieve rules to evaluate the email extension, for example in order to sort mailing list emails directly into the correct subfolder.

13.2. Automatic saving of emails in IMAP folders

You can also instruct Dovecot to save emails automatically in an IMAP folder with the name of the email extension under the INBOX. That allows emails to be sorted automatically, so that no customized Sieve rules are required.

The automatic storage in the IMAP folder of the same name needs to be activated separately in Dovecot. If LMTP is in use, it is enough to make the following setting in `20-lmtp.conf` (Dovecot calls the email extension `detail` in its comments):

```
# If recipient address includes the detail (e.g. user+detail), try to save
# the mail to the detail mailbox. See also recipient_delimiter and
# lda_mailbox_autocreate settings.
lmtp_save_to_detail_mailbox = yes
```

If, however, `dovecot-lda` is in use, it is advisable to modify the transport method in Postfix in `master.cf` as follows (see Chapter 6):

```
dovecot unix - n n - 5 pipe flags=DRhu user=vmail:vmail argv=/usr/lib/dove
cot/dovecot-lda -f ${sender} -d ${user}@${domain} -m ${extension}
```

If, as recommended in this book, you use `INBOX/` as the prefix for all IMAP folder names, you should call `dovecot-lda` as follows:

```
dovecot unix - n n - 5 pipe flags=DRhu user=vmail:vmail argv=/usr/lib/dove
cot/dovecot-lda -f ${sender} -d ${user}@${domain} -m INBOX/${extension}
```

Both `dovecot-lda` and LMTP require by default that a Maildir folder with the name of the email extension in question already exists (in our example, that would be `INBOX.personal`). If the folder is not there, Dovecot saves the email to the `INBOX` and ignores the extension. You can modify this behavior with two parameters in `15-lda.conf`. You can also instruct Dovecot to create non-existent folders automatically and then subscribe to them.

```
# Should saving a mail to a nonexistent mailbox automatically create it?
lda_mailbox_autocreate = yes

# Should automatically created mailboxes also be automatically subscribed?
lda_mailbox_autosubscribe = yes
```

If `mail_debug=yes` is active, you can see from the log file how Dovecot creates the folder and saves the email correctly:

```
2013-08-02T20:21:02.711389+02:00 dovecottest2 dovecot: lmtp(4766, klaus@ex
2013-08-02T20:21:02.711934+02:00 dovecottest2 dovecot: lmtp(4766, klaus@ex
2013-08-02T20:21:02.769359+02:00 dovecottest2 dovecot: lmtp(4766, klaus@ex
```

Chapter 14. Daily maintenance of the email storage with doveadm

In version 2.x and higher, `doveadm` is a powerful administrative tool that is indispensable once you have got used to it. We have examined it in several parts of the book so far, for example when discussing quotas and matters of authentication.

This chapter is about the work performed every day in the users' email storage once the Dovecot server is up and running.

If the email storage is operated in the Maildir format, it is easy to work directly at the directory level with your own tricks and scripts. There is nothing in Maildir that cannot be edited or manipulated directly in ASCII: you can delete or copy email files, create or rename directories (i.e. IMAP folders) and so on. Even help files such as the `subscriptions` file with folder subscriptions are simply text files.

It is an entirely different matter for mdbox storage, as I warned you in Section 7.3. Many essential parts are in a database, so direct manipulation at file level is impossible.

But you should get into the habit of using `doveadm`, regardless of the storage system in use. Above all, you should always base your own scripts and workflow on `doveadm` instead of performing manipulations at file level. That way, a change of format or mixing of storage formats will not be a major issue.

14.1. General operation of doveadm

For all the commands described below, `-u <username>` is usually used to specify the user to be processed.

You can also use wildcards, such as `*@example.com` or even just `*`. If you want a command to apply to all users, you can replace `-u *` with `-A`; we have omitted that option later on to make the text easier to read. The prerequisite is that Dovecot has access to the complete list of all users. In SQL or LDAP data pools, for example, the *iterate query* must be set up for this purpose (see Section 5.13).

Whenever `-u <user>` or `-A` is used, you can also use `-S <socket_path>` to specify a file path to another authentication socket. That can be helpful in tests or multi-instance setups. You will not need to do so during normal operation, so I will not list the option under `doveadm` commands any more in the interest of making the text easier to read.

In the context of `doveadm`, remember that the term *mailbox* refers not to a user's account, but to an IMAP folder, i.e. to `Sent`, `Trash`, `INBOX`. If `mailbox` is demanded in `doveadm` as a parameter, you need to specify IMAP folder names, and you can once again use wildcards: `*project*` would be permitted as an input.

Make sure you always enclose a stand-alone `*` in inverted commas in `doveadm` calls to prevent the shell from expanding it with a list of the files in the local directory.

It is important to remember the spaces or special characters such as `&` (see Section 14.2.2) in folder names when working with folder names during conversions. Always use quotation marks to protect variables contained in folder names. Always write `"$FOLDER"` when working in shell scripts.

Finally, there are the following generic options for `doveadm`:

`-f <format type>`
: influences the output format. Available format types:

`flow`
: outputs values as `key=value`.

`pager`
: outputs values as `key:value`.

`tab`
: outputs a table header, values are separated by tabs.

`table`
: outputs a table header, values are indented with spaces.

`-v` activates detailed output, including a progress bar if applicable. `-D` activates debut output.

Before you descend to the depth of mailboxes with `doveadm`, it may be helpful to remind yourself of two details and terms relating to the IMAP protocol:

sequential number and *UID*
: Emails in a folder have a *sequential number*, i.e. one increasing from `1` to `<n>`, corresponding to the number of emails in the folder, and they also have a UID. When an email is deleted, the sequential number of all subsequent emails changes. Sequential numbers are therefore not a reliable criterion when working with emails. For this reason, an email is usually referred to by its UID (*unique identifier*), which must remain the same throughout the email's lifetime.

GUID and *UID validity*
: Every IMAP folder has a GUID that is used to address it when the email client is synchronized with the server. If a folder is deleted and created again, its name does not change, but its GUID does. If the index of an IMAP folder is destroyed, the folder's GUID does not change, but the *UID validity* does, because all emails in the folder are given new UIDs, and these may not be confused with UIDs that may have been assigned earlier.

This is just a brief reminder, as the topic was discussed in detail in Section 3.2.1.

14.2. Managing IMAP folders

The commands in `doveadm mailbox` are used to manipulate a user's folders – and are also well suited for major migrations and adjustments to all user accounts.

14.2.1. Creating, renaming and deleting folders

`doveadm mailbox create|delete|list|rename -u <user> <folder>`
: used to create, delete, list or rename a user's IMAP folders.

These examples show how to use the commands:

```
doveadm mailbox list   -u klaus@example.com
doveadm mailbox create -u klaus@example.com INBOX/Test
doveadm mailbox list   -u klaus@example.com
doveadm mailbox rename -u klaus@example.com INBOX/Test INBOX/trash
doveadm mailbox delete -u klaus@example.com INBOX/trash
```

Parameter `-s` in `create`, `delete` and `rename` commands Dovecot to subscribe or unsubscribe these folders at the same time.

14.2.2. Converting special characters in folder names

`doveadm mailbox mutf7 <folder>`
: converts folder names from UTF-8 to UTF-7 and back.

IMAP folder names with umlauts and national special characters are usually stored in UTF-7 (= 7 bit) in the file system, and not in UTF-8 (=8 bit). In `doveadm` itself, you can always use the correct spelling with umlauts. If, however, you are working directly in the file system, e.g. during a conversion by means of shell scripts, it can be helpful to have the correct folder names calculated.

By default, the `mutf7` command converts an 8-bit name into a 7-bit name; in this case, I have used a German word with multiple umlauts:

```
flash:~ # doveadm mailbox mutf7 Tüpfelhyänenöhrchen
T&APw-pfelhy&AOQ-nen&APY-hrchen
```

The additional parameter `-7` specifies that the source name is provided in 7 bit and should be converted back to 8 bit:

```
flash:~ # doveadm mailbox mutf7 -7 "T&APw-pfelhy&AOQ-nen&APY-hrchen"
Tüpfelhyänenöhrchen
```

14.2.3. Querying the status and number of emails in a folder

`doveadm mailbox status -u <user> [-t] <fieldlist> <folder>`
: lists the status of the folders that match the `<folder>` pattern. Use additional parameters to specify which information on the folder should be returned.

In `<fieldlist>`, specify the fields you need from the folder in question:

`guid`
: the globally unique identifier of the folder.

`highestmodseq`
: Dovecot logs the number of changes made to an index by means of a `modseq`, a *modification sequence number*. In addition, Dovecot can also detect and correctly synchronize parallel changes on both sides in replication scenarios. Usually you should not need to work directly with the `modseq`.

`messages`
: Number of messages in the folder.

`recent`
: Number of new messages (`\Recent` flag).

`uidnext`
: next UID (that will be assigned to the next email).

`uidvalidity`
: *Unique identifier validity* value of the folder.

`unseen`
: Sequential number of the next unread message in the folder.

`vsize`
: Virtual size of the IMAP folder.[61]

If you specify multiple fields, you have to enclose them in inverted commas or quotation marks as shown in this example:

```
flash:~ # doveadm mailbox status -u klaus@example.com 'recent messages' IN
INBOX messages=94 recent=35
```

Instead of listing individual field names, you could also use `all`:

`all`
: lists all available fields. But be careful. If `-t` has been specified as well, only `messages`, `recent`, `unseen` and `vsize` will be listed.

```
flash:~ # doveadm mailbox status -u klaus@example.com all '*'
INBOX/Sent messages=2 recent=2 uidnext=6 uidvalidity=1324505962 unseen
INBOX/Drafts messages=0 recent=0 uidnext=7 uidvalidity=1324505963 unse
INBOX messages=92 recent=33 uidnext=94 uidvalidity=1324505961 unseen=8
INBOX/TEST messages=0 recent=0 uidnext=1 uidvalidity=1324505964 unseen
```

You can also obtain the total value rather than the individual value per folder.

`-t all`

forms a total for `messages`, `recent`, `unseen` and/or `vsize` from all folders matching the pattern.

```
flash:~ # doveadm mailbox status -u klaus@example.com -t all '*'
messages=94 recent=35 unseen=81 vsize=30743586
```

14.2.4. Subscribing to folders

`doveadm mailbox subscribe|unsubscribe -u <user> <folder>`
: subscribes or unsubscribes a user's IMAP folders, see also Section 3.2.4.

You do not need to make manual entries in the `subscriptions` file, because `doveadm` can do that for you. This has the additional advantage that you do not have to think about converting 8-bit special characters in the folder name.

14.3. Searching for emails

Use the `doveadm search` or `doveadm fetch` commands to search for emails according to specific criteria. You can of course use some shell scripts and `grep` to search Maildir directories for senders, subject lines or other text patterns, but here you also have access to interesting meta information such as the time when an email was received. You can also combine multiple criteria. The `doveadm` search accesses the index and cache files and is therefore faster than a grep search at file system level. Besides, you don't have a choice when it comes to `mdbox` or other storage formats.

`doveadm search -u <user> <searchpattern>`
: finds emails that match the search pattern, but returns only the UIDs of the IMAP folder (where the email was found) and the matching emails.

The search pattern always consists of the keyword (e.g.: `MAILBOX`, `SUBJECT` or `FROM`) and, where it makes sense, of a text pattern that should be used to search for the keyword, so `MAILBOX inbox`, `SUBJECT Postfix` or `FROM heinlein`.

Many keywords refer to yes/no flags and therefore have no additional text pattern: examples include `RECENT`, `ANSWERED` and `UNANSWERED`.

Text patterns in the Subject or From fields are always searched for as components, so it is enough for the search term to occur in some place, just as it is for `grep`. Folder patterns are however treated in a special way to prevent `MAILBOX inbox` from matching to all one hundred folders of a user. In this case, you have to work explicitly with `*`: `MAILBOX inbox/project-*`.

Some examples quickly illustrate how the command behaves in practice:

```
flash:~ # doveadm search -u peer@example.com mailbox INBOX subject Postfix
40c515265e5dbb4d704100008abf93d5 137623
40c515265e5dbb4d704100008abf93d5 138692
```

It is sometimes confusing if multiple fields and patterns are combined in a row. I have therefore got into the habit of capitalizing all fields as you can see from the listings.

Working with `doveadm search` is usually not the last step, because it returns only the UIDs of the emails and the GUIDs of the folders, i.e. references to the actual email that you can then process further.

This example shows how to call up an additional command with the output:

```
flash:~ # doveadm search -u peer@example.com mailbox INBOX subject Postfix
while read guid uid ; do
```

```
        doveadm fetch -u peer@example.com text MAILBOX-GUID $guid UID $uid
done
```

Don’t worry, this example just shows how to process the data further. If you want to fetch emails from the search results, you can do that in a single call via `doveadm fetch` as shown in Section 14.3.4.

14.3.1. Directory of all possible search keys

You will find the list of all *search keys* available for your version of Dovecot on man page `man 7 doveadm-search-query`; here is an overview of the "search keys" in Dovecot 2.2.4. See Section 14.3.2 for information on the format of `<date>` specifications.

`ALL`
: Matches all messages.

`ANSWERED`
: Matches messages with the IMAP flag \Answered set.

`BCC <pattern>`
: Matches messages which contain pattern in the BCC field of the message's IMAP envelope structure.

`BEFORE <date>`
: Matches messages with an internal date before date specification.

`BCC <pattern>`
: Matches messages which contain pattern in the body part.

`CC <pattern>`
: Matches messages which contain pattern in the CC field of the message's IMAP envelope structure.

`DELETED`
: Matches messages with the IMAP flag \Deleted set

`DRAFT`
: Matches messages with the IMAP flag \Draft set.

`FLAGGED`
: Matches messages with the IMAP flag \Flagged set.

`FROM <pattern>`
: Matches messages which contain pattern in the FROM field of the message's IMAP envelope structure.

`HEADER <headername> <pattern>`
: Matches all messages that contain the matching pattern in the specified RFC 2821 header, for example `HEADER X-Spam yes`. If the `<pattern>` is empty, all emails are returned that contain the header, regardless of the value. Example: `HEADER X-internal`.

`KEYWORD keyword`
: Matches messages with the given IMAP keyword (e. g. $Forwarded) flag set.

`LARGER size`
: Matches messages that are larger than the specified size.

`MAILBOX name`
: Matches messages in the mailbox with the specified name. Wildcards are permitted here, e. g. `mailbox INBOX/project-*`.

`MAILBOX-GUID guid`
: Matches messages in the mailbox with the specified GUID. As we showed earlier, this enables you to further process the search results from `doveadm search`.

`NEW`
: Matches messages which have the IMAP flag `\Recent` set but not the IMAP flag `\Seen`.

`OLD`
: Matches messages which do not have the IMAP flag \Recent set.

`ON <date>`
: Matches messages whose internal date matches the given date specification.

`RECENT`
: Matches messages with the IMAP flag \Recent set.

`SAVEDBEFORE <date>`
: Matches messages which were saved before`<date>` specification.[62]

`SAVEDON <date>`
: Matches messages whose save date matches the given date specification.

`SAVEDSINCE <date>`
: Matches messages with a save date matching or after the given date specification.

`SEEN`
: Matches messages with the IMAP flag \Seen set.

`SENTBEFORE <date>`
: Matches messages with a Date: header before date specification.

`SENTON <date>`
: Matches messages with a Date: header matching the given date specification.

`SENTSINCE <date>`
: Matches messages with a Date: header matching or after the given date specification.

`SINCE <date>`
: Matches messages whose internal date is within or after the given date specification.

`SMALLER <size>`
: Matches messages with a size smaller than the given size.

`SUBJECT <pattern>`
: Matches messages which contain pattern in the SUBJECT field of the message's IMAP envelope structure.

`TEXT <pattern>`
: Matches messages which contain pattern in the body part.

`TO <pattern>`
: Matches messages which contain pattern in the TO field of the message's IMAP envelope structure.

`UID <sequence>`
: Matches messages with the given UID(s). A sequence may be a single UID. Can be a sequence range, written as `from:to`, e.g. 100:125. It's also possible to combine multiple sequences as a comma-separated list of UIDs, e.g. `11,50,4-8`.

`UNANSWERED`
: Matches messages which do not have the IMAP flag \Answered set.

`UNDELETED`
: Matches messages which do not have the IMAP flag \Deleted set.

`UNDRAFT`
: Matches messages which do not have the IMAP flag \Draft set.

`UNFLAGGED`
: Matches messages which do not have the IMAP flag \Flagged set.

`UNKEYWORD <keyword>`
: Matches messages which do not have the given IMAP keyword flag set.

`UNSEEN`
: Matches messages which do not have the IMAP flag \Seen set.

You can also negate or combine criteria:

`NOT search <key>`

Inverse matching - matches messages where the search doesn't match the specified search key or its value. `NOT SEEN` has the same meaning as `UNSEEN` in this case.

`search <key> OR search <key>`

Multiple criteria are usually linked with a logical AND; `OR` logically provides an OR connection.[63] // Note: IMAP4rev1 uses the syntax: OR search key search key

14.3.2. Dates in search criteria

Whenever you enter a date, you can use the following formats:

`day-month-year`
: The default format in line with the RFC for IMAP4rev1.

`day`
: Day of the month, i.e. 1-31.

`month`
: The abbreviated month, i.e. `Jan`, `Feb`, `Mar`, `Apr`, `May`, `Jun`, `Jul`, `Aug`, `Sep`, `Oct`, `Nov` or `Dec`.

`year`
: The year as a four-digit number, e.g. 2013. 13 April 2013 is shown here as 13-Apr-2013.

`interval` Specifies a period; the units can be entered as follows: `w` or `weeks`, `d` or `days`, `h` or `hours`, `m` or `mins`, `s` or `secs`. To search for messages from the week before, you can specify `since 1w`, `since 1weeks` or `since 7days`. `Unix timestamp` The 10-digit Unix timestamp in seconds since 1 January 1970, 00:00:00 UTC. `YYYY-MM-DD` Extended ISO-8601 date format. 13 April 2013 is shown here as `2013-04-13`.

14.3.3. Sizes in search criteria

If you use `doveadm` to search for emails of a specific size, you can use the following two keywords:

`octets`
: Size in octets in line with the RFC for IMAP4rev1. You are unlikely to work with this option in your everyday work.

`size`
: Size of the email in bytes. However, you can also use sizes with `B` (= byte), `k` (= kilobyte), `M` (= megabyte), `G` (= gigabyte) or T (= terabyte), for example `100M` (= 100 MB) or 50K (= 50 KB).

14.3.4. Extracting emails as the result of a search

`doveadm fetch -u <user> <fields> <search query>`
: Works like `doveadm search`, but you receive the required parts of the email and not just the UID and GUID.

Example:

```
flash:~ # doveadm fetch -u peer@example.com text mailbox INBOX subject Pos
```

You can access the following parts of the email directly here:

`user mailbox mailbox-guid seq uid guid flags hdr body text size.physical size.virtual date.received date.sent date.saved imap.envelope imap.body imap.bodystructure` and `pop3.uidl`.

You will find a precise description in `man 7 doveadm-fetch`.

In practice, the important part is usually access to the emails in question, so:

`hdr`
: Email header.

`body`
: Email body.

`text`
: Email header and body – so the entire email in line with RFC 2822. You can then save it, import it somewhere else – or even pipe it to the `sendmail` command and thereby send it somewhere via email.

14.3.5. Moving and copying emails

`doveadm copy|move -u <user> <destination> [user <source user>] <search query>`

The `doveadm copy` and `doveadm move` commands copy or move the emails matching the search pattern to the specified IMAP folder of the user.

This example moves all the emails in the `INBOX` from sender `boss@example.com` to target directory `INBOX/Trash`:

```
flash:~ # doveadm move -u klaus@example.com INBOX/Trash MAILBOX inbox FROM
```

By the way, you can also use the `user <source user>` parameter to copy or move the emails of another user.

This example moves all emails from the `INBOX` of `susi@example.com` to folder `INBOX/SUSI` belonging to user `klaus@example.com`:

```
flash:~ # doveadm move -u klaus@example.com INBOX/SUSI user susi@example.c
```

14.4. Exporting mailboxes to a target directory or synchronizing them with the directory

`doveadm backup [-u <user>|-A] [-dfR] [-l <secs>] [-m <mailbox>] [-n <namespace>] [-x <exclude>] [-s <state>] <dest>`
: extracts the mailbox of a user and writes it to the specified target directory.

As in `mail_location` (see Section 5.9), you specify the target format. `maildir:/home/klaus/backup` saves in the Maildir format, while `mdbox:/home/klaus/backup` uses the `mdbox` format. Example:

```
flash:~ # doveadm -v backup -u klaus@example.com maildir:/backup/example.c
```

By the way, if the target directory already contains email data, `doveadm backup` will perform an incremental synchronization, which will be correspondingly faster. If you have already used this method to create a full backup of one or more mailboxes, repeated run-throughs later on will be child's play.

The following parameters are possible:

`-m <mailbox>`
: limits synchronization to a specific IMAP folder.

`-n <namespace>`
: limits synchronization to a specific namespace (see Section 9.2).

`-R`
: executes the "reverse" action; the file directory is therefore written to the mailbox of the user.

`-l <secs>`
: adapts the timeout for any necessary file locking.

`-x <exclude>`
: excludes individual folders from synchronization; `-m` can also be specified multiple times. *Special-use* mailboxes (see Section 4.4.3) can be listed with a slash at the start. `-m \Trash -m \Junk`, for example, avoids unnecessary ballast in the backup.

`-f`
: In order to conserve resources, Dovecot only synchronizes the meta data for folders that differ in their `UIDVALIDITY`, `UIDNEXT` or `HIGHESTMODSEQ`. That works well and conserves resources, but a small risk remains that parallel changes on both sides can result in changes remaining undetected. Parameter `-f` causes Doveoct to perform a complete synchronization of all folders on both sides.

In older Dovecot versions, the `dsync backup` program was responsible for this task. It is now obsolete, so you should use `doveadm backup` instead as shown here.

`doveadm sync [-u <user>|-A] [-dfR] [-l <secs>] [-m <mailbox>] [-n`

`<namespace>] [-x <exclude>] [-s <state>] <dest>`

While `backup` always maps the state of the mailbox into the file directory, `sync` works in both directions and would transfer changes from the directory back to the user's mailbox. In older Dovecot versions, the `dsync mirror` command was used; it has been replaced with `doveadm sync`.

`import [-u <user>|-A] [-s] <source mail location> <dest parent mailbox> <search query>`
: imports the email data from the specified source directory to a separate IMAP folder structure in the mailbox of the specified user.

If you have used `doveadm backup` or `doveadm sync` to export mailboxes from your IMAP server and store them as files on the backup server, the question is how to restore individual emails or folders from there.

Anyone working in support will be familiar with this common but usually very imprecise request: “Please restore the mailbox from the backup from the day before yesterday.” Simply replacing the user’s email directory with the old copy from the backup is usually not helpful, because that will also delete all the emails that have been received since then.

And because the users' requests are often very imprecise and a lot of back and forth is required until you know *which* emails to restore, a simple recovery can quickly take up a lot of time.

Our support team now imports the data from the user’s previous mailbox as a new subfolder. The existing folder structure is retained, and the user can look for his deleted messages and restore them by means of drag & drop.

In practical application, this command imports *all* emails and IMAP folders of the user to a `backup-2013-10-15` directory structure under the user’s current INBOX:

```
flash:~ # doveadm import -u susi@example.com mdbox:/srv/vmail/example.com/
```

If the user previously had these IMAP folders

```
INBOX
INBOX/Personal
INBOX/Personal/Vacation
```

The new folder structure will look like this:

```
INBOX
INBOX/backup-2013-10-15/INBOX
INBOX/backup-2013-10-15/INBOX/Personal
INBOX/backup-2013-10-15/INBOX/Personal/Vacation
INBOX/Personal
INBOX/Personal/Vacation
```

The user can access all the backed-up data and then delete the whole folder structure once it is no longer required.

There are a few things you should remember:

- Importing a directory can cause the sudden breaching of quotas. That does not affect your import process, but it may ensure that the user can then no longer receive any emails.
- A POP3 user would be unable to see the IMAP folders, but would be able to access his IMAP structure via a webmailer.
- An IMAP email client would start downloading all the imported email data. Bandwidth for desktop PCs is usually wide enough to deal with this, but the results can be quite unpleasant on mobiles or laptops with a mobile uplink (abroad?).

As the backed-up data are usually kept on a remote backup system instead of the same server, the following procedure has proved workable:

1. The Maildir/mdbox directory is restored to the IMAP server from the backup server by means of `rsync` or the backup server; it is placed in the home directory *next to* the existing Maildir/mdbox directory as the `backup` directory.
2. From there, the email data can be imported by means of `doveadm import`.
3. Then the backup directory can be deleted again.

Copy the backup directories to the user home as shown, and not to `/root` or `/tmp`; otherwise the risk is too great that forgotten old directories containing sensitive email data are left lying around in a semi-public and visible manner – or even accidentally imported to another user.

14.6. Repairing an index

It shouldn't happen, but sometimes an index in a user's email storage becomes corrupted. This can occur after a system crash, for example, or if you use a Dovecot index via an NFS share without first configuring the system properly in `10-mail.conf`.

```
Error: Corrupted index cache file /srv/vmail/example.com/user/Maildir/dove
```

In most cases, Dovecot heals itself: if it detects a defect in the index, it triggers a rebuild. That works surprisingly well, and you do not usually need to worry about lost emails (as long as the email files are not damaged as well).

Use the `doveadm force-resync` command to force the rebuild of a folder index:

```
flash:~ # doveadm force-resync -a klaus@example.com INBOX
```

If you want to repair all the folders belonging to a user, enter `*` as the folder pattern; as always, make sure you use inverted commas:

```
flash:~ # doveadm force-resync -a klaus@example.com '*'
```

14.7. References to other topics

We have explained the `doveadm` command `altmove` in Section 7.3.4, and the `quota` command in Section 11.10. For a quick reference guide to all `doveadm` commands, see Appendix A.

---

[61] The size is *virtual* because CR+LF (= 2 bytes) is used as the line separator in line with the IMAP protocol, even though the email may contain only LF (= 1 byte).

[62] Search keys `SAVEDBEFORE`, `SAVEDON` and `SAVEDSINCE` are supported by `doveadm`, but they are not defined by the IMAP protocol itself and can therefore not be addressed via the `SEARCH` IMAP command.

[63] IMAP4rev1 itself uses the syntax `OR search key search key`.

Chapter 15. Performance tuning for more than 1000 simultaneous logins

In a simple IMAP server with a few hundred users, there is not much to do in terms of performance tuning. The default values are sensible, and you are unlikely to encounter a situation where you are forced to take unusual countermeasures.

But the situation is very different if you have many hundreds or even thousands of users:

- Dovecot could run out of file handles; by default, only 1024 are permitted per process in some distributions (use the `ulimit -n` command to see if this is the case).
- Dovecot could run out of processes; by default, only 1024 are provided for in the Dovecot configuration (`default_process_limit`)
- You have to start reducing your consumption of working memory and file handles.

15.1. Increasing the maximum number of clients

So how many processes (and therefore logged-in users) can Dovecot handle?

Dovecot contains some protective mechanisms to prevent errors from incurring numerous processes that could result in a downtime of the entire system.

There are two important settings that set the performance limits – shown here with their default values:

`client_limit=1000`
: Number of clients per process

`process_limit=100`
: Number of possible processes of this type

As the total number is made up of `client_limit * process_limit`, that would mean that 1000 * 100 = 100,000 logged-in users have to be handled in parallel.

However, there are areas where every client has its own process and `client_limit` does not play a role; the actual `pop3` or `imap` module is a case in point. But that ends when the `process_limit` is reached. Check whether you need to increase the `process_limit`, particularly for long-running `imap` processes. You can define this value in a service section such as `service imap {}` in `10-master.conf`:

```
service imap {
[...]
    process_limit = 8192
[...]
}
```

15.2. Reducing login processes

For this chapter, we first need to explain what happens when a user has logged into Dovecot, with IMAP for example. In this case, the two processes `imap-login` and `imap` work hand in hand in Dovecot.

1. The connection is first serviced by `imap-login`, which manages the user's login and terminates the SSL/TLS connection if necessary.
2. As soon as the user is logged in, the `imap` process takes over communications with the client, as it speaks the actual IMAP protocol.

That ensures that the process list contains two processes for every logged-in user. In principle, it is also possible to manage all users in one shared login process and only then launch an individual `imap` process for every logged-in user. After, all, there are pros and cons to having an individual login process for every user.

Pro
: The login process runs with the permissions of the logged-in user. If you have separate user IDs, separate login processes provide a small security benefit, because it is much more difficult for potential attackers to access the context of one user from the login process of another user.

Con
: If many thousands of users are logged in, the large number of individual login processes puts a strain on the system, memory and file handles. Not by much, but these things add up.

As all users now usually run under a shared user ID in current systems (see Section 5.8), the small security advantage no longer applies. So there is almost no reason to stick to this concept – when you consider the additional load on the system, the drawbacks count for more than the advantages.

So there are good reasons to use only one shared login process for all users.

However, `imap-login` (and all other `*-login` processes) also perform the task of terminating any SSL/TLS connections. That means that `imap-login` performs the calculations involved in an encrypted connection – and that explains why `imap-login` can often be found in the list of `top` processes on weaker computers and consumes some CPU time. If we permit only one single `imap-login` process, the entire SSL/TLS computing load would be concentrated on a single process and therefore on a single CPU core.

For this reason, you should have as many login processes as there are CPU cores. Every login process will then handle some of the connections in turn (*round robin*) so the resulting

computing load is distributed evenly among all CPU cores. If there are fewer processes than CPU cores, CPU resources will unnecessarily remain unused. If the number of login processes does not match the number of CPU cores (8 cores, but 12 processes), that simply means that some cores are used twice and others once; as a result, you have either wasted resources unnecessarily or achieved a bad overall result. So the number of processes must match the number of cores.

In contrast, having multiple `login` processes in your Dovecot system (`pop3-login`, `imap-login` and possibly `managesieve-login`) is not a problem. Set the number of CPU cores for `pop3-login` and for `imap-login`. The scheduling system of the Linux kernel will distribute these processes among the available cores in the best possible way. You will rarely have Managesieve login processes, as this service is hardly used, and then only for a very short time. In this case it is usually better to have a separate login process for user logins instead of constantly keeping unused `managesieve-login` processes available to match the number of CPU cores. Either way, the effect in practice is tiny, so your decision is not particularly important.

Modify the settings for `pop3-login` and `imap-login` in `10-master.cf`:

```
service imap-login {
# Do not start new processes when new connections come in...
        service_count=0
# ... but always provide at least as many processes as CPU cores:
# process_min_avail = number of CPU cores
        process_min_avail = 12
}

service pop3-login {
# Do not start new processes when new connections come in...
        service_count=0
# ... but always provide at least as many processes as CPU cores:
# process_min_avail = number of CPU cores
        process_min_avail = 12
}
```

Dovecot will launch with 12 processes and distribute all incoming connections among them until the maximum client limit specified in `client_limit` is reached. That enables 12 processes * 1,000 clients = 12,000 connections. If you exceed this number and all login processes are full, Dovecot will launch additional `pop3-login` or `imap-login` processes in order to deal with the additional connections. That means you can easily serve more then 12,000 clients – because the configuration shown simply specifies that *at least* 12 launched processes exit.

If you know, however, that you have a greater number of simultaneous IMAP logins, it is advisable to set `process_min_avail` to the next multiple of the quantity of CPU cores in

order to achieve an even distribution again. For 12 cores, the value would be `process_min_avail=24`, so up to 24,000 connections can be mapped evenly.

You will find these settings for `managesieve-login` not in `10-master.cf`, but in a separate `20-managesieve.cf`, as Managesieve is an external project and therefore often packaged separately (for example in Debian/Ubuntu). But, as I said before, I actually never modify `managesieve-login` there because it is not really relevant.

Once you have restarted Dovecot, you can easily monitor the effect when the system is running. By the way, this listing also demonstrates nicely that 100% of users are logged in via TSL encryption:

```
flash:~ # ps ax | grep imap-login
29615 ?        S    148:12 dovecot/imap-login [345 connections (345 TLS)]
29616 ?        S    283:42 dovecot/imap-login [285 connections (285 TLS)]
29617 ?        S    217:10 dovecot/imap-login [304 connections (304 TLS)]
29618 ?        S    360:57 dovecot/imap-login [340 connections (340 TLS)]
```

In the past, `default_vsz_limit=64M` was also set here in `master.cf`, so an individual process could consume no more than 64 MB of memory. In systems with many thousands of logged-in clients, that was too little for central `imap-login` processes, so you should make sure the `vsz_limit` is 265 MB – and that is plenty. Current Dovecot versions already have a default setting for `vsz_limit` of 256 MB, so you don't usually need to change anything here:

```
flash:~ # doveconf default_vsz_limit
default_vsz_limit = 256 M
```

15.3. Increasing the file handles

It is not just Dovecot; the Linux system is also familiar with a restriction specifying the maximum resource consumption a process may have. Command `ulimit -a` shows you various values, including some for memory consumption, and above all for the maximum number of open files (`ulimit -n`), typically 1024.

Like in large web server installations with many thousands of domains, the 1024 open file handles usually permitted in load operation are insufficient for an IMAP server with many thousands of simultaneous connections. In this case, you will see the error message `Too many open files` in the email log file.

It is easy and necessary to increase this value for large systems. It is simply a form of protection for the Linux kernel to prevent a program or shell script gone wild from opening countless files in a loop. If you know you need more files, you can easily increase the individual value.

You do not even need to increase this value for the whole system, just for Dovecot. A `ulimit` is inherited to all subsequent processes; if the `ulimit` value is modified in the start/stop script, it also applied to all processes started from there.

You can simply go to the start of `/etc/init.d/dovecot` and enter `ulimit -n 8192` (or `ulimit -n 16384` for even larger setups). However, there is a certain risk that your start script will be overwritten during later updates or that the update will generate a warning.

15.3.1. Debian/Ubuntu

Debian/Ubuntu provides a nicer method: enter the call directly in `/etc/default/dovecot`. These files are executed by the start script by means of . `<file>` (or, more precisely "sourced"), so "included" within the same shell. If you place the command there, it is safe during future updates but will still be executed in the right part of the start script.

```
flash:~ # vi /etc/default/dovecot

# Set to '0' to explicitly disable starting Dovecot
# ENABLED=0

# Set to '1' to allow Dovecot daemons to produce core dumps
#ALLOW_COREDUMPS=1

# Raise ulimit -n (default: 1024)
ulimit -n 8192
```

15.3.2. openSUSE/SLES

SUSE does not provide a separate config file for Dovecot like `/etc/sysconfig/dovecot` where you could enter a `ulimit` call. From openSUSE 12.2 (and therefore from future SLES 12) onward, Dovecot is managed and started via `systemd`, so you can make the necessary intervention here.

First, create file `/etc/systemd/system/dovecot.service.d/limits.conf` with the following syntax:

```
flash:~ # cat /etc/systemd/system/dovecot.service.d/limits.conf
[[[
[Service]
LimitNOFILE=8192
]]]
```

From openSUSE version 13.1, configurations are then complete.

In openSUSE versions 12.2 and 12.3, you also have to create file `/etc/systemd/system/dovecot.service` with the following content:

```
flash:~ # cat /etc/systemd/system/dovecot.service
.include /usr/lib/systemd/system/dovecot.service
.include /etc/systemd/system/dovecot.service.d/limits.conf
```

In older openSUSE versions or SLES 11, which do not yet support `systemd`, you can add command `ulimit -n 8192` at the beginning of start script `/etc/init.d/dovecot`. Alternatively, you can also enter an include to `/etc/sysconfig/dovecot` in the start script and “retrofit” this option.[64]

If not, add the following to the start of `/etc/init.d/dovecot`:

```
[...]
# Reset status of this service
rc_reset


# Include sysconfig-file for Dovecot (e.g.: ulimit!)
. /etc/sysconfig/dovecot
```

Then enter the following in file `/etc/sysconfig/dovecot`:

```
# Raise ulimit -n (default: 1024)
ulimit -n 8192
```

Make sure you remember your modifications when you perform updates; they could overwrite the start script in `/etc/init.d/dovecot` and you would have to repeat your changes after an update.

15.3.3. CentOS/RHEL

CentOS and RHEL already execute an include to the `/etc/sysconfig/dovecot` file in their start script `/etc/init.d/dovecot`. Like in Debian, you can add the `ulimit` command directly to this file:

```
# Here you can specify your dovecot command line options.
#
#OPTIONS=""

# Raise ulimit -n (default: 1024)
ulimit -n 8192
```

15.4. Push email and IDLE

Usually, IMAP clients keep the their connection to the server open permanently. As long as there is nothing to do, they switch the connection to a special mode via the IDLE command. In this time, the server can inform clients of newly arrived emails in real time. It is therefore no longer necessary for the client to permanently query all folders (and generate a corresponding load).

Dovecot uses the mechanisms `dnotify`, `inotify` or `kqueue` for this purpose. It uses them to register files and directories for monitoring with the kernel and is then actively informed by the kernel if changes take place in the monitored paths. This way, Dovecot can also report the arrival of a new email to the client immediately even if it was not delivered via `LMTP` or `dovecot-lda`.

Naturally the number of monitored files and clients to be informed is restricted at first in the kernel. By default, 8192 (Debian) or 65536 (SUSE) different user IDs can register paths; every individual user may monitor no more than 128 different paths at the same time:

```
flash:~ # cat /proc/sys/fs/inotify/max_user_watches
8192
flash:~ # cat /proc/sys/fs/inotify/max_user_instances
128
```

If all IMAP processes run under the same user ID `vmail`, you don't need to worry about the number of `max_user_watches` – because you have just one user ID. And as long as you do not need to supply more simultaneous IMAP connections on average than provided in `max_user_instances`, you do not need to change anything here.

But 128 simultaneous IMAP clients are quickly reached. If your load increases, Dovecot will be unable to register new paths for monitoring and then logs:

```
Warning: Inotify instance limit for user 10000 (UID vmail) exceeded, disab
```

In this case you can considerably increase the permitted quantity:

```
flash:~ # echo 16384 > /proc/sys/fs/inotify/max_user_instances
```

To make the change permanent, enter the following in `/etc/sysctl.conf`:

```
fs.inotify.max_user_instances=16384
```

Under kernel 3.2.0 (the default kernel of Debian 7!) there is a bug that causes the `[fsnotify_mark]` process not to function properly. When a user logs out, his IMAP processes are not ended correctly, but continue to wait for feedback from the system in the `S`

status (*interruptible sleep*). They are only terminated fully by the kernel after a timeout a few minutes later.

The large number of waiting (but inactive) processes causes a high load, but does not require any CPU resources; it simply uses up some working memory, so this is more of a cosmetic problem.[65]

The solution is to install a different kernel version or, as a workaround, to deactivate the Inotify interface:

```
flash:~ # echo 0 > /proc/sys/fs/inotify/max_user_instances
flash:~ # echo 0 > /proc/sys/fs/inotify/max_user_watches
```

Even if Inotify is deactivated, Dovecot will actively inform the client of newly arrived emails. By default, Dovecot scans the email directories of an active user for changes every 30 seconds. In high-load systems, it can make sense to increase the interval slightly in `10-mail.conf`:

```
mailbox_idle_check_interval = 30 secs
```

15.5. Using the write cache in the mail storage

Email systems are traditionally very careful to make sure that emails have been safely written to the file system. After all, you don't want to lose anything. Write operations are therefore usually equipped with `fsync()`, which forces the kernel to write any changes from its own write cache to the hard drive system. If a memory system with battery-buffered hard drive controllers is used, the change reaches the cache of the controller, where it is also kept safely and protected from resets.

But, whether SAN or local hard drive, constant write operations involve a corresponding I/O load for the system. New emails are particularly in danger if they have just been saved in the system and would be lost in the event of a system crash if they have not yet been physically written. The data are in less danger during write operations to emails that have already been received; if a change is lost, the worst case is for a deleted email to reappear, or for the information to be lost that an email was moved from one folder to another.

The safe option is to always use `fsync` – but if you have serious I/O problems and have to make savings in terms of write operations, you can also reduce the work involved in the following way while keeping the risk at an acceptable level:

```
# Never use fsync
mail_fsync = never

# But save newly received emails securely
protocol lda {
  # Enable fsyncing for LDA
  mail_fsync = optimized
}
protocol lmtp {
  # Enable fsyncing for LMTP
  mail_fsync = optimized
}
```

15.6. Single instance storage for attachments

When a large PowerPoint file full of jokey images is sent to a company's large team distribution list, it takes up memory in every mailbox of a user. Single-instance storage for emails is not possible here; but if you use the storage format sdbox or mdbox, you can deduplicate attachments – and they are the reason why the email is so inflated.

Experimental feature

This feature has not yet been tested extensively!

The idea of deduplication is fairly simple: Dovecot saves the attachments of all emails under a hash name in a global directory. Careful, that requires a lot of memory; conversely, you save this space in the actual email directory. If you move to a large-volume but slower storage system, the economic aspects can also be interesting.

If you want to use deduplication, enter a directory path in `10-mail.conf` where the attachments for all users are stored. Also specify the attachment size after which separate storage takes place.

```
# Directory root where to store mail attachments. Disabled, if empty.
mail_attachment_dir = /var/attachment

# Attachments smaller than this aren't saved externally. It's also possibl
# write a plugin to disable saving specific attachments externally.
mail_attachment_min_size = 128k
```

There are three deduplication methods, which you can configure in `10-mail.conf` in parameter `mail_attachment_fs`:

`mail_attachment_fs=posix`
: If you use a SAN storage system which is already capable of deduplication, explicit storage as a separate file is enough to enable SAN to perform deduplication. In this case, Dovecot does not need to do anything else.

`mail_attachment_fs=sis posix`
: If your storage system does not have its own deduplication system, Dovecot can first use a hash of the attachment at the time of saving to determine whether this file could already exist. If there is an identical hash in the attachment directory, the two potential duplicates are matched byte by byte to make sure. Of course that uses some resources at the time of saving.

`mail_attachment_fs=sis-queue /var/attachments/queue:posix`
: In larger systems, you may want to move the work involved in data matching to an off-peak time at night. Dovecot will then first save new attachments to a separate queue

directory without applying any matching mechanisms. Later on, the administrator uses the `doveadm sis deduplicate` command to resolve the queue.

```
flash:~ # doveadm sis deduplicate /var/attachments /var/attachments/qu
```

Dovecot then performs the file comparison, deduplicates any duplicates, and moves the attachments to the final storage location specified in `mail_attachment_dir`.

---

[64] In the future, SUSE may prepare this in Dovecot packages in the `server:mail` repository as standard, so you may find these settings there.

[65] In contrast to popular opinion, the *load* of a system does not show the CPU utilization, but rather the number of processes that are currently waiting. There can be a connection between the two, but there doesn't have to be. If you have 2,000 processes that are waiting for some event, you will naturally quickly end up with a load of 2,000 without your CPU actually having to do any work. IMAP servers with many thousands of IMAP processes will therefore quickly show three or four-digit load figures if a network or hard drive gets stuck, while a system with no more than 50 users logged in at the same time will hardly ever exceed a load of 50, even if there is total chaos. Popular rules of thumb where a system load up to 5 is okay are not based on technical fact. A fully loaded system with a single process will not demonstrate a load above one, even if 100% of its CPU resources are currently busy.

Chapter 16. Load-balancing cluster

The term *cluster* is as common as it is imprecise, and it appears in the requirements catalog of almost every major IMAP project. But what type of clusters are there?

Active/passive cluster

In this case, two computers back one another up as failover. If the active server fails, the second (previously passive) server can take over operations. This form of cluster is about *high availability* and not about load balancing.

Dovecot allows you to implement a classic like that using *Heartbeat*/*Pacemaker*, *CARP* or another method, where email data are distributed among both servers via a shared storage (SAN) or a solution such as DRBD.

Partitioned clusters

Here, every server is responsible for some of the users and is unable to take over other users in the event of a fault. An upstream system (e. g. using suitable IMAP proxies) ensures that users are always transferred to the server that is responsible for them. Rather than a cluster, this is a collection of several independent servers with their own users and emails that are all addressed from the outside in the same way.

The load of users is *shared* strictly, so I refer to such solutions as “partitioned clusters”. Every server has only partial knowledge, so a server would be unable to take over a group of users in the event of a fault. For this reason, this version has nothing to do with high availability (but could be combined with a HA backup).

The *Cyrus murder cluster*, which used to be very well-known, is based on such a setup; every Cyrus system always knew the emails of the users that were assigned to it, but this had nothing to do with system stability/high availability.

You can implement such partitioned setups with Dovecot as well. You do not need any separate IMAP proxies such as `perdition`; instead, Dovecot can assign the users to the correct servers itself. Dovecot contains the necessary proxy modules.

Active/active cluster

Here, two or more systems are active at the same time; user requests can arrive on any of the involved systems, and any of theses systems can respond to the requests. The load is therefore *shared*, and the aim of this setup is *load-sharing*. As every server can accept every user, an active/active setup is always also a high availability solution as long as sufficient reserves are planned to deal with the failure of a system.

With Dovecot, you can implement such a setup on the basis of a cluster file system or on the basis of Dovecot's own email replication. If you like, you can use the Dovecot *Director* to ensure that users are preferably forwarded to the same host every time. Even though this is not necessary in technical terms, it does provide some performance advantages, because it takes better advantage of caching effects.

This is a rough overview. Below I will provide specifics on the individual setups.

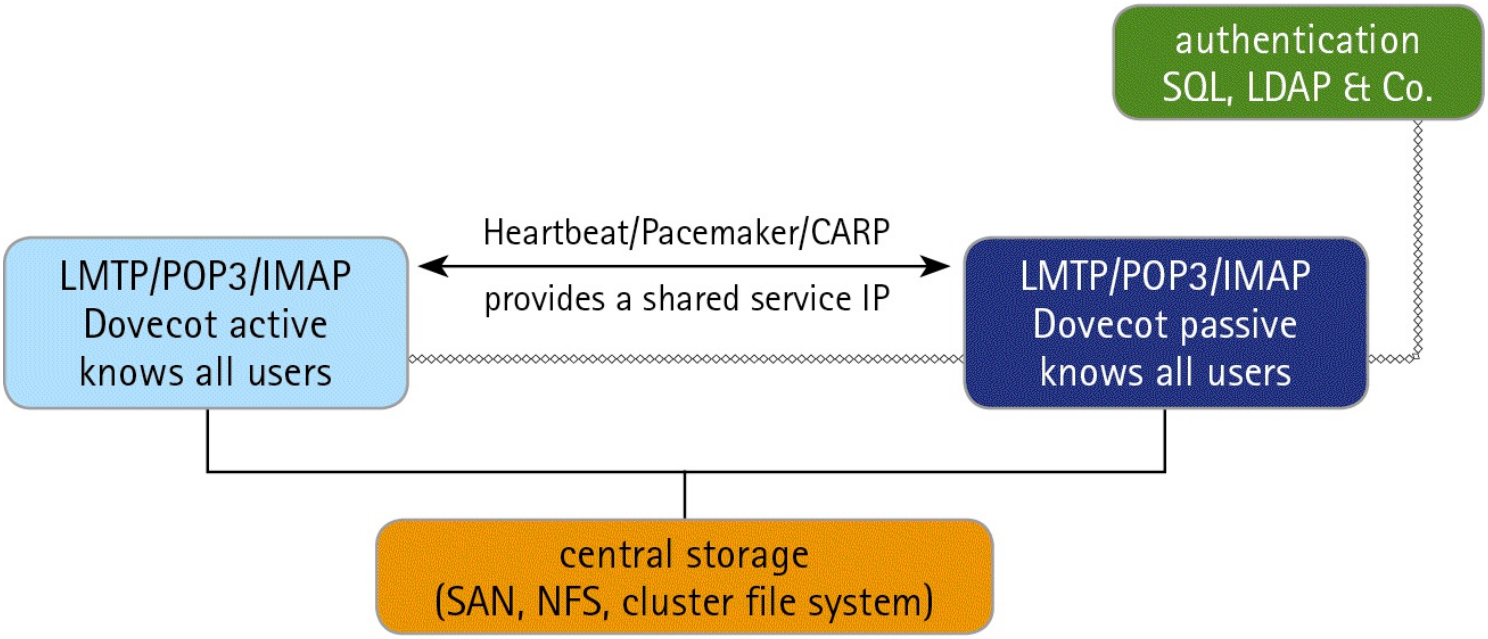
16.1. Active/passive cluster

Active/passive clusters are used to secure against the failure of individual computers – which can be caused by a defect in the hardware or a defect in the operating system. The systems have an individual host IP for administration purposes and also share a *cluster IP* (also referred to as a *service IP*) that is activated in alternating order by the system that is currently active.

As all emails of all users must be available in a failover on the passive system as well, both systems must have access to the same file system:

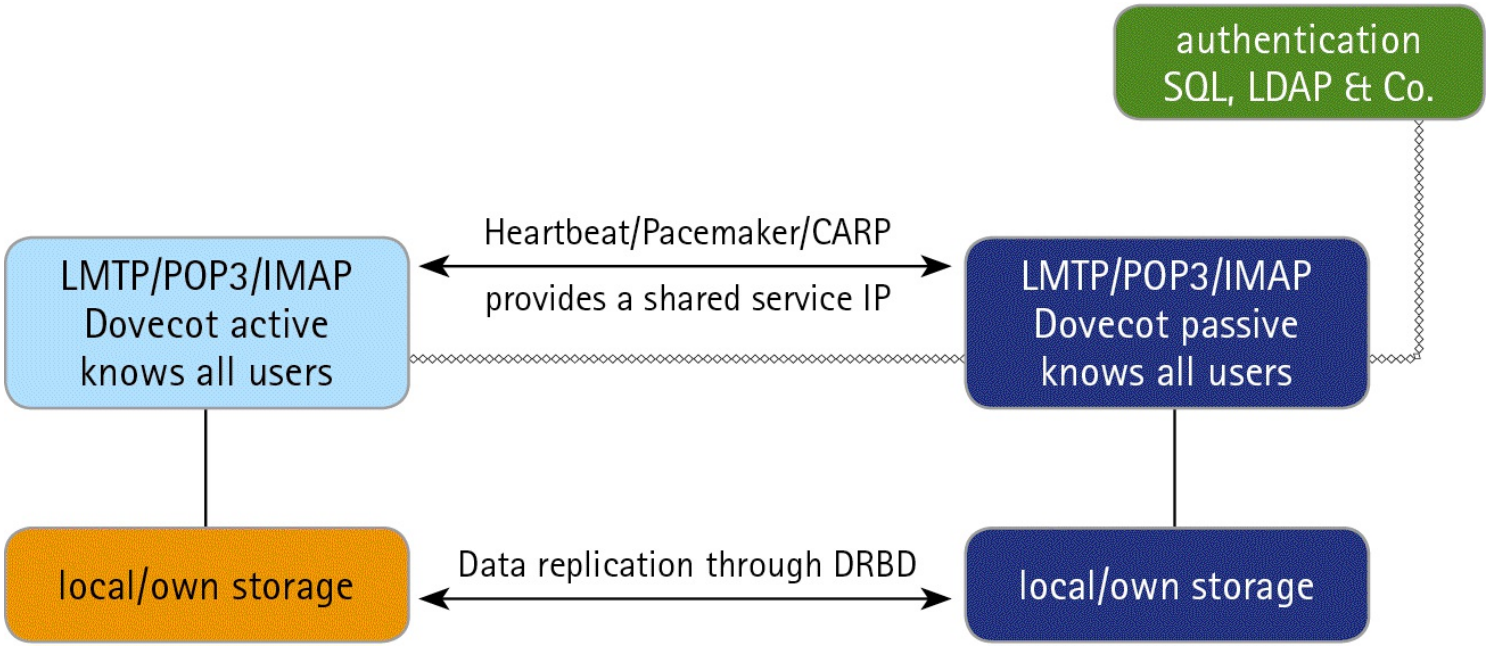
- A SAN-LUN with a normal file system that is visible to both machines, but may of course only be actually mounted on one of the two systems at a time in order to prevent damage to the file system. This solution has the advantage that the volume of emails is stored only once and that the passive machine will definitely have the same data on it as the previously active system in the case of a failover.

Figure 16.1. Active/passive setup with shared storage



- A DRBD partition that connects the hard drives of the two systems. In the end, the data are of course saved in duplicate on the respective local hard drives, which indirectly provides a degree of protection against hard drive defects. Once again, only one of the two systems can have the file system mounted if a normal file system is in use.

Figure 16.2. Active/Passive setup with DRBD replication



- If you want to set up an active-passive setup on the basis of NFS, you are actually setting up an active/active solution, even if you are not actively addressing one of these services. Make sure to read the explanations and warnings against NFS in Section 16.3.

In active/passive setups you need to pay special attention to split-brain situations, as simultaneous access to one file system can result in the loss of all data.

Active/passive setups still have a residual risk of total failure – for example if the file system containing the email data is itself destroyed and both systems therefore have no working access to the pool of emails. That can happen quite quickly: broken index files, logical errors in the file system or a simple administration error (`rm -rf *`) will quickly ensure that the service is no longer offered.

You can find exhaustive instructions on setting up active/passive clusters with *Heartbeat*, *Pacemaker* or *CARP* and on what to bear in mind in many places, and the whole thing has little to do with Dovecot, so I will not provide a concrete installation guide here.

16.2. Partitioned clusters with Dovecot proxy

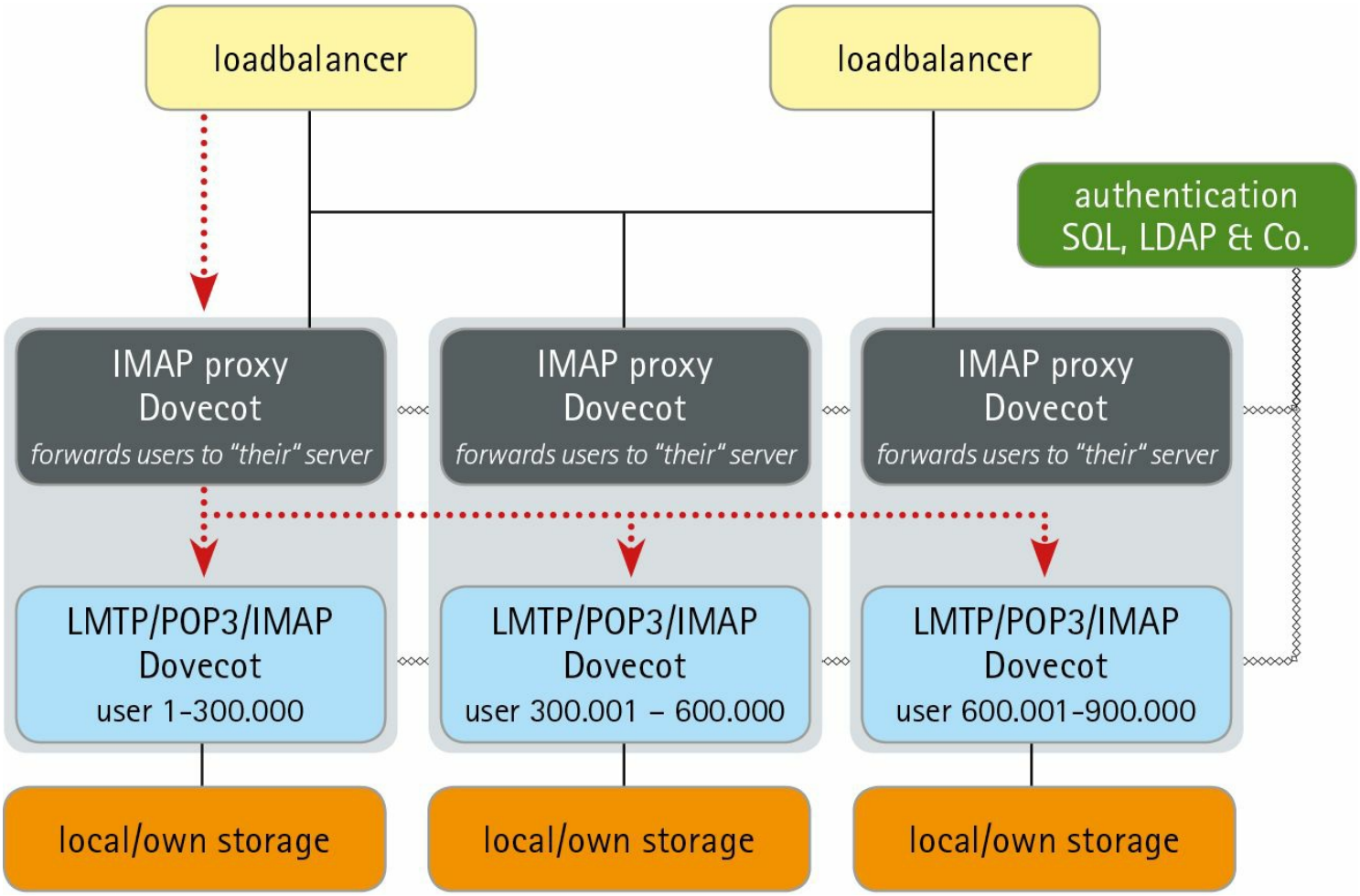
In most very large setups, not every server has access to all emails – simply because the large number of users and size of the file storage require a sensible distribution among multiple individual servers. In the past, IMAP proxies like *Perdition* were used for this purpose, which transfer the IMAP sessions of the different users to the correct target host on the basis of host entries in the authentication backend.[66]

With Dovecot, you do not need to use any additional IMAP proxies. Dovecot modules `imap`, `pop3` and `lmtp` perform the task perfectly well themselves. They use individually set *Passdb extra fields* in a user's authentication data (see Section 5.12) to assign every user to an individual server to which all POP3, IMAP and LMTP connections relating to that user are then transferred.

For this purpose, attributes `proxy_maybe` or `proxy` must be set in the user's Passdb extra fields.

`proxy_maybe=yes`
: Dovecot checks whether Dovecot itself is listed in the user's `host` attribute. If that is the case, the connection is served locally as usual. If a different server is listed, the POP3/IMAP/LMTP connection is forwarded to this host as a TCP/IP connection. `proxy_maybe` therefore allows Dovecot to be the proxy server and the backend server at the same time ("automatic proxying") – see Figure 16.3.

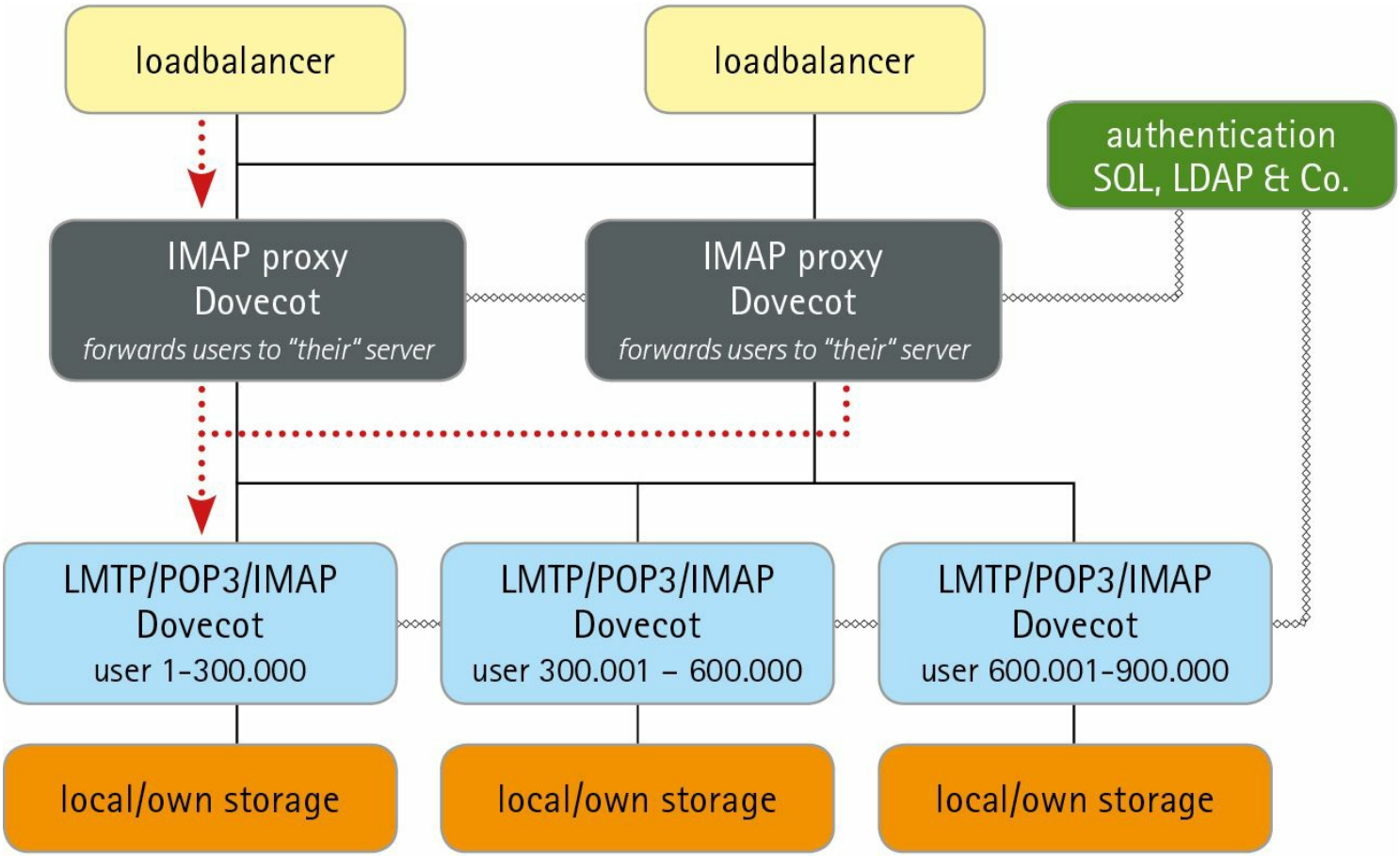
Figure 16.3. Partitioned cluster – the proxy and IMAP servers can run on the same server if `proxy_maybe` is used



`proxy=yes`

In this case, Dovecot always forwards the connection to the server listed in the `host` attribute. If the Dovecot server is specified there, this would lead to an endless loop, and it would be impossible to log in. The `proxy` attribute is therefore used in systems that have dedicated upstream proxy servers and do not work as a backend at the same time – see Figure 16.4.

Figure 16.4. An active/active cluster with partitioned IMAP servers, each of which knows only some of the users. The proxy forwards the user to the server who knows the user.



The actual target server of the user is specified in the Passdb extra fields by means of a `host` attribute:

`host=<ip-address>`
: The IP address of the target server. From Dovecot version 2.1.2 onwards, a host name can also be entered here.[67]

Optionally it is also possible to modify the port number and login name, and to set a short timeout:

`port=<port>`
: the TCP/IP port on the target server if values other than the default values 110 (POP3) or 143 (IMAP) are to be used.

`destuser=<username>`
: uses the user name defined here to log in to the target server.

`proxy_timeout=<timeout>`
: timeout in seconds – terminates if the connection to the target host cannot be created in time.

Sometimes it may be useful to influence the SSL behavior between the proxy and the backend, for example if the backend is not in the LAN and instead available only publicly via the internet:

`ssl=yes`
: instructs Dovecot to connect the backend via SSL and to verify the certificate of the backend. However, this is only safe against man-in-the-middle attacks if only the own CA is trusted.

`ssl=any-cert`
: uses SSL, whether the certificate is valid or not.

`starttls=yes`
: similar to `ssl`, but uses the STARTTLS command on port 110/143.

`starttls=any-cert`
: similar to `ssl=any-cert`: uses STARTTLS but does not verify the validity of the certificate.

An entry in a passwd file in Dovecot could look something like this:

```
klaus@example.com:{PLAIN}test:10000:10000::/srv/vmail/example.com/klaus::
host=10.152.188.113 proxy_maybe=yes destuser=klaus
```

You will find examples for LDAP or SQL in Section 5.12.

LMPT connections can also be forwarded to the user's target host so that emails are saved on the correct server. In this case, there is no need to keep a routing table for users' emails in the upstream MTA (Postfix). Postfix can deliver to any Dovecot server in the cluster and leave the rest up to Dovecot.

However, the Dovecot LMTP daemon must be able to perform an anonymous(!) `passdb` lookup for this purpose in order to read out the Passdb extra fields (for details, see Section 5.5). LMTP proxying therefore does not work with PAM modules or in particularly LDAP setups with particularly restrictive settings. It is, however, easily possible on the basis of the passwd file, normal AD/LDAP setups or SQL.

Proxy connections for LMTP first need to be activated; to do so, make a separate entry in `20-lmtp.conf`:

```
# Support proxying to other LMTP/SMTP servers by performing passdb lookups
lmtp_proxy = yes
```

In the backends(!) you should list the IP addresses of the proxy servers in `login_trusted_networks` so the proxy can transmit the client's original IP address to the backend. This also enables the IMAP proxies to perform a plain-text login without SSL if you decide not to use SSL/TLS between the proxy and the backend:

```
# IMAP proxy connections come from this network:
login_trusted_networks = 10.152.188.0/24
```

If your proxy setup is active, you can trace the proxy forwarding in the log file after a test login:

```
imap-login: proxy(klaus@example.com): started proxying to 10.152.188.113:
```

You can also use the `doveadm proxy` command to view the users currently being forwarded and are then able to terminate their connections:

```
flash:~ # doveadm proxy list
username                proto src ip     dest ip         port
susi@example.com        imap  10.2.5.48  192.168.50.161 143

flash:~ # doveadm proxy kick susi@example.com
1 connections kicked
```

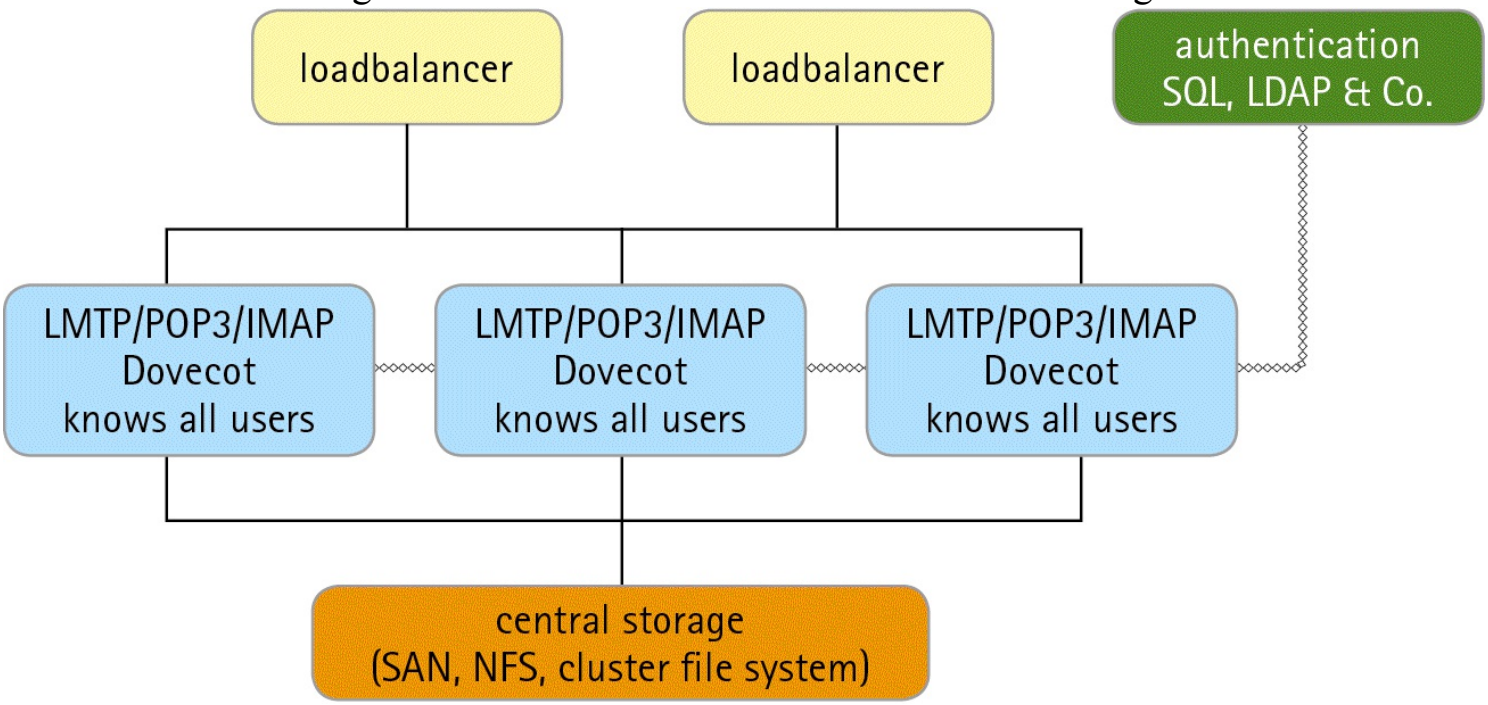
16.3. Active/active cluster with Dovecot Director

A classic active/active setup requires that every system has full read and write access to the file system containing the user data. Therefore a classic active/active cluster requires the use of a true cluster file system or a central NFS storage (I have described an active/active cluster on the basis of replication in Section 16.4).

However, in my opinion active/active setups with shared storage are fairly problematic:

- Cluster file systems can be a great solution, but the administrator needs a lot of specialist knowledge. They have their own potential failures and difficulties that also need to be mastered. Cluster file systems are great at making sure that a file system can be used actively in several systems in parallel, so they ensure *availability in multiple locations at the same time*. But that does not mean that cluster file systems are automatically faster and increase performance. Done well, they improve performance, but they will conversely reduce performance if they are done badly.
- NFS offers parallel access for multiple servers to one single file system with little effort, and can be defined as a cluster file system in this regard. NFS is not as bad as generally believed; with good NFS implementations, you can achieve very good access rates. A local file system will always be faster, so NFS should be considered as a method for broad scaling (multiple servers), but not as an upscaling measure (greater I/O throughput). I/O performance is usually the bottleneck, so the question is why you would even want to do broad scaling. Or, to put it another way: broad scaling can also be counter-productive here.

Figure 16.5. Active/active cluster with shared storage



Even if the Maildir format was originally designed expressly for use via NFS, you must make adjustments to Dovecot if you want to use a cluster file system or NFS; you will find the necessary settings in `10-mail.conf` . Dovecot is highly optimized in terms of its file access and uses techniques such as 'memory mapping '(`mmap`) or write caches to increase throughput. For NFS at least, you have to deactivate nearly all these optimizations, because they are incompatible with NFS or (in the case of delayed writes, for example) cause inconsistencies in the cluster because the different systems in the cluster access outdated data, which can lead to chaos and even corrupt mailboxes. If you use the Dovecot Director, make sure that the same user is always connected to the same target server (more on this in Section 16.3.1). That avoids many caching problems, but it does not prevent them completely.

If you do not use the Director, and/or if you have to assume that a user is logged in simultaneously in different systems, you have to avoid caching problems with index and email files. Deactivate the optimizations in the file access and make sure you configure Dovecot in `10-mail.conf` like this:

```
mmap_disable = yes
dotlock_use_excl = no # only needed with NFSv2, NFSv3+ supports O_EXCL and
mail_fsync = always
mail_nfs_storage = yes
mail_nfs_index = yes
```

Let’s be clear: this deactivates the thing that makes Dovecot so quick. Using NFS instead of a local file system will never improve performance. Quite the opposite: Dovecot now travels with its handbrake on. Even though I am fond of NFS, I avoid using it on mail servers whenever possible.

I know plenty of setups where the reasons for continuing problems could always be found in NFS (and evaporated after migration to a local file system). Timo Sirainen also provides explicit warning in his NFS readme that problems involved in the use of NFS can never be entirely prevented.[68]

If you use an active/active cluster to improve performance, you should first check whether the bottleneck is to do with the server hardware (where you can widen the scale) or with the I/O performance of your file system (where a cluster file system or NFS with handbrake would be counter-productive).

Cluster file systems are not always unsuitable for use on mail servers, as there are plenty of different solutions with very different properties. If you have the necessary know-how in your team and also experience from other operating scenarios, a cluster file system can be a solution. For normal scenarios and “normal” administrators however, the use of a cluster file system should be viewed very sceptically.

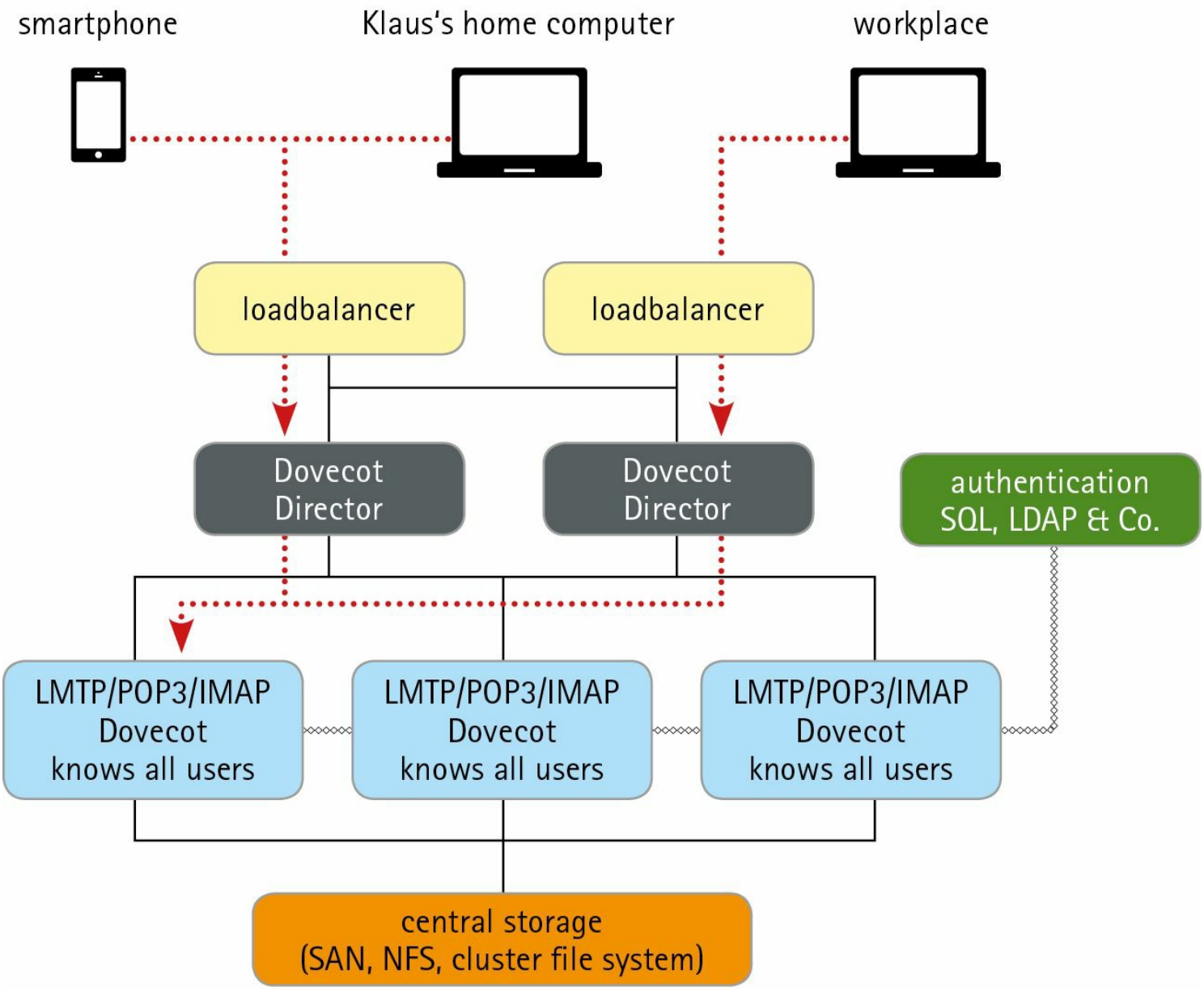
16.3.1. The Dovecot Director

Caching is everything. Even if the active/active system ensures that users are served by every system, it makes sense to bundle all IMAP connections of a user on one server. This creates the best possible reading cache and also avoids potential problems during the writing of files.

A normal loadbalancer would only be able to direct multiple connections from one single client IP address to one single target system by preference (*sticky connections* or *persistent connections*). In practice, however, a user will often log into his account from multiple IP addresses simultaneously: desktop PCs, laptops with WLAN or UMTS, and smartphones all access the identical user name from entirely different networks. A layer-7 proxy is required that analyzes login names and forwards additional connections to a user name preferentially to the system on which this login name is already currently logged in.

The Dovecot *Director* service will perform this task after a few configuration steps. You will find the necessary settings in `/etc/dovecot/conf.d/10-director.conf`. The system consists of one or more director servers that work upstream of multiple IMAP backends. The director servers consult each other via a separate protocol and thereby share knowledge about the backend server a user is already connected to. For this reason, a director server must also know the IP addresses of the other director servers. Figure 16.6 illustrates the scenario.[69]

Figure 16.6. Active/active cluster with Dovecot Director bundles a user's logins



The Director service is an add-on to the general proxy functionality as described in Section 16.2 (partitioned clusters). While the `proxy=yes` parameter would there be used to connect to the server specified in `host=`, `proxy=yes` is used here without specification of the `host=`. Instead, a list of the IP addresses of the email backends is stored in `director_mail_servers`. The director service then dynamically selects the most suitable backend server and dynamically sets the host information at the end of the authentication process. It thereby ensures that the proxy service is connected to the right backend.

In order to activate the Director, you first need to set up the proxy functionality, but you can go without the `host=` parameter in the Passdb extra fields. Then provide Dovecot with a list of director and IMAP servers and activate `inet_listener` for the director server (here: port 9090) so that the director servers can communicate. You will find everything prepared in `10-director.conf`:

```
director_servers = 192.168.3.10:9090 192.168.3.11:9090
```

```
director_mail_servers = 192.168.50.161 192.168.50.162 192.168.50.163

service director {
  unix_listener login/director {
    mode = 0666
  }

  inet_listener {
    port = 9090
  }
}
```

Then activate the director function for IMAP, POP3 and LMPT in `10-director.conf`:

```
service imap-login {
  executable = imap-login director
}
service pop3-login {
  executable = pop3-login director
}

# Enable director for LMTP proxying:
protocol lmtp {
  auth_socket_path = director-userdb
}
```

If you now restart Dovecot, you should see the open director ports:

```
flash:~ # lsof -i :9090
COMMAND   PID USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
dovecot 10477 root   44u  IPv4 644358      0t0  TCP *:9090 (LISTEN)
dovecot 10477 root   45u  IPv6 644359      0t0  TCP *:9090 (LISTEN)
```

If everything is working properly, you should find the login of an example user in the log file of a director server like this:

```
Aug 13 16:00:36 flash dovecot: imap-login: proxy(klaus@example.com): start
```

Query the status of your director cluster by means of the `doveadm director ring status` command:

```
flash:~ # doveadm director ring status
director ip    port type last failed
192.168.50.161 9090 self never
192.168.50.162 9090 l+r  never
```

To view the status of the connected mail servers and the current user numbers, use the `doveadm director status` command:

```
flash:~ # doveadm director status
mail server ip vhosts        users
192.168.50.161    100         5427
192.168.50.162    100         5877
```

First, every mail server is given a weighting of 100 points in the distribution, which means that all servers are addressed equally. If, however, a server is stronger or weaker, you can use the `doveadm director add` command to give it a different weighting. The greater the weighting, the more connections it is assigned:

```
flash:~ # doveadm director add 192.168.50.161 150
flash:~ # doveadm director status
mail server ip vhosts         users
192.168.50.161    150          7203
192.168.50.162    100          4622
```

You can also determine the status of a specific user in this way:

```
flash:~ # doveadm director status klaus@example.com
Current: not assigned
Hashed: 192.168.50.161
Initial config: 192.168.50.161
```

In this case, `klaus@example.com` is currently not connected to the system; however, if he logged in now, he would be assigned the server with IP 192.168.50.161.

16.3.2. Director management during runtime

Normally you should simply leave the management of the mail servers and the associated users to the Dovecot Director itself. However, there are some situations where you may need to reconfigure or check your Dovecot installation during runtime:

- When new mail servers are added to the cluster, you want to add them during runtime without a server restart.
- When a node is removed from the cluster for maintenance purposes.
- When you optimize the distribution of users among the different mail servers by means of fine tuning.

You control your director ring using the corresponding `doveadm` commands. Be aware that these settings are not entered in your configuration files. When you restart your Dovecot cluster, information stored in this manner is gone. Enter new or removed mail servers in the configuration files in `director_mail_servers` as well as in `doveadm`, or save your dynamically determined configuration with the `doveadm dump` command.

The following commands are available to you via `doveadm director`:

`doveadm director add <host> [vhost_count]`
: introduces a new mail server `<host>` and has it managed by the director. Optionally you can specify the weighting that you want the server to have in future. In order to change the weighting of a server that is already active, execute the `add` command for that server again.

`doveadm director flush <host>|all`
: deletes the user associations that are assigned to a mail server. The affected users are redistributed the next time they make a connection. This can be useful for test purposes. `flush all` deletes the associations of all users and all mail servers.

`doveadm director map [host]`
: shows which users are assigned to which mail server. If a specific host is specified, only the mappings for that host are listed.

  Optionally, you can use `-f users_file` to transfer a file with user name (line by line). That way, you avoid the query of all users from putting a strain on the database or the LDAP server.

`doveadm director remove <host>`
: deletes the specified server from the director administration (but does not change its entry in the Dovecot configuration files).

`doveadm director dump`
: lists the current configuration of the director in the form of `doveadm` commands. It is a way to save the mail servers and weightings involved – which can then also be restored using `doveadm` commands.

`doveadm director status [user]`
: shows the utilization statistics for all active mail servers. If a user name is added as an

option, you can query where it is currently assigned, where it will be assigned the next time it connects, and where it would be assigned if the director cluster was restarted.

16.4. Active/active cluster with real-time replication

Dovecot is now capable of replicating the pool of emails between different nodes in real time. That has huge advantages, because it allows an active/active cluster to be set up in just a few steps; the cluster will be able to do without a cluster file system or NFS thanks to data replication, and also provides good protection against loss of data.

During every replication at email level:

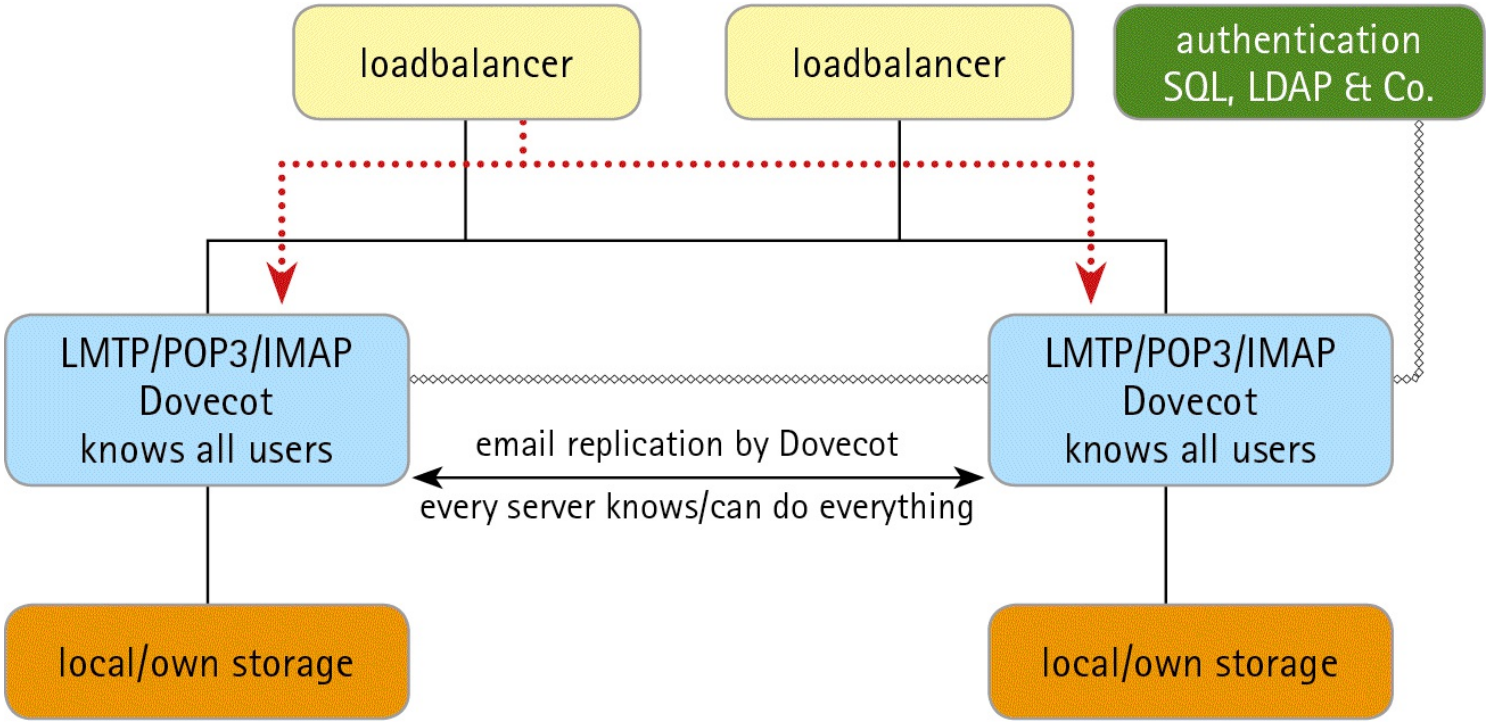
- every host has its own file system where it stores emails – and it is possible to use different file systems such as xfs and ext4 on every node. Unlike in a DRBD replication or a shared LUN or a SAN, the consistency of a file system is not a single point of failure. A file system check of longer duration can be performed in sequence on one server at a time during runtime (even if the remaining node should not fail during this time).
- the split-brain problem of an active/passive setup no longer creates any problems.
- the cluster copes well even with high latencies, and can therefore be spread over different locations.
- a new node can be added later on, because no special preparation is required in the file system.

Do make sure to use a version higher than 2.2.6, because incremental resource-conserving replication is not possible before.

Dovecot's procedure is actually fairly simple:

With the aid of `doveadm backup` and `doveadm sync` (previously `dsync`), Dovecot is already able to synchronize two different email directories efficiently and securely in both directions, i.e. to create an identical state on both sides (see Section 14.4). It used to be a good solution to have corresponding `sync` runs performed via cron job from time to time in order to keep two systems in an almost consistent state.

Figure 16.7. Active/active cluster with Dovecot replication



Today Dovecot can replicate all changes to the mail storages independently via the dsync protocol as an action to its partners. However, it does need to know which user accounts it is to replicate, to ensure that the mailboxes are fully synchronized if in doubt. A `userdb` query must therefore be able to return a list of all accounts (see Section 5.13).

You can determine whether that is the case by using the `doveadm user '*'` command (observe the inverted commas or quotation marks around the asterisk!):

```
flash:~ # doveadm user '*'
klaus@example.com
susi@example.com
micha@example.com
```

Unusually, there are no prepared passages in the Dovecot configurations for the replication settings shown below. Instead of distributing the modifications below among `10-mail.conf` and `10-master.conf`, it is probably less confusing to create these settings in a new file, `12-replication.conf`. Enter the configurations shown in file `12-replication.conf`:

```
# load replication plugins
mail_plugins = $mail_plugins notify replication
```

Set up an `aggregator` service so the email processes report the changes to the pool of emails to the replicator service. The POP3/IMAP/LMTP email processes of an individual (unprivileged) user must be able to access the FIFO and `unix_listener` of the aggregator service. Assign appropriately generous `666` access permissions or – if, as recommended, you handle everything from one central `vmail` user ID – keep the restrictive file permissions and let Dovecot open the sockets under the `vmail` ID:

```
service aggregator {
  fifo_listener replication-notify-fifo {
    user = vmail
#    mode=0666
  }
  unix_listener replication-notify {
    user = vmail
#    mode=0666
  }
}
```

Now activate the permanently running replicator service:

```
service replicator {
  process_min_avail = 1
  unix_listener replicator-doveadm {
    mode = 0660
    user = vmail
    group = vmail
  }
}
```

The actual replication to the partner can be performed in different ways: TCP connects between two `doveadm` instances or SSH logins that are used to execute `doveadm` commands.

16.4.1. Replication on the basis of doveadm via TCP/IP

To do so activate on every involved server a TCP/IP socket via which the `doveadm` server can be reached:

```
service doveadm {
  inet_listener {
    port = 12345
  }
}

doveadm_port = 12345
doveadm_password = secret
```

Finally, tell the replication service how to replicate to its partner:

```
plugin {
  mail_replica = tcp:192.168.50.161
}
```

If you now specify the target directory in `doveadm backup` or `doveadm sync` (or, in older versions, `dsync`), you can specify the target `tcp:hostname` as well as `mdbox:`, `maildir:` oder `mbox:`.

16.4.2. Replication of a public namespace

Replication of a public namespace (see Section 9.6) does not work satisfactorily at the moment for design reasons. It results in errors in the log file and duplicated emails. If you use a public namespace and Dovecot from 2.2.9 onwards, you should first exclude this from the global replication by modifying the `dsync` parameter used during replication.

Use `12-replication.conf` to delete the default parameter `-N` (all namespaces) from `replication_dsync_parameters`:

```
# replication_dsync_parameters = -d -N -l 30 -U
replication_dsync_parameters = -d -l 30 -U
```

You don't want the public namespace to remain unreplicated, so you should set up a cron job that uses the `-N` parameter to replicate a dummy user with the public namespace from time to time.

```
flash:~ # doveadm sync -u dummyuser@example.com -d -l 30 -N
```

16.4.3. Securing via SSL

If you like, you can also secure the `doveadm` connection via SSL. Activate SSL for the listener and add `ssl=yes`:

```
service doveadm {
  inet_listener {
    port = 12345
    ssl = yes
  }
}
```

Like in any SSL/TLS-secured connection, the client should check the authenticity of the server certificate using an appropriate certificate, i.e. a signature from a familiar certification authority (see Section 10.1).

Certificates can be collected in a directory, as is the case in Debian, for example:

```
ssl_client_ca_dir = /etc/ssl/certs # Debian/Ubuntu/SUSE
```

You can also transfer a `pem` file to Dovecot that contains one or all root certificates (RedHat):

```
ssl_client_ca_file = /etc/pki/tls/cert.pem # RedHat
```

Instead of `tcp:hostname`, simply enter `tcps:hostname` in order to switch to the encrypted version.

But be careful. If you use certificates you have signed yourself, you must enter the certificate of your own CA there as well.

16.4.4. Replication on the basis of SSH logins

Replication via SSL logins is also possible; in that case, the `doveadm` command is called up via the SSH connection on the opposite side. An SSH key for login (here: `vmail`) must be stored so that replication can function automatically:

```
dsync_remote_cmd = ssh -l%{login} %{host} doveadm dsync-server -u%u
plugin {
  mail_replica = remote:vmail@otherhost.example.com
}
```

Personally, I would advise you to choose the path described above: using a simple TCP connection between the two `doveadm` instances. It is simpler, and you do not need to enable SSH login for your relatively important `vmail` user.

16.4.5. Monitoring and maintenance with doveadm replicator

If you have activated the `replication` plugin, `doveadm` will also be familiar with the `replicator` command:

`doveadm replicator status`
: returns a general status on the replication queue:

```
Queued 'sync' requests        0
Queued 'high' requests        0
Queued 'low' requests         0
Queued 'failed' requests      0
Queued 'full resync' requests 0
Waiting 'failed' requests     2
Total number of known users   2
```

`doveadm replicator status '*'`
: returns a concrete status for the specified user or for all users that fit the specified pattern. The times specify how long ago the respective replication took place.

```
username                priority fast sync full sync failed
susi@example.com        none     00:01:20  00:07:52  y
klaus@example.com       none     00:01:20  00:07:52  y
```

`doveadm replicator replicate '*'`
: triggers a replication for all users who fit the specified pattern.

`doveadm replicator remove test@example.com`
: blocks the specified user for replication. In order to include a user in replication again, trigger the `doveadm replicator replicate <username>` command once.

Dovecot handles replication while the system is running. You do not need to call up `doveadm replicator replicate` regularly. However, after major restructuring or the first initialization, you should use it to ensure that the basic data pool is identical on both sides.

16.4.6. Other things you need to consider

Dovecot now replicates emails, but it maintains its own index on every system. Make sure

- the mail servers do not use a shared quota database (e.g. the same SQL database), otherwise everything will be counted twice
- you perform a separate `doveadm purge` command on all servers for each cron job if you have mdbox
- actions such as `doveadm force-resync` or `doveadm quota recalc` and similar are performed individually on every server.
- every host in the cluster has its own host name. From Dovecot 2.2.6, you can verify that by means of `dovecot --hostdomain`.

---

[66] Do not confuse this with IMAP proxies such as *up-imapproxy*, which caches connections. Perdition is more of a "load balancer" that distributes the queries among the target hosts but does not optimize and reduce the load or queries themselves.

[67] If a user has a `host` attribute but no `proxy` or `proxy_maybe` attribute, Dovecot sens the mail client a *login referral* in line with RFC 2221. The client is logged in locally on the server without the TCP connection being forwarded to the target host, but the IMAP client is "advised" to perform a login itself on the specified server. Login referrals are fairly exotic and supported only by Pine and Outlook.

[68] http://wiki2.dovecot.org/NFS

[69] If you run Dovecot in a multi-instance setup, the director and the IMAP backend can run on the same machine. `doveadm` helps you manage multi-instance setups, but that goes beyond the scope here; `man doveadm-instance` is your first port of call.

Chapter 17. Autoconfig and Autodiscover – automatic configuration of email clients

Mechanisms established since 2012 ensure that, when setting up an account, an email client can only find the correct server names and SSL/TLS settings directly by means of the email address. This is achieved by fairly simple means: following a specific schema, the attempt is made to load a configuration file from a web space in the domain of the email address that contains the required settings.

So far, so simple. But in spite of this good and simple idea, well-known email clients have still found their own ways to resolve this task…

17.1. Autoconfig with Thunderbird

From version 3.0, Thunderbird searches for URL `http://autoconfig.<emaildomain>/mail/config-v1.1.xml` when setting up new email accounts.

If you set up an account called `klaus@example.com`, Thunderbird will search in `http://autoconfig.example.com/mail/config-v1.1.xml` for an XML-based configuration file that will provide the server names and other configuration details for this email address.

In the example below we have used the provider `mailbox.org`. Here, the XML file below configures the client

- on standard port 143 (or 110) for IMAP (or POP3) with TLS, with 995 and 993 used only as alternatives
- on submission port 587 for SMTP with TLS, with smtps (465) used only as an alternative.
- for secure password procedures (CRAM, see Section 5.10)
- and with a login name that matches the email address.

```
<?xml version="1.0" encoding="UTF-8"?>

<clientConfig version="1.1">
  <emailProvider id="mailbox.org">
    <domain>mailbox.org</domain>
    <displayName>mailbox.org - to keep the personal private</displayName>
    <displayShortName>mailbox.org</displayShortName>

    <incomingServer type="imap">
      <hostname>mail.mailbox.org</hostname>
      <port>143</port>
      <socketType>STARTTLS</socketType>
      <authentication>password-encrypted</authentication>
      <username>%EMAILADDRESS%</username>
    </incomingServer>

    <incomingServer type="imap">
      <hostname>mail.mailbox.org</hostname>
      <port>993</port>
      <socketType>SSL</socketType>
      <authentication>password-encrypted</authentication>
      <username>%EMAILADDRESS%</username>
    </incomingServer>

    <incomingServer type="pop3">
      <hostname>mail.mailbox.org</hostname>
      <port>110</port>
      <socketType>STARTTLS</socketType>
```

```
      <authentication>password-encrypted</authentication>
      <username>%EMAILADDRESS%</username>
    </incomingServer>

    <incomingServer type="pop3">
      <hostname>mail.mailbox.org</hostname>
      <port>995</port>
      <socketType>SSL</socketType>
      <authentication>password-encrypted</authentication>
      <username>%EMAILADDRESS%</username>
    </incomingServer>

    <outgoingServer type="smtp">
      <hostname>mail.mailbox.org</hostname>
      <port>587</port>
      <socketType>STARTTLS</socketType>
      <authentication>password-encrypted</authentication>
      <username>%EMAILADDRESS%</username>
    </outgoingServer>

    <outgoingServer type="smtp">
      <hostname>mail.mailbox.org</hostname>
      <port>25</port>
      <socketType>STARTTLS</socketType>
      <authentication>password-encrypted</authentication>
      <username>%EMAILADDRESS%</username>
    </outgoingServer>

    <documentation url="http://www.mailbox.org/faq/">
      <descr lang="de">FAQ and support database</descr>
      <descr lang="en">Frequently asked questions (FAQ)</descr>
    </documentation>
  </emailProvider>
</clientConfig>
```

The example should be self-explanatory. When searching for the right configuration, the client prefers the section it finds first if there are alternative options. If you want your users to use POP3 rather than IMAP, list the POP3 section first; if you prefer them to use IMAP, list the IMAP section first. You can leave out the sections relating to the POP3 and IMAP versions on ports 993 and 995 if you like; nowadays there is no reason to operate clients directly on the standard ports with STARTTLS.

In section `<authentication>`, specify how the password is transmitted:

`password-cleartext`
: allows plain text login (PLAIN, LOGIN)

`password-encrypted`
: allows only secure password methods, i.e. `CRAM` (see Section 5.10)

An overview of the most important placeholders:[70]

`%EMAILADRESSE%`

the email address for which Thunderbird is currently being configured

`%EMAILLOCALPART%`
: only the local part of the email address, for example as the login name

`%EMAILDOMAIN%`
: the domain of the email address, for example to have a host name such as `imap.%EMAILDOMAIN%` generated automatically

If your login name is completely unlike your email address, you will not get far with a static XML file, for example if the email address and the Windows login name are not the same. In this case, you have to hide a piece of software behind the URL that submits a database query matching the email address in question, determines the login name (and the mail server host name if necessary) and generates the individual XML configuration that matches the email address.

The programming required is manageable – and can possibly be completed by the *AutoMX* project described in Section 17.4.

To save companies with multiple domains or ISPs with many customers from having to set up countless virtual `autoconfig.<customer domain>` web sites, a CNAME can be used in customer domains to refer to a central autoconfig instance belonging to the ISP. In this example, the ISP is `mailbox.org`:

```
autoconfig.example.com. IN CNAME autoconfig.mailbox.org.
autoconfig.example.net. IN CNAME autoconfig.mailbox.org.
autoconfig.example.org. IN CNAME autoconfig.mailbox.org.
```

The host name `autoconfig.mailbox.org` should refer not to a *name-based* virtual host but rather to an *IP-based* virtual host, i.e. an IP address on the server reserved solely for this purpose. The web server then no longer cares which host name the client believes it is asking for; it handles all queries exactly the same and returns the same XML file for all clients.

Now make sure that your required XML file is available for download from the URL `http://autoconfig.mailbox.org/mail/config-v1.1.xml`.

17.2. Autodiscover with Outlook

Outlook (from Outlook 2007 SP 1) also supports the automatic search for the correct mail server settings. However, the feature is called *Autodiscover* rather than *Autoconfig*. There is no need to mention that this mechanism is based on an XML file but proceeds quite differently under the bonnet…

When an email account called `klaus@example.com` is set up, Outlook searches these versions one by one:

1. Download from `https://example.com/autodiscover/autodiscover.xml` – careful, the search term is not `www.example.com`!
2. Download from `https://autodiscover.example.com/autodiscover/autodiscover.xml`
3. Query in the DNS record SRV for `_autodiscover._tcp.example.org` and https download of URI `/autodiscover/autodiscover.xml` from the host specified there.

Set up proper HTTPS!

Unlike in autoconfig, the URL here must always be available under `https://`!

Once again, ISPs are faced with the difficult task of keeping a configuration ready for thousands of customer domains – and many customer sites often do not even provide `https://`.

You have to guide the client to your Autodiscover URL in various ways. In this example, the provider is `mailbox.org`:

The best method is to place an HTTP redirect in the customer's web site – but the original customer web site must already own a valid https certificate, which is often not the case. `https://autodiscover.example.com/autodiscover/autodiscover.xml` must then trigger a redirect to `https://autodiscover.mailbox.org/autodiscover/autodiscover.xml`. Enter the following in the Apache configuration of the domain:

```
Redirect /autodiscover/autodiscover.xml https://www.mailbox.org/autodiscover/autodiscover.xml
```

Alternatively, you can also use an `SRV` entry in the DNS zone of the customer web site to smooth the way and thereby provide the final Autodiscover host name of the provider. To do so, set a DNS record as follows:

```
_autodiscover._tcp.example.com. IN SRV 0 0 443 autodiscover.mailbox.org.
```

That is usually the easiest method. However, Outlook then warns the user that he has been redirected to a different URL for configuration.

Finally, like Mozilla's `autoconfig`, a CNAME is also possible in the DNS data of the customer's domain, but then the HTTPS certificate will no longer match the requested name from the client's perspective. In that case, Outlook will return a corresponding warning about the certificate. But that is not important if you only use self-signed certificates.

Microsoft assumes that this Autodiscover URL hides a web service that receives the XML-packaged query from the client in the form of HTTP-POST, parses it and then returns the completed configuration.

However, like in the above Autoconfig scenario for Thunderbird, you can also place a single hard-coded text file behind this URL that returns the same settings for all email addresses of the provider.

The example below configures the client

- on standard port 143 (or 110) for IMAP (or POP3) with TLS
- on submission port 587 for SMTP with TLS; in this case, the `<UsePOPAuth>` parameter should use the POP3/IMAP passwords
- using the `<SPA>` parameter (*Secure Password Authentication*) for logins using the CRAM password procedure (see Section 5.10)
- and with the email address as the user name, because the empty container `<DomainName></DomainName>` causes Outlook to recycle the domain of the email address

```
<?xml version="1.0" encoding="utf-8" ?>
<Autodiscover xmlns="http://schemas.microsoft.com/exchange/autodiscover/re
<Response xmlns="http://schemas.microsoft.com/exchange/autodiscover/outloo
<Account>
<AccountType>email</AccountType>
<Action>settings</Action>
<Protocol>
<Type>IMAP</Type>
<Server>imap.mailbox.org</Server>
<Port>143</Port>
<DomainRequired>on</DomainRequired>
<DomainName></DomainName>
<SPA>on</SPA>
<SSL>on</SSL>
<AuthRequired>on</AuthRequired>

</Protocol>
<Protocol>
<Type>POP3</Type>
<Server>pop3.mailbox.org</Server>
<Port>110</Port>
<DomainRequired>on</DomainRequired>
<DomainName></DomainName>
```

```
<SPA>on</SPA>
<SSL>on</SSL>
<AuthRequired>on</AuthRequired>
</Protocol>

<Protocol>
<Type>SMTP</Type>
<Server>smtp.mailbox.org</Server>
<Port>587</Port>
<DomainRequired>on</DomainRequired>
<DomainName></DomainName>
<SPA>on</SPA>
<SSL>on</SSL>
<AuthRequired>on</AuthRequired>
<UsePOPAuth>on</UsePOPAuth>
</Protocol>
</Account>
</Response>
</Autodiscover>
```

Once again, the order in which you list IMAP and POP3 should depend on the service that you prefer clients to use.

Microsoft has provided a rather useful summary of additional information about the Autodiscover mechanism on a technet page.[71]

17.3. Automatic configuration for Mac OS, iOS and Windows LiveMail

Even though they profess to be user friendly, Mac systems do not have a mechanism for determining the correct server configuration in the background automatically and unnoticed. However, they do allow the further set-up of an email account to be controlled by a control file that the user has to download manually.

But you can also use the *AutoMX* project (described below) to simply redirect your Mac users to a separate web interface that creates the individual configuration file and offers it for download.

Unlike Outlook, Windows LiveMail does not have an Autodiscover mode (unless you are configuring Hotmail, outlook.com or a mail service from Microsoft). The only option here is traditional manual configuration.

17.4. Dynamic configuration scripts in PHP and Python

In most cases, it should be enough for businesses and ISPs to use the above methods to store hard-coded XML configuration files on the web server. However, if you want to return different mail servers for different accounts (e. g. because you are using partitioned clusters and do not want to use an IMAP proxy, Section 16.2), or if you first want to calculate the matching login name from the email address, you will not get around a programmed solution that evaluates the queries from the client and generates the response configuration by means of an SQL or LDAP lookup.

David Ramsden has introduced a lean PHP solution that you can also use as a basis for additional configuration to match your own infrastructure.[72]

The *AutoMX*[73] project also resolves this task in all its aspects. It is written in Python and expects corresponding requirements for the server modules and modifications to the Apache or nginx configuration. That means you cannot just dump it in a LAMP stack, but first have to prepare the server for it in several places.

AutoMX is especially suitable for determining the individual mail servers for every user by means of database lookups and entering them in the autodiscover script. Use it if you can get no further with the static XML files described above and cannot do without database lookups matching the query.

---

[70] You will find documentation and examples on https://wiki.mozilla.org/Thunderbird:Autoconfiguration:ConfigFileFormat and https://developer.mozilla.org/en-US/docs/Mozilla/Thunderbird/Autoconfiguration/FileFormat/HowTo

[71] http://technet.microsoft.com/en-us/library/cc511507.aspx The page also contains advanced examples of complicated set-ups containing additional options.

[72] http://0wned.it/geek-bits/scripts/open-source-autodiscover-implementation-in-php

[73] http://www.automx.org

Chapter 18. Webmailer as a complementary service

IMAP allows you to read all your emails anywhere, and this approach is practically crying out to be complemented with webmail clients. There are two fundamental approaches: direct file system access to the email storage, or access via IMAP.

You don't want webmailers to access the file system of the IMAP server directly – partly because you would have to open up the email storage, which is not desirable for reasons of security, and partly because that requires standard storage formats such as mbox or Maildir in their purest form.

Some webmailers want direct access to create Procmail scripts or perform other tricks in the email storage. Quite apart from the fact that you should replace Procmail with Sieve, this concept will quickly reach its limits: you cannot separate the web server and the mail server, support for larger email clusters with multiple machines is difficult, and then there are safety concerns and problems with file access permissions that require tricky solutions; then there is the fact that major setups use Dovecot's own format, mdbox, or even other storage formats, and you will see that this is in no way a practical procedure.

People used to claim that direct access via the file system consumes fewer resources than a webmail client that logs in repeatedly using IMAP. But this argument no longer holds true. Thanks to its caching methods, Dovecot is so efficient when accessing emails and when users log in that direct access by the webmailer in the Maildir directories would be counterproductive, particularly as direct file access would cancel out the caching and indexing benefits that Dovecot provides.

Dovecot is now so efficient when users log in and when individual emails are accessed that it is no longer necessary to use a caching proxy such as `up-imapproxy` because it brings almost no improvement (see Section 18.3).

So it is usually better if webmailers simply use the IMAP protocol to access the mail server and therefore don't care how and where the IMAP server saves its email storage and how many systems are in the cluster. Another advantage is that questions of authentication play almost no role in the installation of the webmailer. It passes the user's login data to the actual mail server via the IMAP connection and does not need a separate connection to the authentication database.

That also means that these webmailers can in principle be combined with any IMAP server; even though they are operated in the browser, they communicate with the server in the same way as any other desktop email client.

18.1. Squirrelmail and Horde/IMP

For many years, the *Squirrelmail* project was a commonly used PHP webmailer, but *Roundcube* has clearly taken over its role in recent years. Where Squirrelmail is still in use, it is because nobody has updated the installation yet. Squirrelmail has some nice extensions, but it is very rarely suitable for new installations. The GUI looks old fashioned and is clearly from another HTML age – you won't find a drag and drop feature like in Roundcube here.

Horde/IMP is another top dog and is based on the powerful *Horde* PHP framework. This solution allows you to construct entire groupware servers that go far beyond a simple webmailer function – on the other hand, plenty of traps lurk during the installation of the Horder system, and you certainly can't call it "lean". There are various Horde installation instructions on the internet.[74]

Now we should look at the Roundcube webmailer, which is definitely worth it.

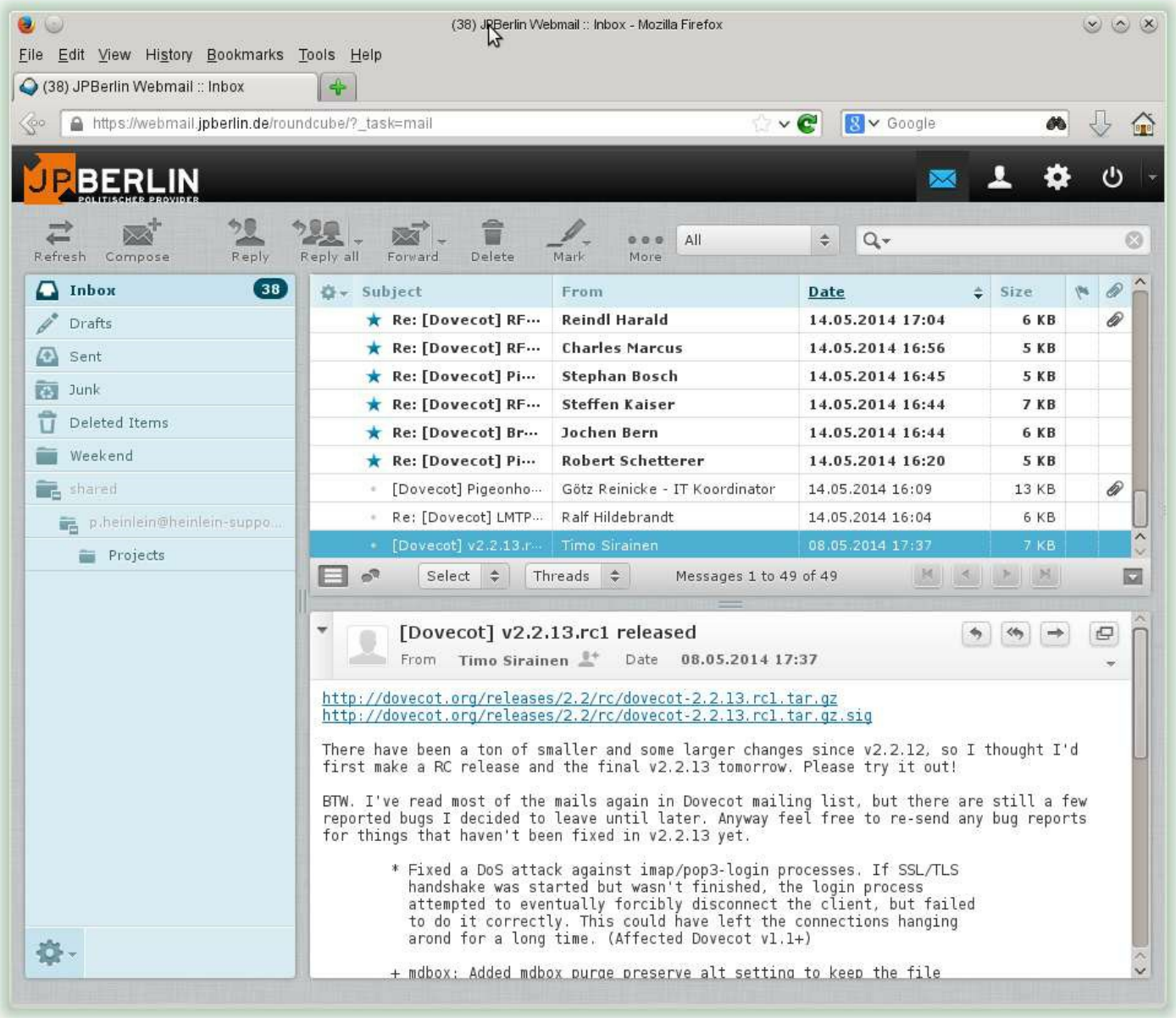
18.2. Roundcube

Roundcube has become the most popular webmailer for those seeking a simple, lean and well-built solution.

- Its clear layout and usability are particularly helpful for inexperienced users.
- Under its bonnet, there is great IMAP support which is fully capable of handling different namespaces, shared folders, ACLs and all the other advanced features.
- There is a very good Sieve module that allows even inexperienced users to click together their own Sieve rules (Roundcube is worth it for that ability alone).
- Of course it also supports global and personal address books.
- There are also plenty of administrative advantages, such as a short time-out between the sending of two emails (a very useful way to keep spammers at bay who want to use stolen account data to send off their spam emails from the webmailer); it is also possible to specify precisely whether a user may change his real name and email address in his sender profile.

Plenty of plugins are available for Roundcube, most of them complementary to the webmailer: folders can be created and subscribed automatically, text emoticons can be replaced with emojis (a feature you should not underestimate, as it is very popular with users), attachments and emails can be downloaded as a zip archive, and, while the user is writing, temporary copies are saved automatically in the draft folder every few seconds.

Figure 18.1. Roundcube is easy to operate



Roundcube is less strong when it comes to calendar and groupware functions. There are some plugins with a good approach, but the quality is definitely different from that of the rest of the Roundcube code. On the other hand, Roundcube (unlike the Horde system) is mainly defined as a webmailer, and not as a framework for all the surrounding tasks.

The developers of *Kolab*, a groupware solution, chose Roundcube as the basis for their web client and included Roundcube developers in their team. As a result, the Kolab project contains working groupware and calendar modules, but they are subject to Kolab licensing and not freely available. It is still interesting that the support from the Kolab team allows plenty of development work to flow into Roundcube, which is free, and where it benefits the entire community. It is worth keeping an eye on this development, as it could help Roundcube make a leap forward.

18.2.1. Installation

Roundcube is based on an SQL database that is used not just to manage user profiles and address books, but also (if configured accordingly) to cache entire emails. PostgreSQL, MySQL and SQLite are supported.

`pgsql` or `mysql`
: You should use one of these if you have corresponding database servers available as part of your internet presence and can provide the databases Roundcube requires without too much effort or cost. Even if you are using multiple Roundcube instances as part of a high-availability setup, a central SQL database can make sense, particularly if it is set up for high availability at your end.

`sqlite`
: Makes a database available only via a local database file, so does not need its own running daemon as a separate database instance. If Roundcube is simple and running in one instance, `sqlite` is an attractive version requiring little administration and monitoring. You can even serve setups with many tens of thousands of users. In cluster setups, however, you have to take care of the replication of the SQLite file in question, so "real" central databases are better at bringing out its advantages. SQLite is also less suitable for setups with more than 10,000 users: as the file has to be equipped with a write lock for every write access, a webmailer can reach its limits if many users access it at the same time.

If you install Roundcube from a Debian package, guide the installation scripts of the package management through a short setup dialog as standard by specifying, amongst others, the database type as well as the access data. Setting up the SQLite database for Roundcube also becomes child's play.

Installation under SUSE

You will find Roundcube in the SUSE installation sources under the package name `roundcubemail` and can simply install it using `zypper in roundcubemail` (or via YaST). You will then find the configuration data in `/etc/roundcubemail` and the program code under `/srv/www/roundcubemail`. You will find the settings for Apache under `/etc/apache2/conf.d/roundcubemail.conf`; that is also where the alias is set, you can reach Roundcube via URL `http://<host>/roundcube`.

When you call up the Roundcube URL, you will see that Roundcube is not yet running because there is no functioning database configuration. Make the settings for database type, server name and access data in `/etc/roundcubemail/db.inc.php`. In `/srv/www/roundcubemail/SQL` you will find SQL dumps for setting up the tables.

Using an SQLite database is easy, you just have to bear the file access permissions in mind. You can store the database file in `/var/lib/roundcubemail`, for example, as shown below:

```
flash:~ # mkdir /var/lib/roundcubemail
flash:~ # chown -R wwwrun:daemon /var/lib/roundcubemail
```

Then you just need to enter the right path to the SQLite database in `/etc/roundcubemail/db.inc.php` (the many slashes in the path are correct):

```
// PEAR database DSN for read/write operations
// format is db_provider://user:password@host/database
// For examples see
// http://pear.php.net/manual/en/package.database.mdb2.intro-dsn.php
// currently supported db_providers: mysql, mysqli, pgsql, sqlite, mssql
// or sqlsrv

$rcmail_config['db_dsnw'] = 'sqlite:////var/lib/roundcubemail/roundcube.s
```

When you call Roundcube for the first time, it sets up the corresponding database file automatically:

```
flash:~ # ls -l /var/lib/roundcubemail/
-rw-r--r--  1 wwwrun daemon 51200 Aug 13 11:41 roundcube.sqlite
```

Installation under Debian

There are also Roundcube packages for Debian. However, these are split into various sub-packages called `roundcube-*`. Install `roundcube-core`, `roundcube-plugins` and `roundcube-plugins-extra` as well as the `roundcube-(mysql|pgsql|sqlite)` package matching your database.

Fortunately, an installation script asks for the database type and necessary information and then performs the configuration of `db.inc.php`. SQLite also works without requiring further modification.

Installation from the source code

Of course you can also install Roundcube directly from the source code. In this case, you will find the config directory containing files `db.inc.php` and `main.inc.php` in the web space; you should also make sure that the web server user can write to the directories `temp` and `logs`, as that is where Roundcube logs helpful messages (if configured accordingly).

18.2.2. Configuration

As your next step, you should take a close look at the Roundcube configuration `main.inc.php`. The file is extensive but has good documentation, so I will just describe the most important settings that you should modify:

`$rcmail_config['default_host'] = 'imap.example.com';`
: Enter the name of the IMAP server with which Roundcube should connect. If you do not enter a name, Roundcube will ask the user for the IMAP server name in the login screen. If you enter multiple host names, Roundcube will offer these to the user in a drop-down box. If you prefix the host name with `ssl://`, the login is performed on SSL ports 465 (SMTPS), 993 (IMAPs) or 995 (POP3s), while `tls://` will ensure that login takes place on the standard ports and is followed by an STARTTLS command: `tls://imap.example.com`.

`$rcmail_config['smtp_server'] = 'smtp.example.com';`
: To prevent Roundcube from passing all emails to `/usr/bin/sendmail`, you should specify an SMTP server here, ideally the same one as your user usually uses. Once again, you can use `ssl://` and `tls://` as the prefix.

`$rcmail_config['smtp_user'] = '%u';` and `['smtp_pass'] = '%p';`
: Usually Roundcube would relay any way via the SMTP server due to its local IP address if the Roundcube IP is contained in `$mynetworks` in Postfix. It is often advisable for Roundcube to log on to the server with SMTP-Auth/SASL. That makes it easier to trace misuse to a specific user account (and also makes it easier to block such an account). Enter `%u` and `%p` so that Roundcube will pass the user's login name and password to the SMTP server.

`$rcmail_config['force_https'] = true;`
: If you want Roundcube to be accessed via `https` and not `http`, this setting will instruct Roundcube to ensure this happens automatically via a redirect.

`$rcmail_config['sendmail_delay'] = 4;`
: Webmailers (particularly those belonging to universities) are a popular target for phishers and spammers, who use stolen access data to send out spam and other things. Make sure that there is always a 4-second break between two emails, and spammers will rapidly lose interest in your system. At the same time, a user will not usually notice this small timeout, because he is unlikely to type and try to send an email in such a short time.

`$rcmail_config['max_recipients'] = 250;`
: This is another way to spoil the fun for spammers: set a high (but existing) limit to the amount of recipients an email can have. It should be rare for a user to send a message to many thousands of recipients from his webmailer (!).

`$rcmail_config['plugins'] = array('managesieve', 'bounce', 'subscriptions_option', 'vcard_attachments', 'emoticons', 'acl');`
: List the plugins from the `plugin/` directory that you wish to activate. But be careful. Some plugins have a separate configuration file in their directory; if plugins are incompletely configured, Roundcube may no longer work.

If you offer Sieve (see Chapter 12), you should definitely activate the `managesieve` plugin and set up the correct access data in `plugins/managesieve/config.inc.php`. Your users will then find a nice Sieve GUI in menu item *Settings/Filters*.[75]

The `ACL` plugin is important if you allow shared folders (see Chapter 9) and want users to be able to manage access for other users via Roundcube as well. Many email clients provide unsatisfactory ACL support, so it is nice that you can set up a central permissions management web interface available to all users in menu item *Settings+Folders+Folder properties* in Roundcube.

`$rcmail_config['date_format'] = 'd.m.Y';`
and `$rcmail_config['date_long'] = 'd.m.Y H:i';` set the default format for displaying the date to the German/European format.

`$rcmail_config['draft_autosave'] = 60;`
Particularly in web browsers, it is easy to close the window even though a long email remains unsent. Roundcube has the ability to buffer the intermediate state every few seconds/minutes in the draft folder. I think that the default value of five minutes is too long; 60 seconds appear more sensible.

Finally you should go carefully through all the settings in the `USER PREFERENCES` block. You can enter default values here so that new users are automatically given proper profiles. Changes here do not affect existing users who have already been created with a profile by Roundcube. So you should enter useful values here *before* you start running the web mailer. The user can still modify the settings to suit his requirements.

18.3. up-imapproxy – fast access through session caching

Webmailers have a fundamental design problem in that they are PHP scripts and therefore don't keep a permanent connection to the IMAP server open. Every time the user accesses his account, clicks on an email or selects a directory, the webmailer's PHP code has to create a new connection to the IMAP server.

That used to be a serious problem, as old IMAP servers such as Courier had few or no caching mechanisms and a user login was correspondingly "expensive", because a heavy file system load is required to import the IMAP folders and emails every time the user logs in.

This argument is no longer relevant: Dovecot is so efficient with its caching mechanisms that the login itself "costs practically nothing" anymore.

In the past, larger webmailers required you to interpose a special IMAP proxy such as `up-imapproxy`, which would efficiently keep connections to the IMAP server; today, you don't need this procedure even in larger setups if you use Dovecot.

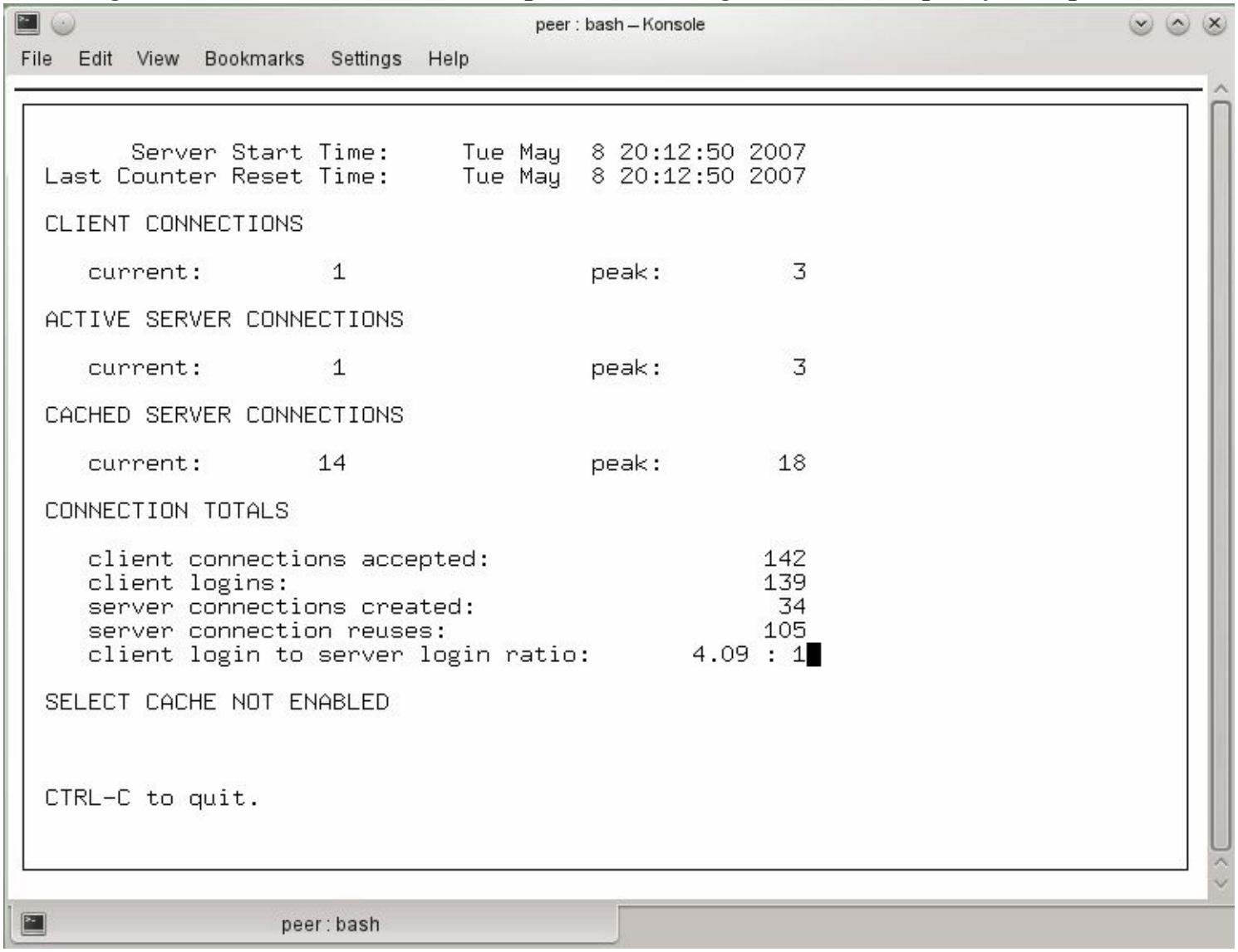
This last option is only added to make the list complete. It adds complexity and is a lot of work; only interpose `up-imapproxy` between the webmailer and the IMAP server if you absolutely have to. I almost never do so any more.

How it works: `up-imapproxy` keeps the connections to the IMAP server open even if the web server has already closed its IMAP connection, or if it had to close it. During the next access, it uses the cached login data to recognise the familar connection and fowards access to the IMAP server via the IMAP session that is still open. The webmailer's PHP script still has to log on to the `up-imapproxy` via IMAP, but then the hash value is simply used to fall back on the actual IMAP session; separate I/O operations such as importing the IMAP folder list do not take place in the `up-imapproxy`.[76]

Even though the IMAP proxy software is stable and trouble-free after installation, it is rarely included in distributions. Red-Hat packages are only built by third parties, and you will find packages I have created for openSUSE and SLES in the openSUSE Build Service.[77] That saves you having to compile the source code, and the init/systemd script is much better adapted to SuSE than the original one.

Next to the actual program `in.imapproxyd`, the `pimpstat` tool is also installed. In a similar manner to `top`, it provides a continuously updated overview of the number of connections, the connections not required because of caching, and various other details (Figure 18.2).

Figure 18.2. `pimpstat` makes the potential savings of the IMAP proxy transparent.



Just to repeat, in normal circumstances the use of up-imapproxy in front of a more recent version of Dovecot does not make sense even in larger setups. Save yourself the effort (and avoid this source of errors).

---

[74] Such as the one by Michael "Django" Nausch on the basis of RHEL/CentOS: http://dokuwiki.nausch.org/doku.php/centos:mail_c6:horde_1

[75] If you care a lot about Sieve, it may be worth taking a look at the alternative `sieverules` plugin: https://github.com/JohnDoh/Roundcube-Plugin-SieveRules-Managesieve

[76] You will find the source code on http://www.imapproxy.org, and there is a mailing list: https://lists.sourceforge.net/lists/listinfo/squirrelmail-imapproxy

[77] http://software.opensuse.org

Chapter 19. Migrating IMAP servers

The migration of the email data of a POP3/IMAP server to a new system is a decisive point that can make or break a conversion.

Be aware of these stumbling blocks:

UIDs should be retained
: Many POP3 clients use the email UIDs to determine which emails they have already downloaded. If you do not ensure that emails will have the same UID after conversion, some POP3 clients will download all of the messages stored on the server once again. That is not too bad for the POP3 users, who can delete the messages from the server after retrieval. But it has a surprising (and cleansing) effect on users who delete their emails locally from the POP3 email client but actually have kept all their emails in their INBOX that they have received since their email account was set up.

  The loss of the UID is less serious for IMAP clients: they too would reject the local cache completely and then download all the emails from the server once again. But the messages do not appear in the INBOX in duplicate as a result. For the server admin, a large-scale loss of email UIDs represents a serious strain in terms of bandwidth and transfer volume.

IMAP namespaces should remain unchanged
: During the conversion, make sure you map the namespaces correctly, i.e. whether the hierarchy separator is "." or "/" and, if applicable, if all folders have been equipped with an INBOX prefix (see Section 4.4.1).

IMAP flags and custom IMAP flags should be retained
: You can test whether your old system even permitted custom flags by performing a manual IMAP login (see Section 3.2.3). If you select a folder with the `SELECT` command, the server will return the possible flags in the list. `\*` means that custom flags are permitted.

Migration must take place in a manageable period of time
: Large mail servers often hold so much data that a migration cannot be performed at one specific time and must instead be performed in stages or while the servers are running.

As you can see, good planning, a comprehensive test run and careful analysis of the test results will provide an essential foundation to prevent unpleasant surprises during the migration.

19.1. Different ways to migrate a mail server

There are three different ways to migrate data from the old system to the new one:

Migration at file level
: Under the bonnet, many systems already work with mbox, Maildir or a storage format very similar to Maildir, so the email files can be accessed directly in the file system in ASCII. With a little shell magic you can program and automate a lot of the migration work; you may lose your IMAP flags, but that is usually not a huge problem. UIDs are also often generated anew, so that all clients synchronize again.

Migration on the basis of POP3/IMAP
: Alternatively, you can use the POP3 or IMAP protocol itself – after all, you can use IMAP to access the data on the existing system as well as upload the data to the new system. IMAP flags and subscribed folders are kept, UIDs have to be newly assigned, so the clients have to synchronize again afterwards.

  But if you are migrating from strange Windows IMAP servers that store their data in some proprietary format, that may be the only feasible method. Even if the data come from an external email provider whose file systems you cannot access, the only way is to use the IMAP protocol.

Migration using doveadm or imapc
: Dovecot itself provides numerous tools and interfaces that you can use to migrate or synchronize email data stocks. `doveadm backup/sync` is a convenient way to convert mbox, Maildir and mdbox formats. If you attach Dovecot in front of the old system as a transparent IMAP proxy, it is very easy to perform a migration while the system is running (see Section 19.3).

The fastest and often best way is a migration at file level, so if you can convert conveniently from the data format of the old system to the mbox or Maildir format used by Dovecot – with Maildir usually being the method of choice.

If you prefer to use mdbox as the target format and have to use the method involving the file system, you should first use one of the methods described here to convert to Maildir and then put your email storage into operation under Dovecot. Later on in a second step you can approach the conversion from Maildir to mdbox under Dovecot – Section 7.5 describes how to do so while the system is live.

19.1.1. Migrations at file level

For many POP3/IMAP server servers there are ready-made migration tools that export the files, auxiliary files and, if applicable, databases from the old system and then usually store them in the Maildir format that is suitable for Dovecot. That is the best way to keep the meta data, subscriptions and UIDs of the emails.

From Dovecot to Dovecot

If you are migrating from one Dovecot server to another, you can simply copy the files, e. g. using `rsync`.

If, however, you want to change the storage format within the Dovecot server, you should choose the migration path described in Section 7.5 so that the UIDs, flags and other meta data of the emails are retained.

From Courier IMAP to Dovecot

Courier always stores its data in a pure Maildir, but the files sometimes have different names than in Dovecot. It is essential that you follow the migration how-to guide on the basis of the `courier-dovecot-migrate.pl` tool here.[78]

From Cyrus IMAP to Dovecot

The `cyrus2dovecot.pl` script performs this task quickly and excellently. It keeps the UIDs, flags and all other meta data and is the method of choice.[79]

From uw-imap to Dovecot

Here, the emails are usually stored in mbox files, which can sometimes be subject to a directory structure that takes some getting used to. The `uw2dovecot.pl` tool performs the conversion task automatically.[80]

mbox-Files (e. g. from qpopper) to Dovecot/Maildir

qpopper and many other simple POP3 servers use traditional mbox files. You can approach them using `doveadm/dsync`, but the `mb2md.pl` tool maintained by Juri Haberland is also very helpful. It imports mbox files and generates completed Maildir directories; it is also suitable for converting large amounts of data.[81]

The following example shows how you can convert a user's individual mbox file. The path for the source file is specified with `-s`, and the path for the target directory with `-d`:

```
flash:~ # mb2md.pl -s /var/mail/<user> -d /srv/vmail/<user>/Maildir/
Converting /var/mail/<user> to maildir: /srv/vmail/<user>/Maildir
Source Mbox is /var/mail/<user>
Target Maildir is /srv/vmail/<user>/Maildir
666 messages.
```

From other IMAP servers to Dovecot

Often other Linux/Unix-based solutions store their emails in a way that is similar to mbox or Maildir. As soon as emails are provided in readable form in individual files, you can do a lot with simple shell scripts or the `mb2md.pl` tool mentioned above. Often,

only the UIDs of emails or the IMAP flags are stored in databases and are lost during a migration, but that is not so bad.

Always remember to modify the file permissions of the generated email data at the end if applicable.

19.1.2. Migrations using the POP3/IMAP protocol

There are systems that give you almost no opportunity to access email data directly, either because they are saved in encrypted form or because they are saved in an entirely different format from RFC 2822. Some Outlook/Windows-fixated systems only convert emails to the RFC format when they are retrieved by means of POP3/IMAP.[82]

There are some tools that perform migration at protocol level, including

`imapsync`[83]
: A flexible and mature migration tool by Gilles Lamiral – in my opinion, this is by far the best tool for this purpose. Gilles Lamiral now charges a fee of 50 euros for downloading `imapsync` – and the tool is definitely worth it. In many distributions, (older) versions of `imapsync` are packaged in a complete state. I will describe `imapsync` in more detail in Section 19.2.

`pop2imap`[84]
: If the old system does not support IMAP at all, `imapsync` has a younger brother able to migrate data from a POP3 server to an IMAP server.

`imap_tools`[85]
: A collection of Perl scrips, including some that copy IMAP server files to mbox files or upload emails from mbox files to IMAP servers. The tools are currently under further development.

`imap_migrate`[86]
: This PHP script expects an empty target mailbox and is therefore not suitable for continuous data matching – attempts will generate duplicates. It can provide a good initial basis for any custom developments.

`imapcopy`[87]
: Development of this tool stopped in 2006, but it may be worth a look if `imapsync` or `imap_migrate` are unsuitable.

All these procedures are based on performing a user login on the old and new systems via POP3/IMAP, so you will always require the user passwords in plain text for these methods. Section 19.5 provides a few hints on how to determine the passwords if they are unavailable or available only in hash form. Alternatively, you can also use logins via master users (see Section 5.11). Simple old mail servers often have no master user and use PAM to authenticate with `/etc/shadow`, so that only hashed passwords are available. In this case, you can fiddle in a master password via PAM for the duration of the migration.

19.2. Migration with imapsync

Based on my own experience, I recommend you use `imapsync` for migrations on the basis of the IMAP protocol. Not only is it stable and under active development, it also provides the option of continuously synchronizing IMAP folders. You can repeatedly and incrementally migrate a mailbox without finding duplicate emails on the target system.

That is important, because migration via IMAP takes a relatively long time simply because of latencies and serial email retrieval. `imapsync` allows you to begin while the system is running so that as many emails as possible from every account can be copied to the target system initially. If there are many thousands of accounts and many GB of email data, the first run can take many days, but that should rarely be an issue because productive operation continues in the meantime.

The next `imapsync` runs take considerably less time because the majority of data have already been copied. That means that the synchronization time window shrinks constantly.

There are two ways of completing the final conversion:

Conversion with downtime
: When the final migration takes place, you have to block user access to the old system and schedule a downtime, which will be correspondingly short if you perform this type of preparation, and which you can schedule for one weekend night. `imapsync` then just has to copy the emails that were added since the last synchronization, remove the emails from the target system that have been deleted in the meantime, and adapt any modified email flags. On the basis of the last run-throughs, you should be able to estimate how long that will take.

Conversion while the system is running
: Alternatively, you can also perform the migration at a fixed time once the preparatory runs have been completed, but only perform the final synchronization from the old system to the new one afterwards. However, in this last run, emails may only be copied from the old system to the new one but may no longer be deleted. To provide a clean synchronization, `imapsync` would usually use the `--delete` parameter to delete the emails it could not (any longer) find on the old system. Of course it may not do that after a migration after conversion, because `imapsync` would otherwise delete all the emails received on the new server in the meantime. In this case, `imapsync` must therefore always be used without the `--delete` parameters. In the worst case, this will restore an email that the user already deleted on the new system.

Use the following example to copy not just the `INBOX`, but also all the other IMAP folders belonging to user `tux`:

```
flash:~ # imapsync --host1 oldmail.example.com --user1 tux \
  --password1 "secret"  --host2 newmail.example.com --user2 t.tux \
  --password2 "secret"
```

If you create a list of all user names and passwords, you can automate a run of many thousands of accounts.

However, the transfer of passwords in call parameters is tricky, because they end up in the bash history. In addition, even unprivileged (!) users will see the `imapsync` process when they look into the process list – along with the passwords provided as options. `imapsync` therefore explicitly offers to read them from a separate file. This file should be in a secure directory and have read permissions only for `root`:

```
flash:~ # cat /root/pw1
secretpassword1
flash:~ # cat /root/pw2
secretpassword2
flash:~ # imapsync --host1 oldmail.example.com --user1 tux \
  --passfile1 /root/pw1  --host2 newmail.example.com --user2 t.tux \
  --passfile2 /root/pw2
```

The following call parameters are helpful:

`--dry`
: `imapsync` executes the snychronization run in read-only mode and does not modify any data – the ideal test.

`--delete`
: deletes the emails on the original host specified by `--host1` after the migration has been completed successfully.

`--delete2`
: deletes emails on the target host specified with `--host2` that do not (any longer) exist on the source host. That is important if you run `imapsync` multiple times, so perform an incremental migration.

`--regextrans2`
: If you have to rename folders on the fly or want to introduce or remove the INBOX prefix, you can use this parameter to specify regular expressions in order to achieve a renaming of the folders:

  ```
  --regextrans2 s/INBOX/INBOX.old-inbox/
  ```

`--ssl1` and `--ssl2`
: switches on SSL encryption to the source or target computer. These two parameters show clearly how the script works: it "sits" between the two servers and keeps a separate connection open to each one.

`--subscribe`
: use this parameter to ensure that the folders subscribed on the old system are also subscribed on the new one (see Section 3.2.4). Do not forget to do so.

`--help`

provides a list of the numerous call parameters. That allows you to specify more complex selection criteria (size, age, folder names) for the emails to be migrated and to make changes to the names of the IMAP folders.

19.3. Transparent migrations with imapc

From Dovecot version 2.1.4 onwards you can also use the `imapc` Dovecot caching mechanism to perform a migration. In these cases, Dovecot is linked as a proxy in front of the original system and transparently transfers all the UIDs and IMAP flags of the emails, so nothing changes from the email client's perspective.

This method works best if you have access to the old system with a `master` user (see Section 5.11) so you don't need to worry about passwords. Otherwise you need access to the plain text passwords of the users to be migrated.

Create a `/etc/dovecot/conf.d/25-imapc.conf` file and configure the access to the old existing system:

```
imapc_host = imap.example.com
imapc_user = %u
imapc_master_user = master
imapc_password = secret
imapc_features = rfc822.size
```

If all folders were arranged under the INBOX in the old system (see Section 4.4.2), i.e. the IMAP namespace contains a `prefix=INBOX.`, you must map this in the `imapc` configuration as well:

```
imapc_list_prefix = INBOX.
```

If you wish, you can also map the access using a secure SSL/TLS connection; if you use valid certificates, you can also have them checked:

```
# for SSL:
imapc_port = 993
imapc_ssl = imaps
imapc_ssl_ca_dir = /etc/ssl
imapc_ssl_verify = yes
```

If you only access the existing system using POP3, you can use the following method:

```
pop3c_host = pop3.example.com
pop3c_user = %u
pop3c_master_user = master
pop3c_password = secret

# for SSL:
#pop3c_port = 995
#pop3c_ssl = pop3s
#pop3c_ssl_ca_dir = /etc/ssl
#pop3c_ssl_verify = yes
```

```
namespace {
  prefix = POP3-MIGRATION-NS/
  location = pop3c:
  list = no
  hidden = yes
}
```

In addition, you have to provide the `doveadm` command with a special `pop3_migration` plugin when migrating from POP3; it helps Dovecot to match IMAP UIDs with POP3 UIDs:

```
protocol doveadm {
  mail_plugins = $mail_plugins pop3_migration
}
plugin {
  pop3_migration_mailbox = POP3-MIGRATION-NS/INBOX
}
```

Once all this has been properly configured, `doveadm backup` will perform a download from the old server to your local storage (parameter `-R` reverses the direction):

```
flash:~ # doveadm backup -R -u user@domain imapc:
```

If you have no `master` login and therefore require the individual access data to access the old system, you can transmit the user names and passwords individually with each call:

```
flash:~ # doveadm -o imapc_user=foo -o pop3c_user=foo -o imapc_password=ba
```

`doveadm backup` would overwrite all local changes with the state of the existing system. Until the migration is complete, no local changes should therefore be made – in particular, no new emails should be saved locally…

Don’t forget to remove the `pop3_migration` plugin when the migration is complete.

19.4. Changes to folder names

If you have to modify the names of IMAP folders during a migration, you may encounter the problem that the target system only permits folders under the `INBOX` and a folder name such as `friends` is no longer permitted. The solution is to convert `friends` into `INBOX.friends`. Some complications can develop during this process.

What happens if a user had `friends` and `INBOX.friends` on the old system? This scenario is a realistic one, as email clients are often unsure whether to create the `Trash`, `Sent` and `Drafts` folders parallel to or under the `INBOX`. If you are unlucky, a user's desktop email client will create the directories in parallel, while the additional webmailer will create them underneath – and then you have a duplicate directory. Of course you can copy them together during migration, but that requires manual work.

On the server side, you have to consider the modified folder names in two places and repair them with `sed` using find/replace:

- In the `subscription` files, Dovecot saves which folders the user has subscribed.
- You may also need to adapt the folder names in Sieve filtering rules. `fileinto Mailinglisten.Dovecot` now becomes `fileinto INBOX.Mailinglisten.Dovecot`. Make sure to make replacements only in the lines that also contain the `fileinto` command. The following `sed` command is a good starting basis:

```
flash:~ # sed 's/fileinto\ \"/fileinto\ \"INBOX\./'  phpscript.sieve
```

If you simply move folders in this way, it may also cause the local filter rules in the users' email client to become unusable and require adjustment. In this case, you have no other option than to inform your users in advance and ask for their understanding.

The Squirrelmail webmailer reacts particularly sensitively to filter rules with non-existent target folders; next to Sieve, it also allows custom filter rules to be defined, which it then applies when retrieving emails. Usually Squirrelmail will import the potentially modified folder structure from the server after the IMAP login. If, however, filter rules are defined that refer to IMAP folders that no longer exist, Squirrelmail will crash and the user is unable to log in again. In this case, you should modify the user-specific profile files in the `data` folder under the Squirrelmail directories. If users have defined their own rules, you will find entries in the profiles that match the following schema:

```
filter0=From,tux@example.com,INBOX.friends.tux
filter1=From,support@heinlein-support.de,INBOX.work.heinlein-support
```

19.5. Determining plain text passwords

Many setups save passwords only in hashed form. If you have to switch authentication sources during a migration, that can lead to disaster. Even if the new authentication database does not save the passwords in plain text, they may be necessary when creating passwords.

In addition, it is common for incompatibilities to develop in the precise way a hash is calculated. The term MD5 hash in particular is often used to referred to very different things, so that hashes are often not compatible with each other. In that event, existing passwords can no longer be used, even though that should not be a problem in theory. With the `doveadm pw` command (Section 5.10.5) you can generate hashes or determine how Dovecot calculates the corresponding hash.

The use of `imapsync` and other IMAP migration tools also assumes that you are in possession of user data that you can use to log in in place of the normal users. If you have neither user passwords nor a general admin/master login with which you can assume the identity of individual users, this incredibly convenient method of data migration is not open to you. You will then wish you had not done without storing plain text passwords. But that does not mean there is no solution; as described earlier, clients actually transmit passwords in plain text every login in the case of hashed passwords.

In that case, check whether the existing product has a debug mode in which passwords may be logged. That way, you can collect the plain text passwords of all active users over time. Pay particular attention to the restrictive file permissions of the email log file.

If the starting basis is a Courier server, you can use the `DEBUG_LOGIN=2` option in the `authdaemonrc` file (previously: `pop3d` and `imapd`). While Courier is running, you can use this method to determine the passwords that you can then transfer to the authentication database.

In systems that do not use plain text passwords, the user are often limited to the `PLAIN` and `LOGIN` authentication methods, where the password is transmitted in plain text during login. That allows you to continuously sniff and evaluate the login data without much effort in order to update your authentication database. Common sniffing tools perform all these tasks completely automatically and finally present a clear list of the overheard user data.

Please note that these explanations are not provided for readers who are subject to German criminal law. Using such tools can now constitute a criminal offense in Germany (“hacker paragraph”, §202c StGB) even if you are actually preventing your user accounts from being hacked and this is your only reasonable way to increase the safety of the system. Administrators with German nationality must therefore check the legal situation in Germany

and abroad before using such tools, and the same is true for administrators with other nationalities who apply these tools within the Federal Republic of Germany.[88]

You can also obtain the plain text password by means of a custom web frontend that users must log in to to trigger the migration of their mailbox to the new server. The script it calls has to create the user account on the new server and can then start tools such as `imapsync`. This method is often faster and more reliable than logging or sniffing passwords. On the other hand, this method is identical to the one used in password phishing. So you should consider carefully how to explain to your users the difference between "your" legal web frontend and a phishing attempt from the outside. Actually you want to educate your users to *not* enter their account data in random web frontends…

Let your users know in advance (maybe even by an actual letter) and make it clear that this is a real exception and that they will never be asked for their login data without a previous announcement. Make sure that your web frontend can be reached under a clean domain that is definitely yours, and inform the users of this situation separately if necessary.

Migrations from Exchange servers to other programs and authentication services can also be performed well in this way, as this software allows neither the logging of passwords nor the reading of password files (regardless in which form) from the Active Directory – even hashed password data remain hidden.[89] I have had good experiences with this method in a variety of projects.

---

[78] http://wiki2.dovecot.org/Migration/Courier

[79] http://www.cyrus2dovecot.sw.fu-berlin.de/

[80] http://wiki2.dovecot.org/Migration/UW

[81] http://batleth.sapienti-sat.org/projects/mb2md/

[82] In such cases, IMAP support may be poor. We have encountered systems in the past that assigned new IMAP UIDs at every login, which made repeated incremental migration impossible. We have also found systems whose live conversion to IMAP was so slow that calling off all emails via IMAP would have taken days even for small amounts of data.

[83] http://imapsync.lamiral.info/

[84] http://www.linux-france.org/prj/pop2imap/

[85] http://www.athensfbc.com/imap_tools

[86] http://freshmeat.net/projects/imapmigration/

[87] http://home.arcor.de/armin.diehl/imapcopy/imapcopy.html

[88] There has been a lot of noise and discussion in recent years about this legendary hacker paragraph. Though it was meant well, badly written legislation has turned it into a boomerang for administrators who want to check that their systems are safe. In any case, even though the hacker paragraph has

been valid for many years, there have been virtually no cases so far, and this unlucky piece of legislation has been studiously avoided by everyone.

[89] I would be grateful for any hint on how to obtain these data in spite of this.

Chapter 20. Enterprise features and support

In May 2011, Timo Sirainen, Mikko Linnamäki and Markku Kenttä founded “Dovecot Oy”, a Finish company headquartered in Helsinki. The company can be found at http://www.dovecot.fi and offers, amongst other things:

- commercial support for Dovecot systems direct from the developers, including 24/7 SLAs
- some plugins and extensions requiring a commercial license for large ISPs under the roof of “Dovecot Pro”

The closed developments of Dovecot Oy are interesting for systems with several hundred thousand or even millions of users, because they resolve the problems (described in this book) that apply in highly scaled environments. In this chapter I will introduce a few concepts and take a brief look at the things we can expect in the coming months and years in terms of Dovecot.

For the features described below, you require a separate license from Dovecot Oy. You will then receive user name/password-protected access to a separate repository of `dovecot-ee` packages and a license file for using the object storage described below.

20.1. obox: Dovecot object storage

Traditional file systems on block-oriented devices often reach their limits even in the two-digit TB range. The time it takes to perform a standard file system check shows that truly large systems can no longer be operated using conventional technologies.

In recent years, various cloud and virtualization solutions have allowed different kinds of *object storage* to establish themselves on the market, which store data quite differently than the traditional file systems do.

An object storage consists of a multitude of, ideally, geographically distributed servers where data are stored (*object storage daemon*, OSD). These data can consist of individual files or of blocks from a large file that is thereby distributed among multiple storage servers. Like in RAID-10, where data blocks are distributed among multiple hard drives, in object storage you distribute data blocks to the storage servers – and thereby exploit the resulting improvements in speed.

The storage servers themselves save the data blocks you have entrusted them with on their own local data storage. In the simplest case, it can even do so in a normal file system such as ext4, xfs or NTFS, if the cloud storage provider does not run an own solution in its infrastructure. Where and how the OSDs store the data is of no importance to you as a user. You use the address for a data object in the global storage and call up data or store them there.

That means an object storage

- can be distributed globally
- can be made replicated and fail-safe by saving the data object several times in different geographical locations, so that the failure of an individual OSD node is not essential
- is no longer block-oriented and can grow to any size in principle
- is always available for concurrent read and write queries from a multitude of clients
- can be operated and addressed in massively parallel fashion due to its distribution; as a result, performance is finally limited by bandwidth and less by the I/O performance of the data storage units

So an object storage solves all the problems of scaling that usually plague postmasters of larger setups.

By the way, one of the most famous object storages is the world wide web, though it wasn't referred to as such in the past. There are search mechanisms that help find the data stored there, which are then addressed via a URL (= address of the data object). It does not matter to the user how a web server (= OSD storage node) stores the data internally. You don't even

need to care about the operating system of an individual web server. The internet is multiply redundant, and data can be stored in any number of replications ("mirror"). In terms of structure, the world wide web is designed for unlimited growth – in any case, scaling does not affect performance – and this is highly important.

After mbox, Maildir and mdbox, there is now a (commercial) plugin for Dovecot for the *obox* format, i.e. for storage in an object storage. Dovecot uses it to access all standard object storages. As of February 2014, examples include:

- Amazon S3
- Ceph via the S3 interface Rados-Gateway
- Dropbox
- OpenStack Swift
- Scality CDMI
- Windows Azure

A Dovecot cluster can thus be connected quickly and conveniently to a commercial provider and thereby achieve global, location-independent replication and availability. But even companies who have discovered object storage for themselves (e.g. as part of the OpenStack cluster with Ceph/Rados) can still use Dovecot with their custom storage and exploit the advantages.

Large mail clusters with millions of users that used to have to be partitioned due to the limits of the respective file systems (Section 16.2) can now be operated as a homogeneous system. The limit created by file systems of restricted sizes no longer exists.

However, object storage, like all services *in the cloud* (i.e. on the internet) has the problem of latency. It takes several milliseconds to send IP packages around the wold, and they add up. In a locally operated Ceph cluster, this factor is hardly relevant, but the usability would be noticeably reduced for the object storage of a cloud provider.

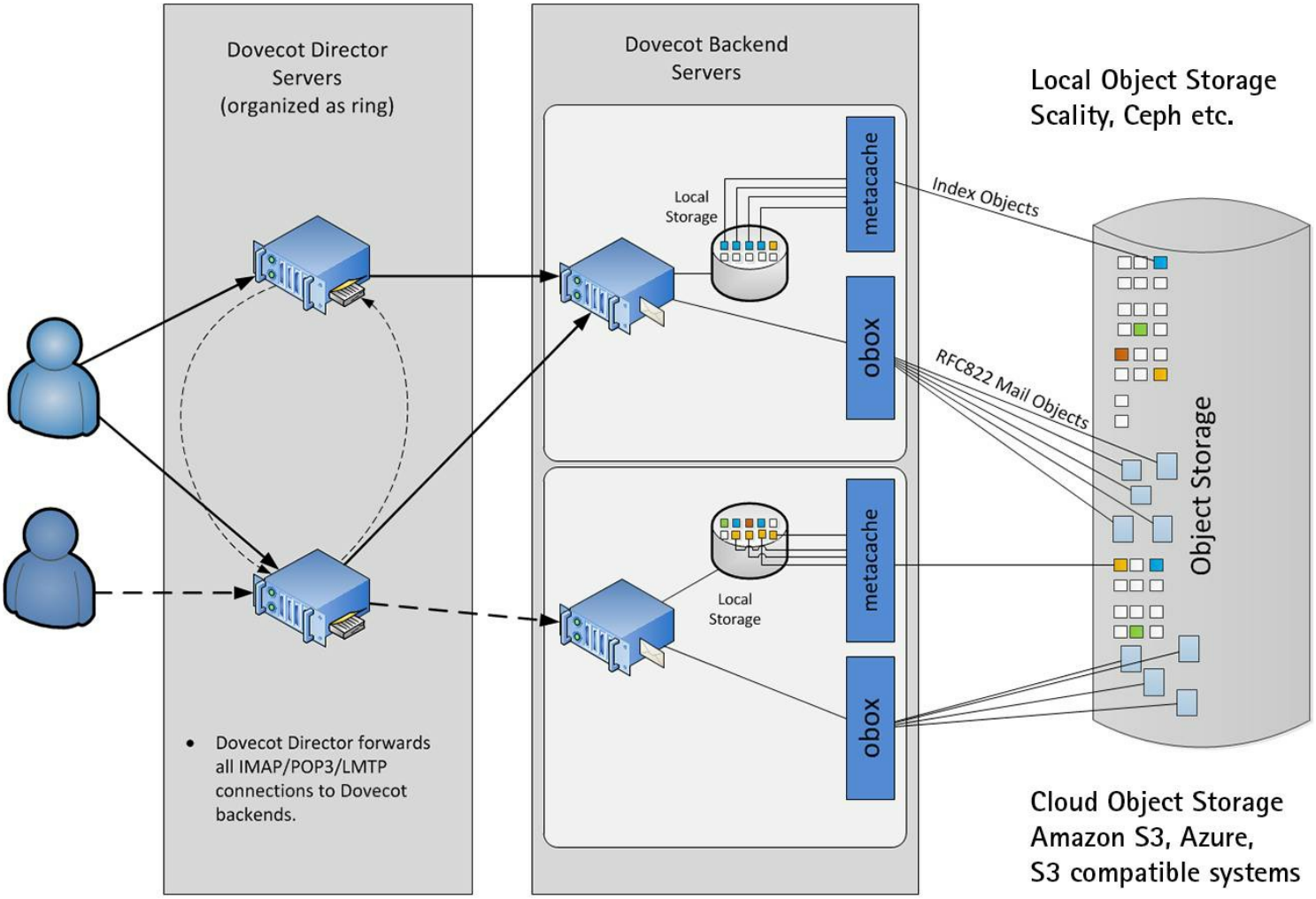
So the main factor for speed is to retain the meta data of email objects locally on Dovecot servers so they are quickly available. That is hardly surprising; after all, good caching in the index files was the main reason, even under Maildir, why Dovecot was able to leave all other IMAP servers behind in terms of performance. But that cannot be the ultimate answer: locally stored index files contradict the concept of a large distributed active/active cluster, where the user's random choice of node should be completely irrelevant.

The Dovecot `obox` storage format is once again the best of all worlds:

- Email objects are distributed in the object storage.
- Email meta data are also distributed in the object storage, but are stored as a cache locally on the IMAP server for active users.

Figure 20.1 shows the schematic structure of a Dovecot cluster with object storage.

Figure 20.1. Dovecot with object storage: Index files are checked out from the object storage and cached locally.



20.2. Configuration of various object storages

To install the object storage, first set up the `dovecot-ee` repositories for which you received access data from Dovecot Oy, and install Dovecot from the `dovecot-ee` packages. You should also install the `dovecot-ee-obox` package. Under Debian, you still need to activate Dovecot in `/etc/default/dovecot`:

```
# Set to '0' or 'n' to explicitly disable starting Dovecot
ENABLED=y
```

Then you should be able to start Dovecot. If you have not done so already, configure your upstream MTA (e.g. Postfix) and set up user authentication – just as we described in the first part of the book. Even if our storage format has switched from Maildir or mdbox to obox, the basic features of the system do not change.

Now the main part: if you do not yet have access to an object storage, you first have to set it up on the websites of suitable providers. In this example, we have chosen Amazon S3, where you can also register a certain volume for free for test purposes. On the provider's website, you will receive access data in the form of an access key and possible a secret key (password). Depending on the provider, you may have to set up a named storage on its website – Amazon refers to this name as *Bucket*.

Once you have gathered all your access data, set up the object storage in `/etc/dovcecot/11-obox.conf` as follows:

```
# Mailbox list index is required by obox.
mailbox_list_index = yes

# The ~/obox is for local index files. It's safe to delete the entire
# ~/obox  directory while Dovecot isn't running. The files are
# downloaded back from object storage.
mail_location = obox:%u:INDEX=~/:CONTROL=~/
mail_plugins = $mail_plugins obox

# Directory or file for trusted SSL CA certificates.
ssl_client_ca_dir = /etc/ssl/certs # Debian/Ubuntu
#ssl_client_ca_file = /etc/pki/tls/cert.pem # RedHat
```

Then, depending on the provider, you need to set the access data using the `obox_fs` plugin:

```
### Object Storage configuration to Amazon S3

# Get ACCESSKEY and SECRET from http://aws.amazon.com/ -> My account ->
# Security credentials -> Access credentials.
# Create the BUCKETNAME from AWS Management Console -> S3 -> Create
# Bucket.
plugin {
```

```
#  obox_fs = fscache 100G:/var/lib/dovecot/cache:s3:https://ACCESSKEY:SECR
}

### Object Storage configuration to Microsoft Azure

# Get ACCESSKEY and STORAGENAME from www.windowsazure.com -> Storage
# (-> Create STORAGENAME) -> STORAGENAME -> Manage Keys.
plugin {
#  obox_fs = fscache 100G:/var/lib/dovecot/cache:azure:ACCESSKEY https://S
}

### Object Storage configuration to Dropbox

# You need to register for ACCESSKEY and SECRET by visiting
# https://dropbox.dovecot.fi/
plugin {
#  obox_fs = fscache 100G:/var/lib/dovecot/cache:dropbox:https://ACCESSKEY
}

### OpenStack Swift

plugin {
# obox_fs = fscache 100G:/var/lib/dovecot/cache:swift:https://USERNAME:ACC
}
```

If, as recommended and described in this book, you set up all users under the user ID `vmail` (= 10,000), you still need to configure the following in `10-master.conf` or `11-obox.conf`:

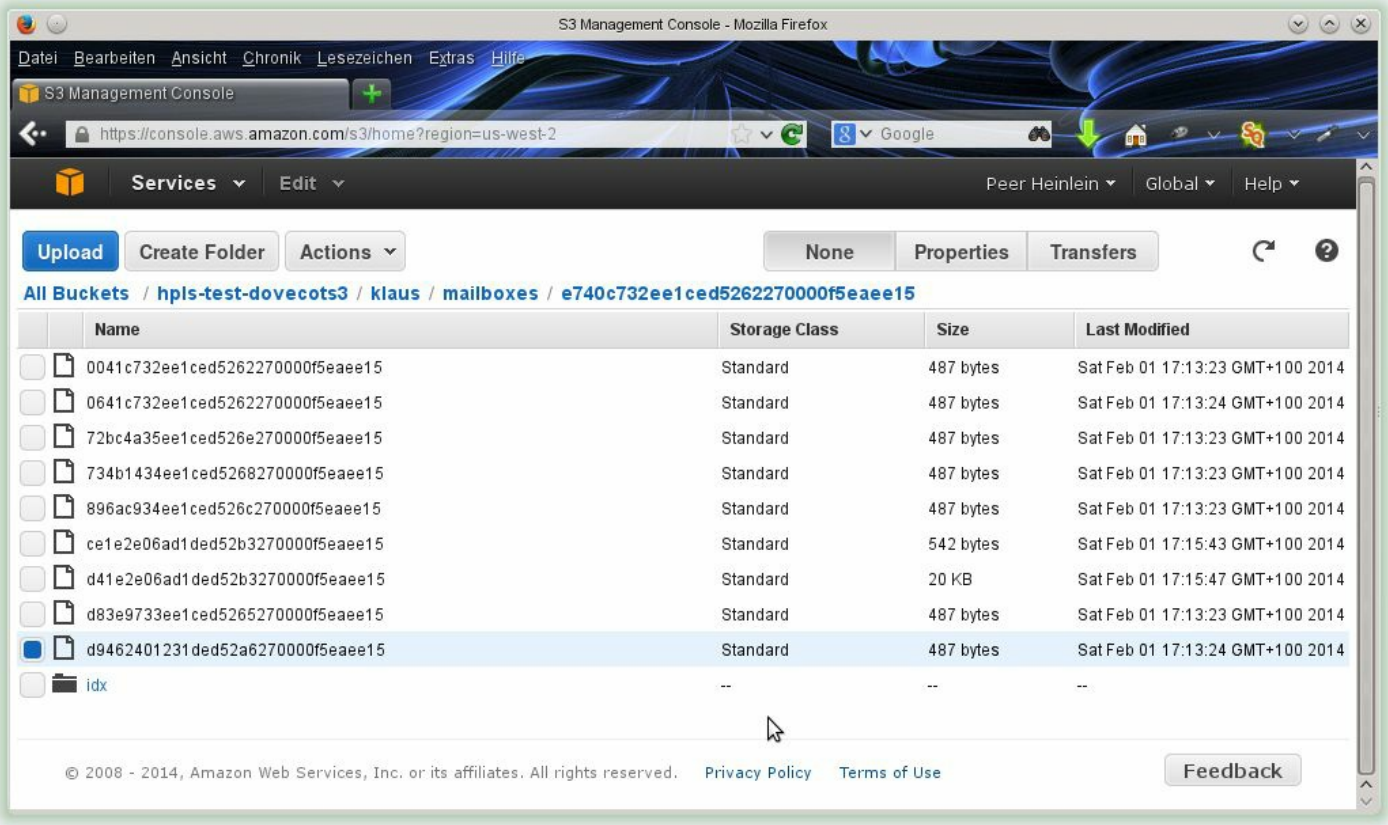
```
service metacache {
  unix_listener metacache {
    user = vmail
  }
}

service metacache-worker {
  user = vmail
}

mail_uid = vmail
```

In addition, you also have to create several directories with suitable file permissions:

```
flash:~ # mkdir /var/lib/dovecot/cache
flash:~ # chown 10000:10000 /var/lib/dovecot/cache/
flash:~ # chmod a+rw /var/run/dovecot/doveadm-server
```

When you now deliver a test email, which is transferred to Dovecot from Postfix using LMTP, it should be saved without difficulty. On the web interface of Amazon S3, the first directories will now appear in your Bucket (Figure 20.2).

Figure 20.2. Use the Amazon S3 web interface to view your stored objects.



If you configure `mail_debug=yes` in `/etc/dovecot/10-logging.conf`, Dovecot is far more chatty when saving the emails: you will see the search for the storage server and the access to the storage system. Here is an abbreviated example:

```
Feb  1 17:22:35 dovecotdebian dovecot: lmtp(10300, klaus): Debug: http-cli
Feb  1 17:22:35 dovecotdebian dovecot: lmtp(10300, klaus): Debug: SSL:
elliptic curve secp384r1 will be used for ECDH and ECDHE key exchanges
Feb  1 17:22:35 dovecotdebian dovecot: lmtp(10300, klaus): Debug: http-cli
klaus/mailboxes/e740c732ee1ced5262270000f5eaee15/8596900d4b1fed523c280000
f5eaee15]: Submitted
Feb  1 17:22:35 dovecotdebian dovecot: lmtp(10300, klaus): Debug: http-cli
ed5262270000f5eaee15/8596900d4b1fed523c280000f5eaee15] (urgent)
Feb  1 17:22:35 dovecotdebian dovecot: lmtp(10300, klaus): Debug: http-cli
ed5262270000f5eaee15/8596900d4b1fed523c280000f5eaee15]
Feb  1 17:22:35 dovecotdebian dovecot: lmtp(10300, klaus): Debug: http-cli
klaus/mailboxes/e740c732ee1ced5262270000f5eaee15/8596900d4b1fed523c280000
f5eaee15]: Partially sent payload
```

Externally, Dovecot should behave as it always does. Emails can be saved via LMTP and retrieved using POP3/IMAP; the `doveadm` commands open up the entire world. Only your file system will remain comparatively empty, as the entire storage is shifted to the internet.

Unusual things are now going on in the background:

- As soon as a user logs on, Dovecot loads the user's index meta data from the object storage and from then on stores them on the IMAP server the user logged on to. Local

file access is still the fastest.

- If you have several servers active in parallel, you should use the Dovecot director to ensure that logins for a user name are always connected to the identical target server (see Section 16.3.1) in order to prevent contradictory changes to the index data.
- Even in long-running IMAP connections, Dovecot will from time to time upload local changes to meta data to the cloud storage, just to be on the safe side. You can configure this in `11-obox.conf`:

```
# How often to upload locally changed index files back to object stora
# When user logs out this is done immediately.
#metacache_upload_interval = 5 mins
```

You should think very carefully about privacy concerns when deciding whether and how to swap out email data to a storage provider. It is legitimate to wonder how much you can trust Amazon, Dropbox or Microsoft…

On the other hand, you are not limited to these providers. In-house local object storage is becoming ever more popular, because such setups resolve the scaling problems of traditional file systems while still keeping the data in-house. Unfortunately, Dovecot does not currently offer native support for Ceph/Rados, but you can use the S3 interface, for example, to address your own Ceph storage in the same way as Amazon S3.

Finally, there are two important parameters for tuning obox:

`metacache_max_space = 123 GB`
: specifies the size that locally stored index files may not exceed.

`metacache_close_delay = 1 min`
: After logout, locally stored index files would immediately be backed up to the object storage. However, that behavior is not great if a user connects to the server after a very short disconnect. This parameter delays backup in order to see if the user wants to log in again immediately. If you operate the obox format behind a Dovecot director system, `metacache_close_delay` must always be smaller than `director_user_expire`.

20.3. Future prospects

Dovecot is currently developing the *abox* format as well as the *obox* format. This feature had not been published when this book went to press, but it is supposed to allow storage of large email archives that can still be searched and accessed efficiently. Single-instance storage of attachments saves storage space, and archives can also be stored in a cloud provider's object storage.

In order to protect data from unauthorized access by third parties, email data are encrypted by an *archive writer* before saving, and decrypted by the *reader* when they are read. You can then safely swap out the email data to a cloud provider. The required keys are loaded from a key storage individually for each domain.

And, if things currently on the horizon come true, the Dovecot family will soon have another member. An SMTP server is likely to be added to the family. Permanently integrated in Dovecot, it will offer fewer configuration options than Postfix or Exim, but it will also need far fewer configuration options. Receipt and sending of emails for SASL-authenticated users, or even the simple receipt of emails as an SMTP gateway to storage in Dovecot, will be easy to set up in one go.

Finally, there are plans to make communication between two providers using Dovecot more trustworthy. We will await these developments with interest.

Part III. Appendix

Appendix A. doveadm – the Finnish army knife

`doveadm` is an all-round tool for working with Dovecot, as you have been able to see throughout the book.

Chapter 14 deals with most of the commands and also describes how to operate `doveadm`.

This chapter is a brief reference of the `doveadm` commands that could not be included in any of the other chapters. If a command has already been discussed in detail in this book, you will find a cross reference.

The following generic options apply for all commands:

`-f <format type>`
: influences the output format. Available format types:

`flow`
: outputs values as `key=value`.

`pager`
: outputs values as `key:value`.

`tab`
: outputs a table header, values are separated by tabs.

`table`
: outputs a table header, values are indented with spaces.

`-v` activates detailed output, including a progress bar if applicable. `-D` activates debut output. Here are all the commands that were available when this book when to print:

`altmove [-u <user>|-A] [-r] <search query>`
: moves the emails that match the `<search query>` to the alternative storage of mdbox (Section 7.3.4).

`auth cache flush`
: deletes the entries in the authentication cache.

`auth test [-a <auth socket path>] [-x <auth info>] <user> [<password>]`
: makes it possible to perform a simple verification of the user name and password in the console (Chapter 5). Before Dovecot 2.1, this command was called `doveadm auth`.

`auth lookup [-a <auth socket path>] [-x <auth info>] <user> [<password>]`
: makes it possible to perform a simple query regarding a user's Passdb extra fields in the console (Section 5.12).

`backup [-u <user>|-A] [-dfR] [-l <secs>] [-m <mailbox>] [-n <namespace>] <dest>`
: writes a user's mailbox (or individual IMAP folders or their namespaces) as a file export to the destination `<dest>` (Section 14.4).

`batch [-u <user>|-A] <sep> <cmd1> [<sep> <cmd2> [..]]`
: performs multiple `doveadm` commands separated by colons:

```
doveadm batch -A : quota get : mailbox status -t all '*'
doveadm batch -A : altmove savedbefore 1w : purge
```

`config [doveconf parameter]`
: the equivalent of calling the command `doveconf` (Section 4.1).

`copy [-u <user>|-A] <destination> [user <source user>] <search query>`
: like `doveadm move`, it copies a user's messages that match the `<search query>` to a different IMAP folder (Section 14.3.5).

`deduplicate [-u <user>|-A] [-m] <search query>`
: searches all messages that match the `<search query>` for duplicate emails and deletes the more recent copy. This is a method for cleaning a mailbox, for example, that has accidentally had the same import twice.

`director add|dump|flush|map|move|remove|ring|ring|ring|status`
: manages the Dovecot directors (Section 16.3).

`dump [-t <type>] <path>`
: extracts an index file to a readable format.

`exec <binary> [binary parameters]`
: allows individual Dovecot modules to be started by hand from `/usr/lib/dovecot`.

`expunge [-u <user>|-A] [-d] <search query>`
: marks a user's messages that match the `<search query>` as deleted.

`fetch [-u <user>|-A] <fields> <search query>`
: exports parts or entire messages that match the `<search query>` (Section 14.3.4).

`flags [-u <user>|-A] add|remove|replace <flags> <search_query>`
: reads, deletes or replaces IMAP flags for messages that match the `<search query>`.

`force-resync [-u <user>|-A] <mailbox mask>`
: forces a repair of the index file of a mailbox. See Section 14.6.

`fs copy|delete|get|iter|iter-dirs|put|stat`
: offers debugging options for mail storages in the `obox` format, so in an object storage in the Enterprise version of Dovecot (Section 20.1).

`help <cmd>`
: outputs help on the `doveadm` in question.

`import [-u <user>|-A] [-s] <source mail location> <dest parent mailbox> <search query>`
: imports the messages that match the `<search query>` to the specified file directory (Section 14.5).

`index [-u <user>|-A] [-q] [-n <max recent>] <mailbox mask>`
: re-indexes the specified IMAP folder, for example if emails were saved in the Maildir structure past LMTP or `dovecot-lda`. Dovecot performs this task automatically the next time the user logs in.

`instance list|remove`
: in multi-instance setups, lists all existing instances or removes one instance from the configuration.

`kick [-a <anvil socket path>] [-f] <user mask>[|]<ip/bits>`
: logs out all users that match the specified user name patter (so `'klaus*'` is just as permissible as `'*'`). You can also specify an IP address or sub-network definition in order to terminate all the connections from this part of the network.

`log errors|find|reopen|test`

shows which log files are configured in Dovecot and issues test log file messages in order to debug their subsequent processing. A `doveadm log errors` shows the last 1000 error messages from Dovecot.

`mailbox create|delete|list|mutf7|rename|status|subscribe|unsubscribe`
manages a user's IMAP folders (Section 14.2).

`mount add|list|remove`
allows the mount points monitored by Dovecot to be added, listed or deleted (Section 8.1.3).

`move [-u <user>|-A] <destination> [user <source user>] <search query>`
like `doveadm copy`, it moves a user's messages that match the `<search query>` to a different IMAP folder (Section 14.3.5).

`penalty [-a <anvil socket path>] [<ip/bits>]`
shows which hosts are currently subject to a penalty, i.e. a delayed login, because of login errors.

`proxy kick|list`
lists the current proxy connections of the users in a partitioned cluster or terminates their connections (Section 16.2).

`purge [-u <user>|-A] [-S <socket_path>]`
actually removes the deleted messages from the m-files in the mdbox format (Section 7.3.3).

`pw [-l] [-p plaintext] [-r rounds] [-s scheme] [-t hash] [-u user] [-V]`
generates password hashes (Section 5.10.5).

`quota get|recalc`
displays or recalculates a user's quota (Section 11.10).

`reload`
reloads the configuration during operation.

`replicator remove|replicate|status`
allows management of real-time replication (Section 16.4.4).

`search [-u <user>|-A] <search query>`
finds the emails that match the `<search query>` and displays the UID and GUID of these emails (Section 14.3).

`sis deduplicate|find`
manages the deduplication of attachments if single-instance storage is configured (Section 15.6).

`stats dump|top`
allows access to utilization statistics.

`stop`
stops Dovecot.

`sync [-u <user>|-A] [-S <socket_path>] [-dfR] [-l <secs>] [-m <mailbox>] [-n <namespace>] [-x <exclude>] [-s <state>] <dest>`
synchronizes a user's mailbox bi-directionally with the data in `<dest>` (Section 14.4).

`user [-a <userdb socket path>] [-x <auth info>] [-f field] [-m] <user mask> [...]`
outputs the results of a user's `userdb` query (Chapter 5).

`who [-a <anvil socket path>] [-1] [<user mask>] [<ip/bits>]`
displays all the logged-in users (or all those that match `<user mask>` or `<ip/bits>`).

`zlibconnect <host> [<port>]`
makes it possible to debug the zlib compression of an IMAP session (only interesting for

developers).

Appendix B. IMAP command reference

IMAP is not just a complex protocol; its definition is just as complex. Next to version 4rev1 of RFC 3501,[90] which defines IMAP and which has been valid since March 2003, there are countless other RFCs and internet drafts with *IMAP extensions*.

This variety is not surprising, because the IMAP protocol laid the foundation with the `CAPABILITY` command for servers to offer optional extensions in a flexible manner that the developers of IMAP never dreamed of.

These countless extensions unfortunately go beyond the scope of this reference, particularly as the support offered by IMAP clients is often incomplete. There are some great ideas for extensions: `URLAUTH` (RFC 4467[91]) enables you to specify the IMAP server, login data, IMAP folder and messages in the style of `imap://tux@example.com/INBOX/;uid=425`, so that references to individual messages can easily be transmitted to third parties or referenced. Other RFCs expand on existing standards: RFC 4731[92] equips the `SEARCH` command with additional functions.

The following reference introduces `IMAP4rev1` as well as the most important IMAP extensions.

Not every IMAP command is allowed at any time during an IMAP session. We have therefore sorted the commands by the states of an IMAP connection described in Section 3.2.2.

IMAP is not case sensitive when it comes to commands and subcommands.

Length is always measured in *octets* in the IMAP protocol, so units consisting of eight bits. As nearly all current systems use eight bits per byte, the two terms can be used synonymously. 120 octets are now the same as 120 bytes.

B.1. IMAP commands available at any time

`CAPABILITY`

queries the capabilities of the server. The server returns an *untagged* response line beginning with `CAPABILITY IMAP4rev1`, in which it lists the options available in the current connection state.

```
a001 CAPABILITY
* CAPABILITY IMAP4rev1 LITERAL+ SASL-IR LOGIN-REFERRALS ID ENABLE IDLE
a001 OK Pre-login capabilities listed, post-login capabilities have mo
```

`LOGINDISABLED` informs the client that the server is currently not prepared to accept an authentication request from this client. This can be the case, for example, if authentication is only permitted in encrypted mode and the client has not yet sent a `STARTTLS`.

`NOOP`

Like in many other protocols, this stands for “no operation”. With this command, the client can keep the connection open and reset any autologout timer:

```
a002 NOOP
a002 OK NOOP completed
```

Many servers return the current message status as part of the `NOOP` response and thereby provide information on new emails:

```
a003 NOOP
* 22 EXPUNGE
* 23 EXISTS
* 3 RECENT
* 14 FETCH (FLAGS (\Seen \Deleted))
a003 OK NOOP completed
```

`LOGOUT`

The client uses this command to initiate the end of the session; the server then sends and OK and terminates the connection:

```
a004 LOGOUT
* BYE logging out
a004 OK Logout completed
```

B.2. Commands in the not-authenticated state

AUTHENTICATE

Initiates the authentication of the client. Depending on the authentication method, the client and server then exchange additional information. In challenge-response processes, the server sends the session key (*challenge*) in a line marked with a +, and the client then uses the challenge, the user name and the password to calculate the matching login string (*response*), see Section 5.10.

```
a001 AUTHENTICATE CRAM-MD5
+ PDUwNjZGNEVFNDNGM0NCQzIzODI1MEVERTc3ODg4Qjg4QGtqaWRkZXI+
cC5oZWlubdVpbiBlZjlkZdQ5YjIxYzk3ZekzNzQ4MzUhMmQ2NDYzZjlhOA==
a001 OK LOGIN Ok.
```

LOGIN

requests a simple plain text login that every server must support in line with RFC. For safety reasons, it should only be possible in SSL/TLS mode, because the user name and password are transmitted unprotected and can therefore be easily overheard:

```
a001 LOGIN tux secret
a001 OK LOGIN completed
```

STARTTLS

The client uses this command to initiate the switch to SSL/TLS-encrypted communication. The original connection continues to be used:

```
a001 CAPABILITY
* CAPABILITY IMAP4rev1 LITERAL+ SASL-IR LOGIN-REFERRALS ID ENABLE IDLE
a001 OK CAPABILITY completed
a002 STARTTLS
a002 OK Begin TLS negotiation now
```

The server has the ability to return other CAPABILITY feedback in an SSL/TLS-encrypted connection. It can then be more generous with the login mechanisms and permit plain text processes in the SSL/TLS tunnel.

B.3. Commands in the authenticated state

`SELECT`

Selects a folder to which all message commands will relate. In response to this command, the server returns several untagged lines in an unspecified order in which he informs the client about the status of the folder:

```
a005 SELECT INBOX
* FLAGS (\Answered \Flagged \Deleted \Seen \Draft)
* 172 EXISTS
* 1 RECENT
* OK [UNSEEN 12] Message 12 is first unseen
* OK [UIDVALIDITY 3857529045] UIDs valid
* OK [UIDNEXT 4392] Predicted next UID
* OK [PERMANENTFLAGS (\Deleted \Seen \*)] Limited
a005 OK [READ-WRITE] SELECT completed
```

In response to the `FLAGS` command, the server tell you which flags it will save for this folder during the IMAP session (see Section 3.2.3). `OK [PERMANENTFLAGS (flag1 flag2 <...>)]` lists all the flags that the client may change permanently. If the server does not provide this information, the client can assume that it may manipulate all flags during this session but that the server will not save these changes.

The number preceding the word `EXISTS` describes how many messages the folder contains, while `RECENT` follows the number of messages marked as *Recent*, i.e. the new emails received since the last login, which this client will be the first to see. in responde to `OK [UNSEEN <number>]` the server returns not the number of unread messages, but the sequential number of the first unread message.

`OK [UIDNEXT <unique-id>]` informs the client of the next UID, while `OK [UIDVALIDITY <unique-id-value>]` returns the currently valid unique ID value (see Section 3.2.1). With the tagged response `<tag> OK [READ-WRITE] SELECT complete` the server states that it has completed the `SELECT` command.

If the client has write access to the folder, the server *should* add the information `[READ-WRITE]` to the `OK` response. If the client has only read permissions, the server *must* output `[READ-ONLY]`.

`EXAMINE`

corresponds to the `SELECT` command, but the client selects the specified folder only to *read* it. The server provides the same response as to the `SELECT` command, but logically specifies `[READ-ONLY]` in the concluding `OK` line:

```
a006 EXAMINE Test
```

```
* 17 EXISTS
* 2 RECENT
* OK [UNSEEN 8] Message 8 is first unseen
* OK [UIDVALIDITY 3857529045] UIDs valid
* OK [UIDNEXT 4392] Predicted next UID
* FLAGS (\Answered \Flagged \Deleted \Seen \Draft)
* OK [PERMANENTFLAGS ()] No permanent flags permitted
a006 OK [READ-ONLY] EXAMINE completed
```

Access via `EXAMINE` may not cause new messages to lose their `\Recent` flag, because that would also be a (server-side) write modification of the directory.

LIST

In response to the `LIST` command, the client receives a list of all the directories it can reach. The server also returns the folder attributes and the applicable *hierarchy delimiter* in its untagged responses.

```
a016 LIST "" "*"
* LIST (\HasNoChildren) "." "INBOX.Personal.Vacation"
* LIST (\HasNoChildren) "." "INBOX.Personal.Orchestra"
* LIST (\HasChildren)   "." "INBOX.Personal"
* LIST (\HasChildren)   "." "INBOX.Book stuff"
* LIST (\HasNoChildren) "." "INBOX.Book stuff.LPIC-1"
* LIST (\HasNoChildren) "." "INBOX.Book stuff.Postfix 4"
* LIST (\HasNoChildren) "." "INBOX.Book stuff.Snort"
* LIST (\HasNoChildren) "." "INBOX.Books stuff.Dovecot"
* LIST (\Unmarked \HasChildren) "." "INBOX"
a016 OK LIST Completed
```

The `LIST` command knows two parameters. The second consists of the mailbox name or a corresponding wildcard pattern. The first is called *reference* and supplies the context in which the mailbox name is interpreted – in relation to the reference:

```
a017 LIST "INBOX.Personal" "*"
* LIST (\HasNoChildren) "." "INBOX.Personal.Vacation"
* LIST (\HasNoChildren) "." "INBOX.Personal.Orchestra"
a017 OK LIST completed
```

The reference is particularly interesting if the IMAP server also allows access to a file system or news server in line with RFC 3501. As neither Dovecot, Courier or Cyrus permit such access, this topic would go beyond the scope of this book.

In normal email operation, the reference argument simply remains empty. The mailbox name corresponds precisely to the folder name as it is used in the `SELECT` command (see that section).

The `*` wildcard stands for any character. `%`, on the other hand, matches all characters with the exception of the hierarchy separator:

```
a018 LIST "" "INBOX.Personal*"
* LIST (\HasNoChildren) "." "INBOX.Personal.Vacation"
```

```
* LIST (\HasNoChildren) "." "INBOX.Personal.Orchestra"
* LIST (\HasChildren) "." "INBOX.Personal"
a018 OK LIST completed
a019 LIST "" "INBOX.Personal%"
* LIST (\HasChildren) "." "INBOX.Personal"
a019 OK LIST completed
```

If you specify an empty mailbox name as a second argument, the `LIST` command returns only the hierarchy separator:

```
a020 LIST "" ""
* LIST (\Noselect) "." ""
a020 OK LIST completed
```

CREATE

Creates a new folder on the server. In the case of subdirectories, the client must specify the full path – including the hierarchy separator specified by the server (in this example: a point).

```
a021 CREATE PERSONAL
a021 OK CREATE completed
a022 CREATE PERSONAL.FRIENDS.VACATION
a022 OK CREATE completed
```

If the layers above do not exist, the server must create them automatically as a hierarchy level. These automatically generated folders cannot contain any emails and cannot be selected using the `SELECT` command. They therefore appear in the list with the `\Noselect` flag. If the client does not create them explicitly as separate folders, it is possible to save emails in them.

The server can decide whether a `Personal` folder can exist next to a `PERSONAL` folder. Only in the `INBOX` is it prohibited to distinguish between upper and lower case letters.

Unfortunately the client cannot just ask which notation the server applies in the respective folder but instead has to use trial and error.

DELETE

deletes the specified directory from the server. If it contains subdirectories (like the `PERSONAL` directory we just created contains the `PERSONAL.FRIENDS.VACATION` subfolder), the server may *not* delete this automatically.

If you delete the `PERSONAL` directory with all the messages stored there, it will appear in the list with the `\Noselect` flag from then on and exist only as an automatic hierarchy folder for the layers underneath.

Logically, directories with the `\Noselect` flag cannot be deleted with `DELETE` because they do not exist independently.

RENAME

Renames a folder on the server. If subfolders exist, their path is also modified. If the superior directories of the new folder do not yet exist, the server will create them automatically.

```
a023 LIST "" *
* LIST () "/" Test
* LIST (\Noselect) "/" foo
* LIST () "/" foo/bar
a023 OK LIST completed
a024 RENAME Test bla
a024 OK RENAME completed
a025 RENAME foo zowie
a025 OK RENAME Completed
a026 LIST "" *
* LIST () "/" bla
* LIST (\Noselect) "/" zowie
* LIST () "/" zowie/bar
a026 OK LIST completed
```

If the INBOX folder is renamed, the INBOX directory *must* be retained as an empty folder; the messages it contains are moved to the new renamed directory. If the INBOX contains subdirectories, these are also retained:

```
a027 LIST "" *
* LIST () "." INBOX
* LIST () "." INBOX.bar
a027 OK LIST completed
a028 RENAME INBOX old-mail
a028 OK RENAME completed
a029 LIST "" *
* LIST () "." INBOX
* LIST () "." INBOX.bar
* LIST () "." old-mail
a029 OK LIST completed
```

SUBSCRIBE

places the specified directory on the list of directories subscribed by the client (or its user) – later on the client can use the LSUB command (see below) to access it in targeted fashion. For the advantages of this procedure, see Section 3.2.4.

The server is allowed to check whether the directory exists at the time of the SUBSCRIBE command. On the other hand, it is not allowed to simply delete directories from a client's subscription list, even if these no longer exist. It is therefore possible to remain subscribed to directories that no longer exist just in case they will exist again in the future.

UNSUBSCRIBE

removes a directory from the list of subscribed directories.

LSUB

The parameters and server responses correspond to those for the `LIST` command (see that section), but `LSUB` (short for “list subscribed”) only returns the directories subscribed by the client.

`STATUS`

allows the targeted querying of status information without the server selecting the directory in question (e. g. like for `EXAMINE`, see that section). The command is particularly useful in determining the status of directories that are currently not selected.

The client specifies in round brackets which parameters of the queried directory it is interested in:

```
a030 STATUS Test (UIDNEXT MESSAGES)
* STATUS Test (MESSAGES 231 UIDNEXT 44292)
a030 OK STATUS completed
```

As `STATUS` queries may require some computation time, RFC 3501 prohibits the `STATUS` command from being applied to directories that are already selected.

`APPEND`

inserts a new message in the current folder. This must be in line with RFC 2822 (i. e. consist of an email header and an email body separated by an empty line). The server must observe the following rules:

- If the client provides an optional date string in the `APPEND` command, the server should accept it as the time this message was received.
- If a folder specified as the destination does not exist, the server may not simply create this directory. Instead, it can respond to the client with `[TRYAGAIN]` to inform it that it needs to create the folder beforehand using `CREATE´.
- If the client transfers flags such as `\Seen` or `\Answered` in round brackets, the server *should* save them accordingly:

  ```
  a031 APPEND saved-messages (\Seen) {310}
  + Ready for literal data
  Date: Sat, 15 Feb 2014 21:52:25 -0800 (PST)
  From: Tux <tux@example.com>

  Hi Klaus, can we meet tomorrow at 3:30 pm?

  a031 OK APPEND completed
  ```

  If this information is not provided, the server should by default not save any flags but instead present the client with the `\Recent` flag. The client must also state in curly brackets how long the email is in bytes (here: 310 bytes) so that the IMAP server can detect the end of the transmission.

- If the message creation fails for any reason, the server must fully restore the previous state of the directory as if the entire action has never taken place.

B.4. Commands in the Selected state

`CHECK`

provides programmers with an opportunity to debug their own IMAP implementation. It does not return a specified return value. Therefore the software is free to answer with any debug output – or to simply return `OK` and not perform any action. Clients should always use `NOOP` to keep the connection open.

`CLOSE`

deletes all emails flagged as `\Deleted` in one folder. The respective folder is then deselected. The IMAP connection returns to the *Authenticated* state.

If the client choses a different folder via `SELECT` or `EXAMINE` or logs out via `LOGOUT`, the server implicitly executes a `CLOSE` so that all emails marked as `\Deleted` are deleted.

`EXPUNGE`

also deletes all emails marked as `\Deleted`, but it keeps the folder in the `Selected` state and also returns an untagged message for each deleted email.

Be careful, deleting emails also changes the sequential numbers of the emails. Since the server removes each email individually, it renumbers them before every delete command to close the gap. If you delete emails 3, 4, 7 and 11 as seen below, the server will return the IDs of emails 3, 3, 5 and 8:

```
a032a STORE 3,4,7,11 +FLAGS (\Deleted)
* 3 FETCH (FLAGS (\Deleted \Seen))
* 4 FETCH (FLAGS (\Deleted))
* 7 FETCH (FLAGS (\Deleted \Flagged \Seen))
* 11 FETCH (FLAGS (\Deleted \Seen))
a032a OK STORE completed
a032b EXPUNGE
* 3 EXPUNGE
* 3 EXPUNGE
* 5 EXPUNGE
* 8 EXPUNGE
a032b OK EXPUNGE completed
```

If the client does not require the list of deleted emails, it should choose the `CLOSE` command over the `EXPUNGE` command.

`SEARCH`

orders the server to search in all emails in the selected folder. The server returns an untagged list of all sequential numbers of the emails that match the search criteria. There are once again a few rules:

- Text search never differentiates between upper and lower-case letters (case

insensitive).
- A text search is successful as soon as the search pattern appears as a partial string in the email.
- Search criteria are always linked by a logical AND unless the client explicitly requires an OR.
- The search pattern can be a list of several search terms combined in brackets if the logical combination with `OR` or `NOT` requires this.

The order of the search criteria is irrelevant.

The following specifications are possible:

`<sequential number1>,<sequential number2>,...`
: if you specify the sequential number(s) of the emails to be searched, the server will restrict its search to these emails. You can use lists (`24,90,30`) as well as ranges (`80:100`), and you can also combine them:

```
a033 search TEXT "yesterday" 24,90,30,80:100
* SEARCH 24 90
a033 OK done
```

Here the word `yesterday` appears in emails 24 and 90.

`ALL`
: stands for all emails in the mailbox.

`ANSWERED`
: all emails flagged as `\Answered`.

`BCC <string>`
: emails that contain the specified `<string>` in the `Bcc:` field of the email header.[93]

`BEFORE <date>`
: emails with an internal date before `<date>`. The time is not considered. `<date>` must be specified in the format `29-Sep-2013`.

`BODY <string>`
: emails that contain the specified `<string>`somewhere in the body:

```
a034 SEARCH BODY "last night"
* SEARCH 218 587 1232 1421 2258 3696 4123
a034 OK SEARCH done.
```

There is an alternative syntax for this command:

```
a034 SEARCH BODY {13}
+ OK
last night
* SEARCH 218 587 1232 1421 2258 3696 4123
a034 OK SEARCH done.
```

If you specify the length of the character string you are searching for in bytes (here `10`) and complete the command with a line break (CRLF), the server will ask for the search string. You can then enter it without protecting quotation marks. That way you can search

for umlauts and other characters with byte values larger than 127 that are not valid in the command itself. Here the character encoding has to be specified using `CHARSET`:

```
a035 SEARCH CHARSET iso-8859-1 BODY {7}
+ OK
München
4412 4416 4420 4427 4429 4430 4434 4435 4438 4440
a035 OK SEARCH done.
```

searches for the string `München` with a length of `7` bytes (in this encoding) in all email parts that are encoded according to ISO-8859-1.

`CC <string>`
: the emails that contain the specified `<string>`in the `Cc:` field of the email header.

`DELETED`
: all emails flagged as `\Deleted`.

`DRAFT`
: all emails flagged as `\Draft`.

`FLAGGED`
: all emails marked as `\Flagged`.

`FROM <string>`
: emails that contain `<string>`in the `From:` field of the email header.

`HEADER <field name string>`
: emails with the specified header field (according to RFC 2822) that contain the character string `<string>`:

```
036 SEARCH HEADER X-Virus-Scanned amavisd
* SEARCH 90 194
036 OK done
```

If `string` is empty, this query relates to all emails that include the header field, regardless of the content:

```
037 SEARCH HEADER X-Virus-Scanned ""
* SEARCH 24 29 90 98 194
037 OK done
```

`KEYWORD <flag>`
: emails that contain the specified flag.

`LARGER <n>`
: emails that are larger than `<n>` bytes. A line break always corresponds to exactly *two* bytes, because RFC 2822 dictates a CR/LF.

`NEW`
: all emails that are flagged as `\Recent` but not as `\Seen`.

`NOT <searchoption>`
: emails that do *not* match the search option:

```
a038 SEARCH FLAGGED BEFORE 1-Jan-2013 NOT FROM "geeko"
* SEARCH 3 6 7 9 10 11 12 15 16  20 21 22 24 25  29 31 32 33
a038 OK SEARCH completed.
```

OLD
: all emails not flagged as \Recent.

ON <date>
: emails with an internal date that is <date>. The time is not considered.

OR <searchoption1> <searchoption2>
: emails that match <searchoption1> or <searchoption2>:

```
039 SEARCH OR FROM tux@ FROM paul@
* SEARCH 25 29 31 32 33 55 64
039 OK done
040 SEARCH (OR FROM tux@ FROM paul@) BEFORE 1-Jan-2013
* SEARCH 25 29 31 32 33
040 OK done
```

RECENT
: all emails flagged as \Recent.

SEEN
: all emails flagged as \Seen.

SENTBEFORE <date>
: emails of which the date specified in the Date: header is before <date>. The time is not considered.

SENTON <date>
: emails where the Date: header contains the <date>. The time is not considered here either.

SENTSINCE <date>
: emails where the Date: header contains a date after <date>. The time is not considered.

SINCE <date>
: emails whose internal date stamp is set to <date> or later. The time is not considered.

SMALLER <n>
: emails that are smaller than <n> octets.

SUBJECT <string>
: emails that contain <string> in the Subject: field of the email header.

TEXT <string>
: emails that contain <string> in header or body:

```
a041 SEARCH TEXT "vacation"
* SEARCH 4 23
a041 OK SEARCH completed.
```

  TEXT is used to search in raw data, where umlauts are normally encoded as =FC or =C3=BC (in case of ü) – at least on servers that behave according to the standards. It therefore makes sense to use the SEARCH subcommand BODY (see there) instead.

TO <string>
: emails that contain <string> in the To: field of the email header.

UID <uid1>,<uid2>,<...>
: emails with a unique identifier that corresponds to one of the numbers specified as an argument.

UNANSWERED
: all emails not flagged as \Answered.

`UNDELETED`
: all emails not flagged as `\Deleted`.

`UNDRAFT`
: all emails not flagged as `\Draft`.

`UNFLAGGED`
: all emails not marked as `\Flagged`.

`UNKEYWORD <flag>`
: all emails *not* marked with the specified flag.

`UNSEEN`
: all emails not flagged as `\Seen`.

`FETCH`
: requests the specified message(s). You can enter specific key words at the end so the client retrieves only specific parts of a message. You can do this, for example, if you only want the client to create the index; in that case, it only needs to retrieve the email headers and not the email body. `FETCH` knows the following key words:

`BODY` or `BODYSTRUCTURE`
: retrieves the MIME structure of the message:

```
a042 FETCH 8 BODYSTRUCTURE
* 8 FETCH (BODYSTRUCTURE (("text" "plain" ("charset" "iso-8859
-1") NIL NIL "quoted-printable" 721 22 NIL ("inline" NIL) NIL)
("image" "jpeg" ("name" "image1_anse_gaulettes1.jpg") NIL NIL))
a042 OK FETCH completed.
```

This example email contains 22 lines and is 721 octets (i.e. bytes) in length. `NIL` stands for “not in list” or “nothing”, depending on the interpretation, and serves here as a wildcard for empty fields in the MIME structure.

`BODY[]` or `RFC822`
: retrieves the specified email or emails in its or their entirety. Whenever there is access via one of these two commands, the server automatically sets the `\Seen` flag. If you want to prevent that, use `BODY.PEEK[]` instead of `BODY[]`; it is familiar with all the extensions described below.

`BODY[HEADER]`
: retrieves the entire header of a message in line with RFC 2822:

```
a043 FETCH 30 BODY[HEADER]
* 30 FETCH (BODY[HEADER] {1458}
Return-Path: <p.heinlein@heinlein-support.de>
X-Original-To: tux@example.com
Delivered-To: tux@example.com
[...]
Message-Id: <201306062304.5047.p.heinlein@heinlein-support.de>

)
a043 OK FETCH completed.
```

The untagged response line contains the length of the server response in curly brackets (in this case `1458` octets or bytes). `BODY.PEEK[HEADER]` has a synonym, `RFC822.HEADER`.

`BODY[HEADER.FIELDS (<field1> <field2> <...>)]`
retrieves only the specified fields from the header of the message:

```
a044 FETCH 30 BODY[HEADER.FIELDS (Message-ID)]
* 30 FETCH (BODY[HEADER.FIELDS ("Message-ID")] {67}
Message-Id: <201306062304.5047.p.heinlein@heinlein-support.de>

)
a044 OK FETCH completed.
a045 FETCH 30 BODY[HEADER.FIELDS (Message-ID Date)]
* 30 FETCH (BODY[HEADER.FIELDS ("Message-ID" "Date")] {105}
Date: Wed, 6 Jun 2013 23:04:04 +0200
Message-Id: <201306062304.5047.p.heinlein@heinlein-support.de>

)
a045 OK FETCH completed.
```

`BODY[HEADER.FIELDS.NOT (<field1> <field2> <...>)]`
returns the header *without* the specified header fields.

`BODY[<level>.MIME]`
complex messages consist of multiple parts: the actual email text, several binary attachments or even a second message that the sender has attached. All these parts are held together by the MIME structure.

You can retrieve the technical details of the various encapsulated levels of the message by means of key word `MIME`. The actual text of the email is on level 1:

```
a046 FETCH 8 BODY[1.MIME]
* 8 FETCH (BODY[1.MIME] {127}
Content-Type: text/plain;
  charset="iso-8859-1"
Content-Transfer-Encoding: quoted-printable
Content-Disposition: inline

)
a046 OK FETCH completed.
```

The first attachment forms level 2:

```
a047 FETCH 8 BODY[2.MIME]
* 8 FETCH (BODY[2.MIME] {159}
Content-Type: image/jpeg;
  name="image1.jpg"
Content-Transfer-Encoding: base64
Content-Disposition: attachment;
        filename="image1.jpg"

)
a047 OK FETCH completed.
```

`BODY[TEXT]` or `RFC822.TEXT`

retrieves the full email content (without the header):

```
a048 FETCH 28 BODY[TEXT]
* 28 FETCH (BODY[TEXT] {389}

Dear Carolin,
dear Antonia,

Surprise! The Dovecot book is almost complete. It has taken a long tim

Just 30 more pages of final corrections, and I hope the editor will no

And then the book can go to print tomorrow.

Isn't that great!

Peer

a048 OK FETCH completed.
```

You can limit the output, e. g. to the first 140 characters:

```
a049 FETCH 28 BODY[TEXT]<0.140>
* 28 FETCH (BODY[TEXT] {200}

Dear Carolin,
dear Antonia,

Surprise! The Dovecot book is almost complete. It has taken a long tim
a049 OK FETCH completed.
```

There may be *no* spaces between the closing square bracket and the opening angle bracket. If you want to view the email body between characters 100 and 200, the content of the angle brackets changes to `<100.200>`. However, `RFC822.TEXT` is not familiar with this syntax extension.

`ENVELOPE`

the server responds to this command by generating a listing of the most important data of the RFC-2822 header of the email that can be used for display in a message overview (subject, date, sender, message ID). This has nothing to do with the SMTP envelope used in the delivery of the email, because this no longer exists once the email has been saved.

```
a050 FETCH 8 ENVELOPE
* 8 FETCH (ENVELOPE ("Tue, 8 May 2013 17:40:02 +0200" "image =
?iso-8859-1?q?for_the_empty?= frame" (("Ivonne Reimann" N
IL "i.reimann" "heinlein-support.de")) (("Ivonne Reimann" NIL
 "i.reimann" "heinlein-support.de")) (("Ivonne Reimann" NIL "
i.reimann" "heinlein-support.de")) ((NIL NIL "tux" "example.c
om")) NIL NIL NIL "<201305081740.03490.i.reimann@heinlein-sup
port.de>"))
a050 OK FETCH completed.
```

`FLAGS`
: lists all the flags set for the specified message(s).

`INTERNALDATE`
: returns the internal date of the message:

```
a051 FETCH 45,46 INTERNALDATE
* 45 FETCH (INTERNALDATE "30-Nov-2013 15:57:48 +0000")
* 46 FETCH (INTERNALDATE "30-Nov-2013 15:57:48 +0000")
a051 OK done
```

`RFC822.SIZE`
: the size of the message in line with RFC 2822 (i.e. with CR/LF as the line end) in bytes:

```
a052 FETCH 42 RFC822.SIZE
* 42 FETCH (RFC822.SIZE 3649)
a052 OK done
```

`UID`
: The unique ID of the message:

```
a053 FETCH 42 UID
* 42 FETCH (UID 43)
a053 OK done
```

This shows clearly that sequential numbers and UIDs do not usually match. It is possible to combine multiple key words in round brackets:

```
a054 FETCH 56 (FLAGS INTERNALDATE RFC822.SIZE)
* 56 FETCH (RFC822.SIZE 3731 FLAGS (\Seen) INTERNALDATE "30-Nov-2013 1
a054 OK done
```

There are abbreviations for combinations that are frequently required with `FETCH`:

`ALL`
: corresponds to `(FLAGS INTERNALDATE RFC822.SIZE ENVELOPE)`

`FAST`
: corresponds to `(FLAGS INTERNALDATE RFC822.SIZE)`

`FULL`
: corresponds to the combination `(FLAGS INTERNALDATE RFC822.SIZE ENVELOPE BODY)`

`STORE`
: adds flags to one or more messages. The server responds with one or more untagged `FETCH` responses in which it combines the flags applicable to the affected messages. The client can forgo this `FETCH` response (e.g. to save transmission volume and time) by using the `STORE` option `FLAGS.SILENT` instead of `FLAGS`; this option is also available in the three versions mentioned below.

`+FLAGS (<flag1> <flag2> <...>)`
: adds the flags specified here:

```
a055 STORE 2:4 +FLAGS (\Deleted)
```

```
* 2 FETCH (FLAGS (\Deleted \Seen))
* 3 FETCH (FLAGS (\Deleted))
* 4 FETCH (FLAGS (\Deleted \Flagged \Seen))
a055 OK STORE completed
```

By the way, custom flags (also referred to as *key words*) are set in the same way as system flags, with the difference that they do not have a leading `\` in their name:

```
a056 STORE 100 +FLAGS (project)
100 FETCH (UID 102 FLAGS (\Seen project))
a056 OK done
```

`-FLAGS (<flag1> <flag2> <...>)`
: removes the flags specified here.

`FLAGS (<flag1> <flag2> <...>)`
: sets all the flags specified here. Existing flags not mentioned here again are deleted without replacement. Exception: the `\Recent` flag is retained.

```
a055 STORE 2:4 FLAGS (\Deleted)
* 2 FETCH (FLAGS (\Deleted))
* 3 FETCH (FLAGS (\Deleted))
* 4 FETCH (FLAGS (\Deleted))
a055 OK STORE completed
```

`COPY`
: copies the specified message(s) to a different folder.[94] The flags of the message *should* be retained, and the server *should* set the `\Recent` flag, as the messages are new in their new location:

```
a057 COPY 2:4 MEETING
a057 OK COPY completed
```

As for the `APPEND` command (see there) the server *should* not simply create missing folders, but instead return a `NO` response with `[TRYCREATE]` to the client to encourage it to create the directory. If the `COPY` command fails for some reason, the server is of course obliged to restore the previous condition of the directory.

`UID`
: This special command can be added in front of the `COPY`, `FETCH` and `STORE` commands to show that these three commands should interpret the numbers specified as arguments not as sequential numbers but as unmodifiable UIDs. While the following command copies the messages with sequential numbers 2 to 4 to folder `INBOX.Personal`

```
a058 COPY 2:4 INBOX.Personal
a058 OK COPY completed
```

the following command copies the messages with UIDs 400 to 403, which does not mean that the folder necessarily contains more than 400 messages:

```
a059 UID COPY 400:403 INBOX.Personal
```

```
a059 OK COPY completed
```

If you want to limit the range of messages to be processed to the largest UID in this folder, use the `*` wildcard: `403:*`. Even if it is smaller than the starting value of 403, the server will return a message as a hit (i.e. the one with the highest UID) unless the folder is completely empty.

The following example demonstrates that UIDs are not in an uninterrupted sequence:

```
a060 UID FETCH 4827313:4828442 FLAGS
* 23 FETCH (FLAGS (\Seen) UID 4827313)
* 24 FETCH (FLAGS (\Seen) UID 4827943)
* 25 FETCH (FLAGS (\Seen) UID 4828442)
a060 OK UID FETCH completed
```

The server simply ignores non-existent UIDs. Even if none of the specified UIDs exists, it returns `OK` even though it has not actually done anything. As sequential numbers are always numbered in uninterrupted sequence, this problem cannot occur with normal `COPY`, `FETCH` and `STORE` commands.

If the client precedes the `UID` command with a `Search` command, it tells the server to return its results in the form of UIDs (and not as sequential numbers). However, the server will continue to interpret the message IDs provided by the client as sequential numbers:

```
a061 UID SEARCH 1:100 FROM "Smith"
* UID SEARCH 80 242 882
a061 OK SEARCH done.
```

Of course the UID is still valid as a search criterion. In the next example, the client searches messages 1 to 100 for emails with a UID larger than or equal to 403. The server provides their UIDs as an untagged response:

```
a062 UID SEARCH 1:100 UID 403:*
* SEARCH 6924 8697 16600 16908 19373 19374
a062 OK SEARCH done.
```

B.5. IMAP extensions

Over time, numerous additional IMAP extensions have been defined, which are unfortunately spread confusingly over various RFCs. Around 50 IMAP extensions are currently registered.[95]

The client uses the `CAPABILITY` command to determine which IMAP extensions the server supports:

```
a CAPABILITY
* CAPABILITY IMAP4rev1 UIDPLUS CHILDREN NAMESPACE THREAD=ORDEREDSUBJECT
THREAD=REFERENCES SORT IDLE AUTH=CRAM-MD5 AUTH=CRAM-SHA1 ACL
a OK CAPABILITY completed
```

Many extensions provide precisely one new command with the same name, while others provide none or several (differently named) IMAP commands. Below is an overview of the most important IMAP extensions:

`UIDPLUS`
: The server sends expanded responses in line with RFC 4315 (previously RFC 2359) that also contain the UIDs of the emails.[96] This allows the client to send fewer queries; in addition, this extension provides the basis for offline IMAP (also referred to as *cached IMAP* or *disconnected IMAP*).

`CHILDREN`
: Server responses specify the attributes `\HasChildren` or `\HasNoChildren` in their directory listings.[97] Once again, the client has to send fewer queries, because it no longer needs to check in each case whether an IMAP folder contains additional subfolders.

`MOVE`
: The server makes it possible to move emails to other folders via `MOVE` or `UID MOVE`. Originally the IMAP protocol allowed only `COPY` and `EXPUNGE`.[98]

`NAMESPACE`
: As the IMAP namespace does not have a fixed definition, the client can use this command to request the supporting name schema of the server.[99]

`SORT`
: The server supports server-side search commands via `SORT`.[100]

`SPECIAL-USE`
: The server supports the mark-up of *special use* IMAP folders such as Trash, Draft or Sent (see Section 4.4.3).[101]

`THREAD=<algorithm>`
: The server supports the `THREAD` search method in the specified version.[102]

`IDLE`
: The server supports push email via the `IDLE` command (see Section 15.4).[103]

`AUTH=<method>`
: returns the login methods supported in addition to `LOGIN` and `PLAIN`.[104]

`ACL` and `RIGHTS`
: support the set-up of ACLs (*Access Control Lists*) via IMAP so that a user can release individual IMAP folders for other users (see Chapter 9).[105]

`QUOTA`
: If quotas are activated for an account, the client can use them to query the permitted upper limit and the current level of occupation (see Chapter 11).[106]

B.6. Experimental commands

Commands beginning with `X` are not defined in standards and count as experimental or proprietary. The server may never send X responses on its own account unless a client has explicitly attempted to use such a command.

If a server supports proprietary X commands, it will provide this information in response to the `CAPABILITY` command.

---

[90] http://www.faqs.org/rfcs/rfc3501

[91] http://tools.ietf.org/html/rfc4467

[92] http://tools.ietf.org/html/rfc4731

[93] Most webmailers delete the `Bcc` field in the email header, because it is used to specify recipients that are not supposed to know about each other. Therefore an INBOX should actually not contain any emails with this header.

[94] You can now move emails with the `MOVE` IMAP extension, see Section B.5.

[95] For an up-to-date overview, see http://www.iana.org/assignments/imap-capabilities/imap-capabilities.xhtml

[96] http://tools.ietf.org/html/rfc4315

[97] http://tools.ietf.org/html/rfc3348

[98] http://tools.ietf.org/html/rfc6851

[99] http://tools.ietf.org/html/rfc2342

[100] http://www.ietf.org/internet-drafts/draft-ietf-imapext-sort-18.txt

[101] http://tools.ietf.org/html/rfc6154

[102] http://www.ietf.org/internet-drafts/draft-ietf-imapext-sort-18.txt

[103] http://tools.ietf.org/html/rfc2177

[104] http://tools.ietf.org/html/rfc2060

[105] http://tools.ietf.org/html/rfc4314

[106] http://tools.ietf.org/html/rfc2087

Appendix C. POP3 command reference

POP3 is also familiar with different but very simple connection states:

Authorization state
: The connection has been set up, but the user is not yet logged in. In this state you can therefore only ask the server for its extensions, request SSL encryption and issue the authentication commands `USER`, `PASS`, `AUTH` and `APOP`.

Transaction state
: The user is logged in and can retrieve his emails – authentication commands are no longer permitted.

Update state
: The client has logged out with `QUIT`. Now the server has to actually remove the messages that were marked as deleted and then terminate the connection.

  If the connection to a client was terminated without `QUIT`, the update state is not reached. Emails marked for deletion therefore remain on the server.

C.1. An overview of all commands

`USER <user name>`
: transmits the user name during login.

`PASS <password>`
: transmits the password in plain text during login.

`STAT`
: delivers the number of emails in the `INBOX` and their total size in bytes.

`LIST`
: delivers a numbered list of all emails and their respective size in octets (today, octets are usually the equivalent of one byte). If you enter one of these numbers as an argument, the server returns the size of the respective email.

`RETR <n>`
: fetches email number `<n>` from the email server.

`DELE n`
: marks(!) email number `<n>` for deletion on the email server.

`NOOP`
: This command without a function (“no operation”) helps the client to keep the connection to the server open even during inactive periods. In addition, it can be used to test whether the connection exists.

`RSET`
: resets all the delete markings from emails set during the current POP3 session, a bit like an “undelete” command.

`QUIT`
: terminates the POP3 session. All emails marked for deletion are deleted.

C.2. Optional commands

Beyond that, there are some commands that not every server has to know:

`AUTH <method>`
: The client can suggest an authentication method to the server. If the server supports it, it enters the corresponding protocol. If it does not support it, it generates an error code:

```
AUTH KERBEROS_V4
-ERR Authentication failed.
```

`APOP`
: Initiates login with an encrypted password.

`CAPA`
: asks the server for the available POP3 extensions:[107]

```
CAPA
+OK
CAPA
TOP
UIDL
RESP-CODES
PIPELINING
STLS
USER
SASL PLAIN LOGIN CRAM-MD5 DIGEST-MD5
.
```

In this case, the server offers both extensions:

`STLS`
: the POP3 `STARTTLS` extension announced by the server as `STLS` provides a command of the same name that the client can use to request TLS encryption in the *authorization state*.[108]

`TOP`
: with `TOP <n> <x>` as its syntax, it retrieves the header and the first `<x>` lines of email number `<n>`.

`USER`
: indicates that plain text login is possible with the `USER` command.

`LOGIN-DELAY <n>`
: informs the client of the minimum number of seconds that must elapse between two logins.

`PIPELINING`
: allows the client to issue multiple POP3 commands directly in a row without waiting for responses from the server. The server must return the responses in the order that the commands were given, or it will not be possible to match them up.

UIDL
: retrieves the unique UID of the email whose number the client specifies as the argument. This may not change even over multiple POP3 sessions and can be used by the client to synchronize the mailbox.

IMPLEMENTATION
: this capability allows the server to identify itself with a personal ID text that is specified as an argument, but it does not provide any new POP3 commands.

---

[107] http://tools.ietf.org/html/rfc2449

[108] http://tools.ietf.org/html/rfc2595

Appendix D. The Sieve script language

Sieve makes it possible to filter emails by a variety of criteria directly on a mail server and to react with a range of actions. It is a script language that is similar in structure to PHP and Perl, and can easily be generated by a user or a Sieve editor.

It is mainly defined in RFC 5228[109] but also has numerous extensions[110], some of which will be described here. This chapter is not a full Sieve reference, but simply an introduction to using Sieve. RFC 5228 provides a comprehensive Sieve reference.[111]

Sieve filters contain conditions that have to be met in order for the action listed in curly brackets to be performed. Each action must be completed with a semicolon:

```
if <conditions> {
  <action1>;
  <action2>;
}
```

A practical example:

```
require "fileinto";
if address :is "from" "test@example.com" {
        fileinto "INBOX/Test";
        stop;
}
```

In this case, the `fileinto` Sieve extension is loaded first. Then all emails from `test@example.com` (the decisive factor is the `From:` header) are moved to the `INBOX/Test` subfolder. As the instruction also ends in `stop;`, filtering is complete for this email; otherwise subsequent filter rules could trigger further actions.

D.1. Linking conditions

In principle, you can also specify Sieve commands directly. However, in practice you will only want to use Sieve commands in specific conditions and therefore embed them in one or more nested if-then structures:

`if (<condition1>, <condition2> <...>) {<action1>; <action2>;}`
: The actions in curly brackets are executed if all specified conditions apply. The simplest application is `if true`, which applies to every email. `if false` is also possible, and will logically apply to no emails at all. That is an easy way to deactivate a rule (temporarily) without deleting it – for example your out-of-office responder.

But all other standard logical links are possible:

`if anyof (<condition1>, <...>) {<action1>; <...>;}`
: The instructions in curly brackets are executed if one of the specified conditions applies.

```
require "fileinto";
if anyof (header :is "From" "listadmin@example.com",
          header :is "To" "user@liste.example.com",) {
    fileinto "INBOX/mailingliste";
}
```

`if allof (<condition1>, <...>) {<action1>; <...>;}`
: as with `if anyof`, just that *all* conditions specified here must apply:

```
if allof (address :is "from" "chef@example.com", size :over 10M ) {
    discard;
}
```

`elsif allof (<condition1>, <...>) {<action1>; <...>;}`
: alternative rules that apply if the `if` branch does not apply. You can use `anyof` as well as `allof`.

`else {<action1>; <action2>; <...>;}`
: instructions that have to be executed if none of the rules defined in the `if` and `elsif` branches applies.

The following example demonstrates how to use `elseif` and `else`:

```
require ["fileinto", "vacation"];
if address :is ["to", "cc"] "paul.meier@example.com" {
  fileinto "INBOX/Important";
}
elsif address :is ["to", "cc"] "paul@example.com" {
  keep;
}
else {
```

```
  vacation :addresses ["admin-team@example.com", "support@example.com"]  "
}
```

Emails containing `paul.meier@example.com` in a header address field are sorted into the `INBOX/Important` email directory. If the address has the value `paul@example.com`, the email remains in the `INBOX`.

In all cases, a `vacation` action is triggered if the recipient or aliases `admin-team@example.com` or `support@example.com` are found in the header recipients.

D.2. Overview of the various Sieve conditions

The following conditions can be used:

`address`
: The `address` test condition always applies when you explicitly want to check address fields in the email header (so `From:`, `To:`, `Cc:` and `Bcc:`).

```
if address :is ["to", "cc"] "klaus@example.com" {
        discard;
}
```

If multiple email addresses are specified, for example a lengthy list of `To:` recipients, `address` will check every email address where the `header` condition would check the recipient list as an explicit line of text.

`envelope`
: If you also want to analyze the `FROM:` and `RCPT TO:` fields from the SMTP or LMTP envelope, you can use the `envelope` Sieve extension.[112] The following filter rule, for example, deletes all emails that are addressed to `spam@example.com` in the envelope:

```
require "envelope";
if envelope :all :is "to" "spam@example.com" {
        discard;
}
```

`exists`
: Checks whether a specific header line exists. To find out whether a specific header is missing, use `if not exists`.

```
if not exists ["From","Date"] {
     discard;
}
```

`header`
: You can use `header` to analyze the content of any header lines (often `Date:` or `Subject:`):

```
require "fileinto";
if header :contains "Subject" "MEMO:" {
        fileinto "INBOX/Memos";
}
```

`size`
: If you want to filter by the size of emails with their attachments, use `size`. For example, the rule `if size :over 10M` applies to all emails that are larger than 10 MB. For KB

and GB, enter `K` or `G` instead of `M`; if you want to filter by bytes, enter only the numeric value. Use `:under 1M` to look for emails that are smaller than 1 MB.

Unfortunately it is not possible to query directly whether the size of the email matches a specific value *precisely*, but you could combine a larger than query and a smaller than query to one query:

```
if allof(not size :under 10M, not size :over 10M) {
  [...]
}
```

D.3. Match types for header and address

For the `header` and `address` conditions, contents can be compared in different ways:

`:is`
: The header content or the individual email address must match the specified pattern exactly.

`:contains`
: The header content or the individual email address must contain the specified pattern.

`:matches`
: The header content or the individual email address must match the wildcard pattern. The following rule will delete all the emails where the `To:` or `Cc:` header contains an email address with the user part `finances`, any domain part and the top level domain `.com`:

```
if header :matches [ "To", "Cc" ] "finances*.com" {
        discard;
}
```

`:regex`
: The header content or the individual email address must match the specified regular expressions (regex):

```
require "regex";
if header :regex "Subject" "^\*\*\*EROTIC" {
        discard;
}
```

Comparators determine whether the pattern comparison is case sensitive or not.

`:comparator "i,ascii-casemap"` (default)
: treats upper and lower case identically and is active by default.

`:comparator "i,octet"`
: compares the specified string exactly and does not tolerate upper or lower case deviations:

```
if header :contains :comparator "i;octet" "Subject"
      "MAKE MONEY FAST" {
      discard;
}
```

D.4. Sieve actions

Now we get to the pleasant part of the Sieve reference. Let's take a look at the available actions. After all, they are what all this is about.

Once again, some actions are available as *Sieve extensions* and have to be loaded beforehand in the generic section of the Sieve script by means of a `require` command. That applies in particular to the actions `fileinto`, `reject`, `vacation` and `envelope`. Multiple plugins are specified by means of multiple `require` lines or in a single `require` specification as a comma-separated list in angle brackets:

```
require ["fileinto", "vacation"];
if <conditions> {
  fileinto [...]
}
```

An email could match more than one rule. As long as a Sieve action has not yet triggered a final decision (`reject`) or `stop` has not explicitly precluded the further processing of rules for this email, all rules are checked and all actions triggered by them are executed.

The most important Sieve actions are:

`discard`
: prevents the email from being stored in the INBOX by default (but it may be saved somewhere else by another action).

`fileinto <directory>`
: must be loaded as an IMAP extension[113] and then moves the message to the specified IMAP folder. If it does not yet exist, the email is stored in the `INBOX` as normal. If multiple `fileinto` commands are specified multiple copies of the email are saved:

```
require "fileinto";
if <condition> {
  fileinto "INBOX/Projects";
  fileinto "INBOX/Test";
}
```

`keep`
: saves the email in the `INBOX` (as well), which is useful in combination with `fileinto` or `redirect`:

```
if <condition> {
  fileinto "INBOX/Projects";
  keep;
}
```

`redirect <email address>`

redirects the email to the email address specified here – the email is returned to the MTA. An additional `keep` ensures that the email is still saved in the INBOX:

```
if <condition> {
  redirect "paul@example.com";
  keep;
}
```

`reject <text>`

must be loaded as an IMAP extension[114] and then returns the email to the sender with a message:

```
require "reject";
if <condition> {
  reject "I do not want this message";
}
```

`vacation <text>`

must be loaded as an IMAP extension[115] and can return an out-of-office replay to the sender ("Autoresponder"). Often the condition is simply set to `true` so that all messages are responded to:

```
require "vacation";
if true {
  vacation "I will be away from the office on business until 20 June."
}
```

By default, a sender will only receive such an automatic response every seven days. The parameter `:days <number>` specifies an individual period of time:

```
require "vacation";
if <condition> {
  vacation :days 2 "I will be away from the office on business until 2
}
```

For security reasons, Sieve will only react if it also finds the recipient in the address line of the email header. You can use the `:addresses` option to add additional permitted recipient addresses in order to also trigger the automatic response for emails that are redirected to the mailbox in question.

```
require "vacation";
if <condition> {
  vacation :days 2 :addresses ["geeko@example.com", "horst@example.com
}
```

---

[109] http://tools.ietf.org/html/rfc5228

[110] http://www.iana.org/assignments/sieve-extensions/sieve-extensions.xhtml

[111] I would like to thank Managesieve author Stephan Bosch very much for correcting and improving this chapter.

[112] http://tools.ietf.org/html/rfc5228

[113] http://tools.ietf.org/html/rfc5228

[114] http://tools.ietf.org/html/rfc5429

[115] http://tools.ietf.org/html/rfc5230

Index

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z