


# Make Your Own E-Mail Server - Part 1 - FreeBSD, OpenSMTPD, Rspamd and Dovecot Included

 Stefano Marinelli

included in [Freebsd](#) [Ipv6](#) [Server](#) [Networking](#) [E-Mail](#) [Rspamd](#) [Hosting](#) [Tutorial](#) [Security](#) [Dovecot](#) [Opensmtpd](#) [Ownyourdata](#)

[08-03-2024](#) [About 4800 words](#) [23 minutes](#)

 08-03-2024  About 4800 words  23 minutes



## CONTENTS



The main power of the Internet has always been one: decentralization.

Ever since I've been able to, I've always managed my email boxes independently and provided mail hosting services to my clients. Over the years, things have become increasingly complex: on one hand, there's been a disproportionate increase in spam, and on the other, big players (like Google and Microsoft) have tried to gain a lot of ground in managing these services. Private users can obtain many free services, and business users no longer need to worry about the underlying infrastructure. However, in my opinion, the price to pay is very high: the loss of ownership of one's data.

These operators, in fact, use full access to our emails to improve their services or analyze us with the aim of selling us advertisements.

As often happens in these cases, many users appreciate this type of approach and have switched to the services of big players, causing a progressive increase in the level of influence that these companies can have on something free and decentralized like email.

In my opinion, it still makes sense to manage one's own mail servers. Standards evolve, and it is therefore appropriate to follow innovations, adapt to best practices, and secure one's services and users.

Over the last ten years, I have also used and installed various [Zimbra](#) OSE servers. Aside from a few minor issues, the setup has proven stable and reliable. Recently, the company that develops this excellent tool has decided to no longer provide packages for the Open Source version. They can be generated with [scripts provided by some \(great!\) users](#), but they cannot be a solid and stable base in the long term.

I have therefore decided to gradually return to a modular, adaptable, customizable, and updatable mail host setup without worries or headaches. Such a system can include any kind of service (including webmail, integration with caldav/carddav, etc.) and it will be possible to disable what is not strictly necessary. Safer and more secure.

This will be the first in a series of articles, and at the end of this reading, you will have a secure, modern, reliable, and modular mail server. The instructions are designed for FreeBSD and related jails but with very few modifications can be applied to any BSD as well as Linux or other similar operating systems.

## Setup Planning

---

Over the years, I've used many different SMTP servers. Originally, the good old [Sendmail](#). Subsequently, and for many years, I used [Exim](#), also because it was the default system in Debian. I then migrated almost all setups towards [Postfix](#), probably the most widespread smtp server on the net, and I never had any particular problems. In recent years, I have decided to use, when possible, the excellent [OpenSMTPD](#). Being based on OpenBSD and easily installable on other OSes, it shares the primary concepts of security and reliability of OpenBSD as well as the syntax (both in command line and in configuration files) of all other OpenBSD tools. For this type of setup, I will use `opensmtpd`.

The choice to use FreeBSD (rather than OpenBSD) stems from two main factors:

1. The possibility of dividing into [jails](#), physically separating the various services and minimizing dependencies.
2. The possibility of using ZFS, with all the advantages it brings: the ability to take snapshots, the ability to perform backups quickly and efficiently, and the ability to migrate entire jails without particular problems.

The other tools that will be used are:

- [Rspamd](#) - one of the best integrated systems for checking incoming mail. Customizable, adaptable, and modular, it also provides interesting monitoring systems

on the type of incoming and outgoing mail.

- [Redis](#) - which will be used by rspamd to manage its "memory".
- [Dovecot](#) and Sieve - for delivering mail to local mailboxes as well as allowing remote email consultation (via imap) and enabling the setting of rules thanks to [Pigeonhole Sieve](#).

The users of the mail server will not be "physical" users of the server itself, but virtual users, totally unrelated to system users.

This article will be long and detailed. The process may seem complex at first glance, but in reality, it's simpler than one might think. In the classic Unix philosophy, each component performs a task and does it well. Configuring the individual components separately will ensure greater scalability, reliability, and resilience to updates as well as simpler debugging compared to all-in-one solutions (like many of those found in convenient, ready-to-use docker-compose.yaml files).

## Installation and Configuration

---

I won't describe the FreeBSD installation and configuration procedure. In my case, I activated a VPS on Hetzner - in this situation, the smallest of the available ARM servers. Yes, it's possible to efficiently manage a mail server for several users on a machine costing less than 4 euros a month. ZFS can be useful, but it's not necessary for this type of setup.

The first step is therefore to choose a location to install everything. The main condition is that the VPS/server/host must have at least one static public IP and, in 2024, I now consider it necessary to have (at least) one IPv6 address. They must also be able to send/receive mail on port 25, which is not guaranteed and not all VPS providers allow. Hetzner, for example, blocks port 25 for new users.

Once you have identified and acquired the server on which to install everything, it's appropriate to check that the IPs (both IPv4 and IPv6) to be used are not on any of the many blocklists/blacklists. My experience indicates that at least 60% of the IPs assigned to me are on one or more of these lists, so the first step is to request delisting, to ensure proper operation when the server is ready.

## To Encrypt or Not to Encrypt?

A subsequent assessment is whether to use encryption for the data. Connections to and from the mail server will always be encrypted, but it's also possible to encrypt the data at rest. In the case of FreeBSD, it's possible to enable encryption for the entire installation (the entire virtual disk will be encrypted, and ZFS (or UFS) will be created on top of it) or, in the case of using ZFS, it will be possible to encrypt only the root dataset of the jails, the one that will be used by BastilleBSD. In the first case, it will be necessary to enter the password at every boot of FreeBSD, therefore connect via console at every restart and enter the password. In the second case, FreeBSD will be able to boot but it will be necessary to

connect and enter the keys before mounting the BastilleBSD ZFS dataset. None of these solutions will prevent unauthorized access to emails in case of machine compromise. The main advantage will be "only" not to save the emails in clear on disks which, in the case of a VM, will still be shared and managed by third parties. In the case of physical servers, however, things will undoubtedly be different.

Encrypting the entire disk must be done directly during the FreeBSD installation phase, very simply. As for the option to encrypt only the BastilleBSD dataset, you can proceed later, with a command similar to:

```
zfs create -o encryption=on -o keylocation=prompt -o keyformat=passphrase  
zroot/bastille
```

at every reboot, it will be necessary to enter the password and mount the datasets:

#### ▼ Code

```
1 zfs load-key -r zroot/bastille  
2 zfs mount -a
```

## Configuring FreeBSD and BastilleBSD

After the installation of the OS, the first step is to configure IPv6 on the VPS. In the case of Hetzner, unfortunately, they only provide a /64, so it will be necessary to segment the assigned network. In this example, it will be divided into /72 subnetworks - to find valid subclasses, it will be possible [to use a calculator](#).

The `/etc/rc.conf` file should have entries similar to:

#### ▼ Code

```
1 ifconfig_vtnet0="DHCP"  
2 ifconfig_vtnet0_ipv6="inet6 2a01:4f8:cafe:cafe::1 prefixlen 72"  
3 ipv6_defaultrouter="fe80::1%vtnet0"
```

In short, keep the base address assigned by Hetzner, but change the prefix length to 72 - thus giving the possibility of having other networks available.

It is now necessary to enable forwarding for IPv4 and IPv6. Add these lines to the `/etc/sysctl.conf` file:

#### ▼ Code

```
1 net.inet.ip.forwarding=1  
2 net.inet6.ip6.forwarding=1
```

After reboot, test it:

```
ping6 google.com
```

If everything has been configured correctly, the ping will be executed and google.com will reply.

It will also be necessary to set up reverse DNS, otherwise many mail servers will reject our emails. This must be done both for the IPv4 address and the designated IPv6 address (within the class - an operation that can also be done after creating the jails, but before starting to send email messages). Also, create the correct A and AAAA records on the DNS for the "mail.example.com" domain.

The FreeBSD installer does not automatically update the operating system, so it is appropriate to do so now:

```
freebsd-update fetch install
```

It seems, moreover, that there is still [an old bug still present](#) and manifesting when, on a FreeBSD server installed in a VM based on KVM (thus also those of Hetzner), routing is performed (as in our case) between VNET jails and host.

Adding this configuration to `/boot/loader.conf` will solve the problem:

#### Code

```
1 hw.vtnet.X.tso_disable="1"
2 hw.vtnet.tso_disable="1"
3 hw.vtnet.lro_disable="1"
4 hw.vtnet.X.lro_disable="1"
5 hw.vtnet.csum_disable="1"
6 hw.vtnet.X.csum_disable="1"
```

It's time to install and configure [BastilleBSD](#). This handy tool will make managing jails much simpler and more straightforward, and [its configuration is well described on the project page](#).

## Bridge and Firewall

For some time now, I've been favoring the use of VNET jails for these types of setups. I believe that having a complete network stack within the jails gives more freedom in configuration, firewall management (applicable also within individual jails), and technical possibilities (such as creating VPNs within the jails themselves, ensuring greater portability).

I suggest creating a bridge on the FreeBSD server and placing all the jails within this bridge. It will be enough to modify the `/etc/rc.conf` file and change/add:

#### Code

```
1 cloned_interfaces="lo1 bridge0"
2 ifconfig_bridge0="inet 192.168.123.1 netmask 255.255.255.0"
3 ifconfig_bridge0_ipv6="inet6 2a01:4f8:cafe:cafe:100::1 prefixlen 72"
4 ipv6_gateway_enable="YES"
5 ipv6_activate_all_interfaces="YES"
6 gateway_enable="YES"
```

Now let's configure the firewall. In this example, I will use the IPv4 and IPv6 addresses that I will later assign to the jails. Here's an example of a working `pf.conf` :

#### Code

```
1 ext_if="vtnet0"
2
3 set block-policy return
4 scrub in on $ext_if all fragment reassemble
5 set skip on lo
6 set skip on bridge0
7
8 table <jails> persist
9 # IPv4 private address ranges
10 table <private> const { 10/8, 172.16/12, 192.168/16 }
11 nat on $ext_if from 192.168.123.0/24 to ! <private> -> ($ext_if:0)
12 nat on $ext_if from <jails> to any -> ($ext_if:0)
13 rdr-anchor "rdr/*"
14 #rdr via ipv4 to mail
15 rdr pass on $ext_if proto tcp from any to ($ext_if) port { 25, 465, 587, 14
16 #rdr via ipv4 to nginx-proxy
17 rdr pass on $ext_if proto tcp from any to ($ext_if) port { 80, 443 } -> 192
18
19 block in all
20 #PASS ICMP
21 pass in quick proto icmp from any to any
22 # Pass ICMP on ipv6
23 pass quick proto ipv6-icmp
24 pass out quick keep state
25 antispoof for $ext_if inet
26 pass in inet proto tcp from any to any port ssh flags S/SA keep state
27 #Pass ipv6 to mail jail
28 pass in quick on $ext_if inet6 proto tcp from any to 2a01:4f8:cafe:cafe:100
29 #Pass ipv6 to nginx-proxy jail
30 pass in quick on $ext_if inet6 proto tcp from any to 2a01:4f8:cafe:cafe:100
```

At this point, it is advisable to reboot the machine, to be sure that everything comes back up in the right way.

## Creating the "nginx-proxy" and "redis" Jails

Let's start with the first jail. It will be an nginx reverse proxy. At the moment it will not be used as such but will be useful as a machine for generating certificates and, in the future, to act as a reverse proxy for various internal web services.

```
bastille create -B nginx-proxy 14.0-RELEASE 192.168.123.2/24 bridge0
```

Once the jail is created, it will be necessary to modify some configurations. In the jail itself (bastille console nginx-proxy), modify the `/etc/rc.conf` file and add the IPv4 gateway and

IPv6 configurations, i.e., giving the address specified earlier in the `pf.conf` of the FreeBSD server and as the gateway the IP address of the related bridge:

▼ **Code**

```
1 defaultrouter="192.168.123.1"
2 ifconfig_vnet0_ipv6="inet6 2a01:4f8:cafe:cafe:100::80 prefixlen 72"
3 ipv6_defaultrouter="2a01:4f8:cafe:cafe:100::1"
```

Restart nginx-proxy with `bastille restart nginx-proxy`, and you will be able to re-enter the jail. At that point, `ping google.com` and `ping6 google.com` should work. The jail will then be able to operate, responding on 80 and 443 both in IPv4 (thanks to the NAT configured previously on the FreeBSD server) and in IPv6, being directly connected in routing.

Now install the necessary software:

```
pkg install -y nginx py39-certbot py39-certbot-nginx
```

To automatically renew the certificates, add this line to `/etc/periodic.conf`:

▼ **Code**

```
1 weekly_certbot_enable="YES"
```

It's now possible to generate the certificate, which we will also use for all other exposed services:

```
certbot certonly -d mail.example.com
```

Once the certificates are generated, we will need to make a small change as `opensmtpd` (which we will install later in another jail) is (rightly) very restrictive on permissions:

```
chmod -R 400 /usr/local/etc/letsencrypt/archive/mail.example.com/
```

Now let's create a jail for redis. I usually put it in a dedicated jail also in terms of clustering/multi-server setup:

```
bastille create -B redis 14.0-RELEASE 192.168.123.4/24 bridge0
```

Also for the redis jail, it's appropriate to set up the gateway, so in the `/etc/rc.conf` of the jail:

▼ **Code**

```
1 defaultrouter="192.168.123.1"
```

I don't configure IPv6 as it's not necessary (this jail will only be reachable from the LAN) but it's possible to do so. Once the jail is restarted (`bastille restart redis`), it will be able to reach the outside:

```
pkg install -y redis
```

Not wanting to disable the protected mode of redis and not wanting to open it without authentication, it will be appropriate to modify the `/usr/local/etc/redis.conf`

configuration file and add the password and some options to optimize its use with rspamd:

✓ **Code**

```
1 requirepass cafecafe
2 maxmemory 512mb
3 maxmemory-policy volatile-lru
```

Of course, set a unique password. Also, change the line concerning the bind:

✓ **Code**

```
1 bind *
```

Now execute:

✓ **Code**

```
1 service redis enable
2 service redis start
```

Redis should be active and ready to receive connections.

## The "mail" Jail

It's time to create the jail with the main services:

```
bastille create -B mail 14.0-RELEASE 192.168.123.3/24 bridge0
```

and, also in this case, it will be necessary to enter the jail and modify some configurations in `/etc/rc.conf` :

✓ **Code**

```
1 defaultrouter="192.168.123.1"
2 ifconfig_vnet0_ipv6="inet6 2a01:4f8:cafe:cafe:100::25 prefixlen 72"
3 ipv6_defaultrouter="2a01:4f8:cafe:cafe:100::1"
```

Once the jail is restarted, it will be possible to install the necessary packages:

```
pkg install -y rspamd opensmtpd opensmtpd-extras opensmtpd-extras-table-passwd
opensmtpd-filter-senderscore opensmtpd-filter-rspamd dovecot dovecot-
pigeonhole
```

It is now appropriate to give this jail an FQDN, as it will be the name with which it presents itself when connecting to the outside. Just modify `/etc/rc.conf` :

✓ **Code**

```
1 hostname="mail.example.com"
```

The hostname should be equivalent to the reverse DNS set earlier, so the machine will present itself with a name that, doing a reverse lookup, will correspond to the IP of origin.



This jail does not contain the certificates that are present in the nginx-proxy jail. There are various methods to share them, such as using rsync, putting them on an NFS share, etc., but the simplest in this case will be to do a bind mount between the two jails, an operation that BastilleBSD can handle automatically.

Create the correct directory in the "mail" jail:

```
mkdir /usr/local/etc/letsencrypt
```

Exit the jail, shut it down, and modify the fstab file of the jail (e.g., /usr/local/bastille/jails/mail/fstab ) by adding a line like this:

▼ **Code**

```
1 /usr/local/bastille/jails/nginx-proxy/root/usr/local/etc/letsencrypt /usr/l
```

Start the jail, and at that point, the directory created earlier should display the directories and certificates from the nginx-proxy jail.

Due to spammers who, year after year, become increasingly aggressive, many mail servers require perfect configuration of the main sender authentication methodologies. Today, the main (and now necessary, under penalty of message non-delivery) are [SPF](#), [DKIM](#), and [DMARC](#).

The first step is to generate a DKIM key for the domain(s) that will send mail from the server we are configuring. This operation can be performed in a few steps, by placing the keys in the /usr/local/etc/mail/dkim directory:

▼ **Code**

```
1 mkdir /usr/local/etc/mail/dkim
2 cd /usr/local/etc/mail/dkim
3 openssl genrsa -out example.com.key 2048
4 openssl rsa -in example.com.key -pubout -out public.key
5 chmod 0400 example.com.key
6 chown rspamd *
```

Since rspamd will take care of signing outgoing messages, it is appropriate that only the related user be able to read the file with the private key.

Once the pair of keys has been created, it will be necessary to configure DNS to provide the public key to mail servers that will request it. The record should look like this:

▼ **Code**

```
1 mail._domainkey TXT "v=DKIM1;k=rsa;p=your-public-key-goes-here"
```

Once the record has been entered and propagated, it will be possible to test its correctness through one of the many websites that offer this service, remembering that the selector we have configured is "mail" (the first part of the record). Obviously, this is a free text, which can be modified at will (but will then need to be specified in the rspamd configuration, later on).

Being in the DNS configuration phase, it will also be appropriate to create SPF and DMARC records:

SPF record:

▼ Code

```
1 example.com. IN TXT "v=spf1 a ip4:your.ip.address ip6:2a01:4f8:cafe:cafe:10
```

DMARC:

▼ Code

```
1 _dmarc.example.com. IN TXT "v=DMARC1;p=none;pct=100;rua=mailto:postmaster@e
```

These records, obviously modified based on the chosen domain, are configured conservatively but suitable for a test. Once the entire setup is in place, it will be possible to handle things in a more restrictive manner.

The mail server users will be “virtual” users (i.e., not system users of the server), so all their mail will be “handled” by a unique user, which needs to be created:

▼ Code

```
1 pw user add vmail -u 2000 -d /var/vmail -s /usr/sbin/nologin
2 mkdir /var/vmail
3 chown vmail /var/vmail
```

Let’s now create some files and directories, useful for the subsequent configuration:

▼ Code

```
1 touch /usr/local/etc/mail/passwd
2 touch /usr/local/etc/mail/virtuals
3 mkdir /usr/local/etc/rspamd/local.d
```

Remove the current `/usr/local/etc/mail/smtpd.conf` and replace it with content like this:

▼ Code

```
1 table passwd passwd:/usr/local/etc/mail/passwd
2 table virtuals file:/usr/local/etc/mail/virtuals
3
4 pki mail.example.com cert "/usr/local/etc/letsencrypt/archive/mail.example.
5 pki mail.example.com key "/usr/local/etc/letsencrypt/archive/mail.example.c
6
7 filter check_dyndns phase connect match rdns regex { '.*\.dyn\..*', '.*\.ds
8     disconnect "550 no residential connections - Thou shalt not pass"
9
10 filter check_rdns phase connect match !rdns \
11     disconnect "550 no rdns - Thou shalt not pass"
12
13 filter check_fcrdns phase connect match !fcrdns \
```

```

14     disconnect "550 no FCrDNS - Thou shalt not pass"
15
16 filter senderscore \
17     proc-exec "/usr/local/libexec/opensmtpd/opensmtpd-filter-senderscore -b
18
19 filter rspamd proc-exec "/usr/local/libexec/opensmtpd/opensmtpd-filter-rspa
20
21 listen on 0.0.0.0 tls pki mail.example.com \
22     filter { check_dyndns, check_rdns, check_fcrdns, senderscore, rspamd }
23
24 listen on ::0 tls pki mail.example.com \
25     filter { check_dyndns, check_rdns, check_fcrdns, senderscore, rspamd }
26
27 listen on 0.0.0.0 port submission tls-require pki mail.example.com auth <p
28
29 listen on ::0 port submission tls-require pki mail.example.com auth <passw
30
31 listen on 0.0.0.0 port 465 smtps pki mail.example.com auth <passwd> filter
32
33 listen on ::0 port 465 smtps pki mail.example.com auth <passwd> filter rsp
34
35 action "local_mail" lmtp "/var/run/dovecot/lmtp" rcpt-to virtual <virtuals>
36 action "outbound" relay helo mail.example.com
37
38 match from any for domain example.com action "local_mail"
39 match for local action "local_mail"
40
41 match from any auth for any action "outbound"
42 match for any action "outbound"

```

The beauty of OpenSMTPD lies in its simplicity. Essentially, that's all there is to it, but here's an explanation of what this file does:

- `table passwd` and `table virtuals` define tables for user credentials and virtual domains/users, crucial for authentication and email forwarding/aliasing.
- `pki` lines specify the SSL/TLS certificate and key for encrypted connections.
- `filter` directives apply various checks and filters to incoming connections, including dynamic DNS and reverse DNS validations, sender reputation scoring, and Rspamd filtering.
- `listen` lines configure the server to listen on all IPv4 and IPv6 addresses for incoming SMTP and submission (port 587) connections, applying TLS and authentication as specified.
- `action` lines define actions for email delivery, using LMTP for local delivery based on virtual mappings and setting up relay actions for outbound emails.
- `match` rules determine how emails are processed, either delivered locally or relayed externally based on source, destination, and authentication.

To test the newly inserted configuration, run `smtpd -n`.

The next step is to configure Rspamd. Rather than modifying the base configuration files, we'll proceed by creating "overrides", placing specific configurations in the directory created earlier ( `/usr/local/etc/rspamd/local.d` ):

For Redis configuration in `/usr/local/etc/rspamd/local.d/redis.conf` :

▼ **Code**

```
1 servers = "192.168.123.4";
2 password = "cafecafe";
```

For SPF settings in `/usr/local/etc/rspamd/local.d/spf.conf` :

▼ **Code**

```
1 spf_cache_size = 1k;
2 spf_cache_expire = 1d;
3 max_dns_nesting = 10;
4 max_dns_requests = 30;
5 min_cache_ttl = 5m;
```

Now, configure DKIM signing, which rspamd will handle for every sent email. The selector must match the one entered in the DNS record previously:

In `/usr/local/etc/rspamd/local.d/dkim_signing.conf` :

▼ **Code**

```
1 domain {
2   example.com {
3     path = "/usr/local/etc/mail/dkim/example.com.key";
4     selector = "mail";
5   }
6 }
```

To enable support for some "known" phisher lists, create `/usr/local/etc/rspamd/local.d/phishing.conf` :

▼ **Code**

```
1 # Configuration options can be added here`
2 openphish_enabled = true;
3 phishtank_enabled = true;
```

For SURBL (Spam URI Real-time Blocklists) configuration in `/usr/local/etc/rspamd/local.d/surbl.conf` :

▼ **Code**

```
1 # follow redirects when checking URLs in emails for spaminess
2 redirector_hosts_map = "/usr/local/etc/rspamd/redirectors.inc";
```

For URL reputation checks in `/usr/local/etc/rspamd/local.d/url_reputation.conf` :

▼ **Code**

```
1 # check URLs within messages for spaminess
2 enabled = true;
```

And for caching some URL tags in Redis, in  
`/usr/local/etc/rspamd/local.d/url_tags.conf` :

#### ▼ Code

```
1 # cache some URL tags in redis
2 enabled = true;
```

Rspamd should be ready for initial testing.

Now it's time to configure Dovecot, which will handle the final phase of delivery and provide mail access, as well as use Sieve for filtering and rule application.

The first step is to start with a default configuration:

```
cd /usr/local/etc/dovecot/ && cp -R example-config/* .
```

Next, modify some files. Start with `conf.d/15-mailboxes.conf` - add `auto = create` to the main folders, like this:

#### ▼ Code

```
1 namespace inbox {
2   # These mailboxes are widely used and could perhaps be created automatica
3   separator = .
4
5   mailbox Drafts {
6     special_use = \Drafts
7     auto = create
8   }
9   mailbox Junk {
10    special_use = \Junk
11    auto = create
12  }
13  mailbox Trash {
14    special_use = \Trash
15    auto = create
16  }
17
18  # For \Sent mailboxes there are two widely used names. We'll mark both of
19  # them as \Sent. User typically deletes one of them if duplicates are cre
20  mailbox Sent {
21    special_use = \Sent
22    auto = create
23  }
24  mailbox "Sent Messages" {
25    special_use = \Sent
26  }
27 }
```

Modify `/usr/local/etc/dovecot/dovecot.conf` to include:

▼ **Code**

```
1 protocols = imap lmtp sieve
```

In `conf.d/10-auth.conf`, switch to system authentication with passwdfile:

▼ **Code**

```
1 #!include auth-system.conf.ext
2 !include auth-passwdfile.conf.ext
```

Then, set up the `passdb` and `userdb` configuration in `conf.d/auth-passwdfile.conf.ext`. Replace the entire file with these contents:

▼ **Code**

```
1 passdb {
2     driver = passwd-file
3     args = scheme=CRYPT /usr/local/etc/mail/passwd
4 }
5
6 userdb {
7     default_fields = uid=vmail gid=vmail home=/var/vmail/%d/%n
8     args = /usr/local/etc/mail/passwd
9     driver = passwd-file
10 }
```

Edit `conf.d/10-ssl.conf` to update `ssl_key` and `ssl_cert`:

▼ **Code**

```
1 ssl_cert = </usr/local/etc/letsencrypt/archive/example.com/fullchain1.pem
2 ssl_key = </usr/local/etc/letsencrypt/archive/example.com/privkey1.pem
```

In `conf.d/20-imap.conf`, include:

▼ **Code**

```
1 (...)
2 mail_plugins = $mail_plugins imap_sieve zlib
3 mail_max_userip_connections = 100
```

And in `conf.d/20-lmtp.conf`:

▼ **Code**

```
1 protocol lmtp {
2     mail_fsync = optimized
3     mail_plugins = $mail_plugins sieve
4 }
```

Set the mail location in `conf.d/10-mail.conf`:

▼ **Code**

```
1 mail_location = maildir:/var/vmail/%d/%n
2
3 maildir_stat_dirs=yes
```

Now, let's configure Sieve in `conf.d/90-plugin.conf` :

#### ▼ Code

```
1 plugin {
2   #setting_name = value
3   sieve_plugins = sieve_imapsieve sieve_extprograms
4   sieve_global_extensions = +vnd.dovecot.pipe +vnd.dovecot.environment
5   imapsieve_mailbox1_name = Junk
6   imapsieve_mailbox1_causes = COPY APPEND
7   imapsieve_mailbox1_before = file:/usr/local/lib/dovecot/sieve/report-spam
8
9   imapsieve_mailbox2_name = *
10  imapsieve_mailbox2_from = Junk
11  imapsieve_mailbox2_causes = COPY
12  imapsieve_mailbox2_before = file:/usr/local/lib/dovecot/sieve/report-ham.
13
14  imapsieve_mailbox3_name = Inbox
15  imapsieve_mailbox3_causes = APPEND
16  imapsieve_mailbox3_before = file:/usr/local/lib/dovecot/sieve/report-ham.
17  sieve_pipe_bin_dir = /usr/local/lib/dovecot/sieve
18
19  #Move the spam to Junk folder
20  sieve_after = /usr/local/lib/dovecot/sieve/spam-to-folder.sieve
21 }
```

And in `conf.d/20-managesieve.conf.ext` :

#### ▼ Code

```
1 service managesieve-login {
2   inet_listener sieve {
3     port = 4190
4   }
5
6   service_count = 1
7   vsz_limit = 64M
8 }
9
10 service managesieve {
11   process_limit = 1024
12 }
13
14 protocol sieve {
15   mail_max_userip_connections = 10
16
17   # Configuration for Sieve script management via ManageSieve protocol
```

```

18 }
19
20 plugin {
21     sieve_pipe_bin_dir = /usr/local/lib/dovecot/sieve
22     sieve = file:/var/vmail/%d/%n/sieve;active=/var/vmail/%d/%n/.dovecot.siev
23     sieve_user_log = file:/var/vmail/%d/%n/sieve/sieve_error.log
24 }

```

This setup outlines the Dovecot configuration necessary for handling mail delivery, access, and Sieve-based filtering. With these settings, Dovecot is prepared to manage both incoming and outgoing mails securely and efficiently, including support for managing Sieve scripts via the ManageSieve protocol.

Sieve scripts train Rspamd on **spam** and **ham**. Moving email into and out of the junk folder triggers an event to train Rspamd.

These files are located at `/usr/local/lib/dovecot/sieve` .

Create the `report-ham.sieve` file:

#### ▼ Code

```

1 require ["vnd.dovecot.pipe", "copy", "imapsieve", "environment", "variables
2
3 if environment :matches "imap.mailbox" "*" {
4     set "mailbox" "${1}";
5 }
6
7 if string "${mailbox}" "Trash" {
8     stop;
9 }
10
11 if environment :matches "imap.user" "*" {
12     set "username" "${1}";
13 }
14
15 pipe :copy "sa-learn-ham.sh" [ "${username}" ];

```

Create the `report-spam.sieve` file:

#### ▼ Code

```

1 require ["vnd.dovecot.pipe", "copy", "imapsieve", "environment", "variables
2
3 if environment :matches "imap.user" "*" {
4     set "username" "${1}";
5 }
6
7 pipe :copy "sa-learn-spam.sh" [ "${username}" ];

```

Create the `spam-to-folder.sieve` file:



▼ **Code**

```
1 require ["fileinto","mailbox"];
2
3 if header :contains "X-Spam" "Yes" {
4   fileinto :create "Junk";
5   stop;
6 }
```

Compile the files:

▼ **Code**

```
1 sievec report-ham.sieve
2 sievec report-spam.sieve
3 sievec spam-to-folder.sieve
```

Create the following two shell scripts in `/usr/local/lib/dovecot/sieve` .

Add the following to `sa-learn-ham.sh` :

▼ **Bash**

```
1 #!/bin/sh
2 exec /usr/local/bin/rspamc -d "${1}" learn_ham
```

Add the following to `sa-learn-spam.sh` :

▼ **Bash**

```
1 #!/bin/sh
2 exec /usr/local/bin/rspamc -d "${1}" learn_spam
```

Make the files executable:

▼ **Code**

```
1 chmod 755 sa-learn-ham.sh
2 chmod 755 sa-learn-spam.sh
```

Everything has been configured correctly. However, a fundamental part is missing: the users.

To begin, encrypt the password by executing:

```
smtptcl encrypt
```

Input the plain password and hit ENTER. Copy the displayed encrypted password for the next step. Now, insert the entry into `/usr/local/etc/mail/passwd` :

▼ **Code**

```
1 foo@example.com:your encrypted password goes here::::::
```

Remember to replace the placeholder with your encrypted password and ensure there are *six colons* at the end.

Finally, add the virtual user and associate it with the `vmail` system user in `/usr/local/etc/mail/virtuall` :

▼ **Code**

```
1 foo@example.com vmail
```

Note: Additional aliases can be included, such as:

▼ **Code**

```
1 postmaster@example.com foo@example.com
```

Let's enable and start the services:

▼ **Code**

```
1 service rspamd enable
2 service rspamd start
3 service smtpd enable
4 service smtpd start
5 service dovecot enable
6 service dovecot start
```

At this point, take a look at the logs (especially `/var/log/maillog` ) to get an idea of what's happening. If everything is running correctly, you can now proceed with some tests. Any mail client (such as Thunderbird, etc.) should be able to connect via IMAP and send emails using SMTP, thanks to authentication. Remember that the username corresponds to the one entered in the `/usr/local/etc/mail/passwd` file and typically matches the full email address.

Once everything is in order, you can set the MX record of the domain "example.com" to "mail.example.com".

One of the key tests to perform is to send an email to a Gmail inbox. If the email is correctly delivered and not marked as Spam, there's a good chance that the setup is correct. There are also several online tools available that can provide useful insights into any configuration errors.

You have just installed and operationalized a complete mail server.

It is now possible to [proceed with part 2, which involves the installation of Nextcloud and Snappymail](#), in order to have a comprehensive groupware system, while keeping one's data on their own servers.

## Links:

This article was inspired by several other articles available online that provided me with a lot of useful information. Here is a non-exhaustive list:

- [A Comprehensive Guide to Setting Up Your Mail Server](#)
- [Self-Hosted Email Services on OpenBSD](#)

- [Building a FreeBSD Mail Server: Part 1](#)
- [An OpenBSD E-mail Server Using OpenSMTPD, Dovecot, Rspamd, and RainLoop](#)
- [Setting up a mail server with OpenSMTPD, Dovecot and Rspamd](#)



## Related Content

- [Make Your Own VPN - FreeBSD, Wireguard, Ipv6 and Ad-Blocking Included](#)
- [Make Your Own VPN - Wireguard, Ipv6 and Ad-Blocking Included](#)
- [How We Are Migrating \(Many Of\) Our Servers From Linux to FreeBSD - Part 1 - System and Jails Setup](#)
- [How to Create a FreeBSD Jail Hosting XRDP and XFCE for Remote Desktop Access](#)
- [Migrating From VM to Hierarchical Jails in FreeBSD](#)

Updated on 08-03-2024

---

🔖 [Freebsd, Ipv6, Server, Networking, E-Mail, Rspamd, Hosting, Tutorial, Security, Dovecot, Opensmtpd, Ownyourdata](#)