目录

A-mail-server's-101

What is a mail server and how does it perform its duty?
Here's an introduction to the field that covers everything you need to know to get started with `docker-mailserver`.

Anatomy of a mail server

A mail server is only a part of a client-server relationship aimed at exchanging information in the form of emails. Exchanging emails requires using specific means (programs and protocols).

`docker-mailserver` provides you with the server portion, whereas the client can be anything from a terminal via text-based software (eg. Mutt) to a fully-fledged desktop application (eg. Mozilla Thunderbird, Microsoft Outlook…), to a web interface, etc.

Unlike the client-side where usually a single program is used to perform retrieval and viewing of emails, the server-side is composed of many specialized components. The mail server is capable of accepting, forwarding, delivering, storing and overall exchanging messages, but each one of those tasks is actually handled by a specific piece of software. All of these "agents" must be integrated with one another for the exchange to take place.

`docker-mailserver` has made informed choices about those components and their (default) configuration. It offers a comprehensive platform to run a fully featured mail server in no time!

Components

The following components are required to create a complete delivery chain:

- MUA: a Mail User Agent is basically any client/program capable of sending emails to arbitrary mail servers; while also capable of fetching emails from mail servers for presenting them to the end users.
- MTA: a Mail Transfer Agent is the so-called "mail server" as seen from the MUA's perspective. It's a piece of software dedicated to accepting submitted emails, then forwarding them-where exactly will depend on an email's final destination. If the receiving MTA is responsible for the hostname the email is sent to, then an MTA is to forward that email to an MDA (see below). Otherwise, it is to transfer (ie. forward, relay) to another MTA, "closer" to the email's final destination.
- MDA: a Mail Delivery Agent is responsible for accepting emails from an MTA and dropping them into their recipients' mailboxes, whichever the form.

Here's a schematic view of mail delivery:

```
Sending an email:   MUA ----> MTA ----> (MTA relays) ----> MDA
Fetching an email:  MUA <---------------------------------- MDA
```

There may be other moving parts or sub-divisions (for instance, at several points along the chain, specialized programs may be analyzing, filtering, bouncing, editing… the exchanged emails).

In a nutshell, `docker-mailserver` provides you with the following components:

- A MTA: Postfix
- A MDA: Dovecot
- A bunch of additional programs to improve security and emails processing

Here's where `docker-mailserver`'s toochain fits within the delivery chain:

```
                          docker-mailserver is here:
                                  ┌──────────┐
Sending an email:   MUA ---> MTA ---> (MTA relays) ---> ┤ MTA ┐ │
Fetching an email:  MUA <------------------------------ ┤ MDA ┘ │
                                  └──────────┘
```

> Let's say Alice owns a Gmail account, `alice@gmail.com`; and Bob owns an account on a `docker-mailserver`'s instance, `bob@dms.io`.
>
> Make sure not to conflate these two very different scenarios: A) Alice sends an email to `bob@dms.io` => the email is first submitted to MTA `smtp.gmail.com`, then relayed to MTA `smtp.dms.io` where it is then delivered into Bob's mailbox. B) Bob sends an email to `alice@gmail.com` => the email is first submitted to MTA `smtp.dms.io`, then relayed to MTA `smtp.gmail.com` and eventually delivered into Alice's mailbox.
>
> In scenario *A* the email leaves Gmail's premises, that email's *initial* submission is *not* handled by your `docker-mailserver` instance(MTA); it

> merely receives the email after it has been relayed by Gmail's MTA. In scenario *B*, the `docker-mailserver` instance(MTA) handles the submission, prior to relaying.
>
> The main takeaway is that when a third-party sends an email to a `docker-mailserver` instance(MTA) (or any MTA for that matter), it does *not* establish a direct connection with that MTA. Email submission first goes through the sender's MTA, then some relaying between at least two MTAs is required to deliver the email. That will prove very important when it comes to security management.

One important thing to note is that MTA and MDA programs may actually handle *multiple* tasks (which is the case with `docker-mailserver`'s Postfix and Dovecot).

For instance, Postfix is both an SMTP server (accepting emails) and a relaying MTA (transferring, ie. sending emails to other MTA/MDA); Dovecot is both an MDA (delivering emails in mailboxes) and an IMAP server (allowing MUAs to fetch emails from the *mail server*). On top of that, Postfix may rely on Dovecot's authentication capabilities.

The exact relationship between all the components and their respective (sometimes shared) responsibilities is beyond the scope of this document. Please explore this wiki & the web to get more insights about `docker-mailserver`'s toolchain.

About security & ports

In the previous section, different components were outlined. Each one of those is responsible for a specific task, it has a specific purpose.

Three main purposes exist when it comes to exchanging emails:

- *Submission*: for a MUA (client), the act of sending actual email data over the network, toward an MTA (server).
- *Transfer* (aka. *Relay*): for an MTA, the act of sending actual email data over the network, toward another MTA (server) closer to the final destination (where an MTA will forward data to an MDA).
- *Retrieval*: for a MUA (client), the act of fetching actual email data over the network, from an MDA.

Postfix handles Submission (and might handle Relay), whereas Dovecot handles Retrieval. They both need to be accessible by MUAs in order to act as servers, therefore they expose public endpoints on specific TCP ports (see. *Understanding the ports* for more details). Those endpoints *may* be secured, using an encryption scheme and TLS certificates.

When it comes to the specifics of email exchange, we have to look at protocols and ports enabled to support all the identified purposes. There are several valid options and they've been evolving overtime.

Here's **`docker-mailserver`**'s *default* configuration:

Purpose	Protocol	TCP port / encryption
Transfer/Relay	SMTP	25 (unencrypted)
Submission	ESMTP	587 (encrypted using STARTTLS)
Retrieval	IMAP4	143 (encrypted using STARTTLS) + 993 (TLS)
Retrieval	POP3	*Not activated*

If you're new to email infrastructure, both that table and the schema may be confusing.
Read on to expand your understanding and learn about `docker-mailserver`'s configuration, including how you can customize it.

Submission - SMTP

For a MUA to send an email to an MTA, it needs to establish a connection with that server, then push data packets over a network that both the MUA (client) and the MTA (server) are connected to. The server implements the SMTP protocol, which makes it capable of handling *Submission*.

In the case of `docker-mailserver`, the MTA (SMTP server) is Postfix. The MUA (client) may vary, yet its Submission request is performed as TCP packets sent over the *public* internet. This exchange of information may be secured in order to counter eavesdropping.

Two kinds of Submission

Let's say I own an account on a `docker-mailserver` instance, `me@dms.io`. There are two very different use-cases for Submission:

1. I want to send an email to someone
2. Someone wants to send you an email

In the first scenario, I will be submitting my email directly to my `docker-mailserver` instance/MTA (Postfix), which will then relay the email to its recipient's MTA for final delivery. In this case, Submission is first handled by establishing a direct connection to my own MTA-so at least for this portion of the delivery chain, I'll be able to ensure security/confidentiality. Not so much for what comes next, ie. relaying between MTAs and final delivery.

In the second scenario, a third-party email account owner will be first submitting an email to some third-party MTA. I have no control over this initial portion of the delivery chain, nor do I have control over the relaying that comes next. My MTA will merely accept a relayed email coming "out of the blue".

My MTA will thus have to support two kinds of Submission:

- Outward Submission (self-owned email is submitted directly to the MTA, then is relayed "outside")
- Inward Submission (third-party email has been submitted & relayed, then is accepted "inside" by the MTA)

Outward Submission

The best practice as of 2020 when it comes to securing Outward Submission is to use *Implicit TLS connection via ESMTP on port 465* (see RFC 8314). Let's break it down.

- Implicit TLS means the server *enforces* the client into using an encrypted TCP connection, using TLS. With this kind of connection, the MUA *has* to establish a TLS-encrypted connection from the get go (TLS is implied, hence the name "Implicit"). Any client attempting to either submit email in cleartext (unencrypted, not secure), or requesting a cleartext connection to be upgraded to a TLS-encrypted one using `STARTTLS`, is to be denied. Implicit TLS is sometimes called Enforced TLS for that reason.
- ESMTP is SMTP + extensions. It's the version of the SMTP protocol that most mail servers speak nowadays. For the purpose of this documentation, ESMTP and SMTP are synonymous.
- Port 465 is the reserved TCP port for Implicit TLS Submission (since 2018). There is actually a boisterous history to that ports usage, but let's keep it simple.

> Note: This Submission setup is sometimes refered to as SMTPS. Long story short: this is incorrect and should be avoided.

Although a very satisfactory setup, Implicit TLS on port 465 is somewhat "cutting edge". There exists another well established mail Submission setup that must be supported as well, SMTP+STARTTLS on port 587. It uses Explicit TLS: the client starts with a cleartext connection, then the server informs a TLS-encrypted "upgraded" connection may be established, and the client *may* eventually decide to establish it prior to the Submission. Basically it's an opportunistic, opt-in TLS upgrade of the connection between the client and the server, at the client's discretion, using a mechanism known as STARTTLS that both ends need to implement.

In many implementations, the mail server doesn't enforce TLS encryption, for backwards compatibility. Clients are thus free to deny the TLS-upgrade proposal (or misled by a hacker about STARTTLS not being available), and the server accepts unencrypted (cleartext) mail exchange, which poses a confidentiality threat and, to some extent, spam issues. RFC 8314 (section 3.3) recommends for mail servers to support both Implicit and Explicit TLS for Submission, *and* to enforce TLS-encryption on ports 587 (Explicit TLS) and 465 (Implicit TLS). That's exactly `docker-mailserver`'s default configuration: abiding by RFC 8314, it enforces a strict (`encrypt`) STARTTLS policy, where a denied TLS upgrade terminates the connection thus (hopefully but at the client's discretion) preventing unencrypted (cleartext) Submission.

- **`docker-mailserver`**'s default configuration enables and *requires* Explicit TLS (STARTTLS) on port 587 for Outward Submission.
- It does not enable Implicit TLS Outward Submission on port 465 by default. One may enable it through simple custom configuration, either as a replacement or (better!) supplementary mean of secure Submission.
- It does not support old MUAs (clients) not supporting TLS encryption on ports 587/465 (those should perform Submission on port 25, more details below). One may relax that constraint through advanced custom configuration, for backwards compatibility.

A final Outward Submission setup exists and is akin SMTP+STARTTLS on port 587, but on port 25. That port has historically been reserved specifically for unencrypted (cleartext) mail exchange though, making STARTTLS a bit wrong to use. As is expected by RFC 5321, `docker-mailserver`

uses port 25 for unencrypted Submission in order to support older clients, but most importantly for unencrypted Transfer/Relay between MTAs.

- **docker-mailserver**'s default configuration also enables unencrypted (cleartext) on port 25 for Outward Submission.
- It does not enable Explicit TLS (STARTTLS) on port 25 by default. One may enable it through advanced custom configuration, either as a replacement (bad!) or as a supplementary mean of secure Outward Submission.
- One may also secure Outward Submission using advanced encryption scheme, such as DANE/DNSSEC and/or MTA-STS.

Inward Submission

Granted it's still very difficult enforcing encryption between MTAs (Transfer/Relay) without risking dropping emails (when relayed by MTAs not supporting TLS-encryption), Inward Submission is to be handled in cleartext on port 25 by default.

- **docker-mailserver**'s default configuration enables unencrypted (cleartext) on port 25 for Inward Submission.
- It does not enable Explicit TLS (STARTTLS) on port 25 by default. One may enable it through advanced custom configuration, either as a replacement (bad!) or as a supplementary mean of secure Inward Submission.
- One may also secure Inward Submission using advanced encryption scheme, such as DANE/DNSSEC and/or MTA-STS.

Overall, docker-mailserver's default configuration for SMTP looks like this:

Retrieval - IMAP

A MUA willing to fetch an email from a mail server will most likely communicate with its IMAP server. As with SMTP described earlier, communication will take place in the form of data packets exchanged over a network that both the client and the server are connected to. The IMAP protocol makes the server capable of handling *Retrieval*.

In the case of docker-mailserver, the IMAP server is Dovecot. The MUA (client) may vary, yet its Retrieval request is performed as TCP packets sent over the *public* internet. This exchange of information may be secured in order to counter eavesdropping.

Again, as with SMTP described earlier, the IMAP protocol may be secured with either Implicit TLS (aka. IMAPS/IMAP4S) or Explicit TLS (using STARTTLS).

The best practice as of 2020 is to enforce IMAPS on port 993, rather than IMAP+STARTTLS on port 143 (see RFC 8314); yet the latter is usually provided for backwards compatibility.

docker-mailserver's default configuration enables both Implicit and Explicit TLS for Retrievial, on ports 993 and 143 respectively.

Retrieval - POP3

Similarly to IMAP, the older POP3 protocol may be secured with either Implicit or Explicit TLS.

The best practice as of 2020 would be POP3S on port 995, rather than POP3+STARTTLS on port 110 (see RFC 8314).

docker-mailserver's default configuration disables POP3 altogether. One should expect MUAs to use TLS-encrypted IMAP for Retrieval.

How does **docker-mailserver** help with setting everything up?

As a *batteries included* Docker image, docker-mailserver provides you with all the required components and a default configuration, to run a decent and secure mail server.

One may then customize all aspects of its internal components.

- Simple customization is supported through docker-compose configuration and the env-mailserver configuration file.
- Advanced customization is supported through providing "monkey-patching" configuration files and/or deriving your own image from docker-mailserver's upstream, for a complete control over how things run.

On the subject of security, one might consider docker-mailserver's default configuration to *not* be 100% secure:

- it enables unencrypted traffic on port 25

- it enables Explicit TLS (STARTTLS) on port 587, instead of Implicit TLS on port 465

We believe `docker-mailserver`'s default configuration to be a good middle ground: it goes slightly beyond "old" (1999) RFC 2487; and with developer friendly configuration settings, it makes it pretty easy to abide by the "newest" (2018) RFC 8314.

Eventually, it is up to *you* deciding exactly what kind of transportation/encryption to use and/or enforce, and to customize your instance accordingly (with looser or stricter security). Be also aware that protocols and ports on your server can only go so far with security; third-party MTAs might relay your emails on insecure connections, man-in-the-middle attacks might still prove effective, etc. Advanced counter-measure such as DANE, MTA-STS and/or full body encryption (eg. PGP) should be considered as well for increased confidentiality, but ideally without compromising backwards compatibility so as to not block emails.

The README is the best starting point in configuring and running your mail server. You may then explore this wiki to cover additional topics, including but not limited to, security.

Configure-AWS-SES

Note: new configuration, see Configure Relay Hosts

Instead of letting postfix deliver mail directly it is possible to configure it to deliver outgoing email via Amazon SES (Simple Email Service). (Receiving inbound email via SES is not implemented.) The configuration follows the guidelines provided by AWS in http://docs.aws.amazon.com/ses/latest/DeveloperGuide/postfix.html, specifically, the STARTTLS method.

As described in the AWS Developer Guide you will have to generate SMTP credentials and define the following two environment variables in the docker-compose.yml with the appropriate values for your AWS SES subscription (the values for AWS_SES_USERPASS are the "SMTP username" and "SMTP password" provided when you create SMTP credentials for SES):

```
environment:
- AWS_SES_HOST=email-smtp.us-east-1.amazonaws.com
- AWS_SES_USERPASS=AKIAXXXXXXXXXXXXXXXX:kqXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

If necessary, you can also provide AWS_SES_PORT. If not provided, it defaults to 25.

When you start the container you will see a log line as follows confirming the configuration:

```
Setting up outgoing email via AWS SES host email-smtp.us-east-1.amazonaws.com
```

To verify proper operation, send an email to some external account of yours and inspect the mail headers. You will also see the connection to SES in the mail logs. For example:

```
May 23 07:09:36 mail postfix/smtp[692]: Trusted TLS connection established to email-smtp.us-east-1.amazonaws.com[107.20.142.169]:25:
TLSv1.2 with cipher ECDHE-RSA-AES256-GCM-SHA384 (256/256 bits)
May 23 07:09:36 mail postfix/smtp[692]: 8C82A7E7: to=<someone@example.com>, relay=email-smtp.us-east-1.amazonaws.com[107.20.142.16
delay=0.35, delays=0/0.02/0.13/0.2, dsn=2.0.0, status=sent (250 Ok 01000154dc729264-93fdd7ea-f039-43d6-91ed-653e8547867c-000000)
```

Configure-Accounts

Adding a new account

Users (email accounts) are managed in `/tmp/docker-mailserver/postfix-accounts.cf`.

The best way to manage accounts is to use the reliable `setup.sh` script.

Or you may directly add the *full* email address and its encrypted password, separated by a pipe.

Example:

```
user1@domain.tld|{SHA512-CRYPT}$6$2YpW1nYtPBs2yLYS$z.5PGH1OEzsHHNhl3gJrc3D.YMZkvKw/vp.r5WIiwya6z7P/CQ9GDEJDr2G2V0cAfjDFeAC
user2@otherdomain.tld|{SHA512-CRYPT}$6$2YpW1nYtPBs2yLYS$z.5PGH1OEzsHHNhl3gJrc3D.YMZkvKw/vp.r5WIiwya6z7P/CQ9GDEJDr2G2V0cAfjD
```

In the example above, we've added 2 mail accounts for 2 different domains. Consequently, the mail-server will automagically be configured as multi-domains.

Therefore, to *generate* a new mail account data, directly from your docker host, you could for example run the following:

```
docker run --rm \
  -e MAIL_USER=user1@domain.tld \
  -e MAIL_PASS=mypassword \
  -ti tvial/docker-mailserver:latest \
  /bin/sh -c 'echo "$MAIL_USER|$(doveadm pw -s SHA512-CRYPT -u $MAIL_USER -p $MAIL_PASS)"' >> config/postfix-accounts.cf
```

You will then be asked for a password, and be given back the data for a new account entry, as text.

To actually *add* this new account, just copy all the output text in `config/postfix-accounts.cf` file of your running container.

Please note the `doveadm pw` command lets you choose between several encryption schemes for the password. Use doveadm pw -l to get a list of the currently supported encryption schemes.

> Note: changes to the accounts list require a restart of the container, using `supervisord`. See #552

Mailbox quota

On top of the default quota (`POSTFIX_MAILBOX_SIZE_LIMIT`), you can define specific quotas per mailbox. Quota implementation relies on dovecot quota which requires dovecot to be enabled. Consequently, quota directives are disabled when `SMTP_ONLY=1` or when `ENABLE_LDAP=1` or when explicitly disabled with `ENABLE_QUOTAS=0` .

A warning message will be sent to the user when his mailbox is reaching quota limit. Have a look at 90-quota.cf for further details.

Commands

exec in the container

- `setquota <user@domain.tld> [<quota>]` : define the quota of a mailbox (quota format e.g. 302M (B (byte), k (kilobyte), M (megabyte), G (gigabyte) or T (terabyte)))
- `delquota <user@domain.tld>` : delete the quota of a mailbox
- `doveadm quota get -u <user@domain>` : display the quota and the statistics of a mailbox

`dovecot-quotas.cf`

This file is a key-value database where quotas are stored.

dovecot-quotas.cf

```
user@domain.tld:50M
john@other-domain.tld:1G
```

Notes

- *imap-quota* is enabled and allow clients to query their mailbox usage.
- When the mailbox is deleted, the quota directive is deleted as well.
- LDAP ? Dovecot quotas supports LDAP but it's not implemented (*PR are welcome!*).

Configure-Aliases

Please first read Postfix documentation on virtual aliases.

Configuring aliases

You can use setup.sh instead of creating and editing files manually.

Aliases are managed in `/tmp/docker-mailserver/postfix-virtual.cf`.

An alias is a *full* email address that will either be:

- delivered to an existing account registered in `/tmp/docker-mailserver/postfix-accounts.cf`
- redirected to one or more other email addresses

Alias and target are space separated.

Example (on a server with domain.tld as its domain):

```
# Alias delivered to an existing account
alias1@domain.tld user1@domain.tld

# Alias forwarded to an external email address
alias2@domain.tld external@gmail.com
```

Configuring regexp aliases

Additional regexp aliases can be configured by placing them into `config/postfix-regexp.cf`. The regexp aliases get evaluated after the virtual aliases (`/tmp/docker-mailserver/postfix-virtual.cf`).

For example, the following `config/postfix-regexp.cf` causes all email to "test" users to be delivered to qa@example.com:

```
/^test[0-9][0-9]*@example.com/ qa@example.com
```

Address tags (extension delimiters) as an alternative to aliases

Postfix supports so-called address tags, in the form of plus (+) tags - i.e. address+tag@example.com will end up at address@example.com.

This is configured by default and the (configurable !) separator is set to `+`.

For more info, see How to use Address Tagging (user+tag@example.com) with Postfix and the official documentation.

Note that if you do decide to change the configurable separator, you must add the same line to *both* `config/postfix-main.cf` and `config/dovecot.cf`, because Dovecot is acting as the delivery agent. For example, to switch to `-`, add:

```
recipient_delimiter = -
```

Configure-DKIM

DKIM is a security measure targeting email spoofing. It is greatly recommended one activates it. See the Wikipedia page for more details on DKIM.

Enabling DKIM signature

To enable DKIM signature, you must have created at least one email account.

Once its done, just run the following command to generate the signature:

```
./setup.sh config dkim
```

The script assumes you're being in the directory where the `config/` directory is located. The default keysize when generating the signature is 4096 bits for now. If you need to change it (e.g. your DNS-Provider limits the size), then provide the size as the first parameter of the command:

```
./setup.sh config dkim <keysize>
```

For LDAP systems that do not have any directly created user account you can run the following command (since `8.0.0`) to generate the signature by additionally providing the desired domain name (if you have multiple domains use the command multiple times or provide a comma-separated list of domains):

```
./setup.sh config dkim <key-size> <domain.tld>[,<domain2.tld>]
```

Now the keys are generated, you can configure your DNS server with DKIM signature, simply by adding a TXT record.

If you have direct access to your DNS zone file, then it's only a matter of pasting the content of `config/opendkim/keys/domain.tld/mail.txt` in your `domain.tld.hosts` zone.

```
$ dig mail._domainkey.domain.tld TXT
---
;; ANSWER SECTION
mail._domainkey.<DOMAIN> 300 IN TXT    "v=DKIM1; k=rsa; p=AZERTYUIOPQSDFGHJKLMWXCVBN/AZERTYUIOPQSDFGHJKLMWXCVBN/AZERTYUI
```

▶ Configuration using a web interface

After generating DKIM keys, you should restart the mail server. DNS edits may take a few minutes to hours to propagate.

Note: Sometimes the key in `config/opendkim/keys/domain.tld/mail.txt` can be on multiple lines. If so then you need to concatenate the values in the TXT record:

```
$ dig mail._domainkey.domain.tld TXT
---
;; ANSWER SECTION
mail._domainkey IN TXT ( "v=DKIM1; k=rsa; "
  "p=AZERTYUIOPQSDF..."
  "asdfQWERTYUIOPQSDF..." )
```

the target (or value) field must then have all the parts together: `v=DKIM1; k=rsa; p=AZERTYUIOPQSDF...asdfQWERTYUIOPQSDF...`

Verify-only

If you want DKIM to only *verify* incoming emails, the following version of /etc/opendkim.conf may be useful (right now there is no easy mechanism for installing it other than forking the repo):

```
# This is a simple config file verifying messages only

#LogWhy              yes
Syslog              yes
SyslogSuccess       yes

Socket              inet:12301@localhost
PidFile             /var/run/opendkim/opendkim.pid
```

```
ReportAddress          postmaster@my-domain.com
SendReports            yes

Mode                v
```

Debugging

Tools

- DKIM-verifer: A add-on for the mail client Thunderbird.

DKIM TXT Record

You can debug your TXT records with the `dig` tool.

```
dig TXT mail._domainkey.domain.tld
```

Output:

```
; <<>> DiG 9.10.3-P4-Debian <<>> TXT mail._domainkey.domain.tld
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 39669
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 512
;; QUESTION SECTION:
;mail._domainkey.domain.tld. IN TXT

;; ANSWER SECTION:
mail._domainkey.domain.tld. 3600 IN TXT "v=DKIM1; k=rsa; p=MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCxBSjG6RnWAdU3oOlqsdf2WC

;; Query time: 50 msec
;; SERVER: 127.0.1.1#53(127.0.1.1)
;; WHEN: Wed Sep 07 18:22:57 CEST 2016
;; MSG SIZE  rcvd: 310
```

Switch off DKIM

Simply remove the DKIM key by recreating (not just relaunching) the mailserver container.

Configure-DMARC

DMARC Guide: https://github.com/internetstandards/toolbox-wiki/blob/master/DMARC-how-to.md

Enabling DMARC

In `docker-mailserver`, DMARC is pre-configured out-of the box. The only thing you need to do in order to enable it, is to add new TXT entry to your DNS.

In contrast with DKIM, DMARC DNS entry does not require any keys, but merely setting the configuration values. You can either handcraft the entry by yourself or use one of available generators (like https://dmarcguide.globalcyberalliance.org/).

Typically something like this should be good to start with (don't forget to replace `@domain.com` to your actual domain)

```
_dmarc.domain.com. IN TXT "v=DMARC1; p=none; rua=mailto:dmarc.report@domain.com; ruf=mailto:dmarc.report@domain.com; sp=none; ri=
```

Or a bit more strict policies (mind `p=quarantine` and `sp=quarantine`):

```
_dmarc IN TXT "v=DMARC1; p=quarantine; rua=mailto:dmarc.report@domain.com; ruf=mailto:dmarc.report@domain.com; fo=0; adkim=r; aspf=
```

DMARC status is not being displayed instantly in Gmail for instance. If you want to check it directly after DNS entries, you can use some services around the Internet such as https://dmarcguide.globalcyberalliance.org/ or https://ondmarc.redsift.com/. In other case, email clients will show "DMARC: PASS" in ~1 day or so.

Reference: #1511

Configure-Fail2ban

Fail2ban is installed automatically and bans IP addresses for 3 hours after 3 failed attempts in 10 minutes by default. If you want to change this, you can easily edit config/fail2ban-jail.cf. You can do the same with the values from fail2ban.conf, e.g dbpurgeage. In that case you need to edit config/fail2ban-fail2ban.cf

Important: The mail container must be launched with the NET_ADMIN capability in order to be able to install the iptable rules that actually ban IP addresses. Thus either include `--cap-add=NET_ADMIN` in the docker run commandline or the equivalent docker-compose.yml:

```
  cap_add:
    - NET_ADMIN
```

If you don't you will see errors of the form

```
iptables -w -X f2b-postfix -- stderr: "getsockopt failed strangely: Operation not permitted\niptables v1.4.21: can't initialize iptabl
es table `filter': Permission denied (you must be root)\nPerhaps iptables or your kernel needs to be upgraded.\niptables v1.4.21: can'
t initialize iptables table `filter': Permission denied (you must be root)\nPerhaps iptables or your kernel needs to be upgraded.\n"
2016-06-01 00:53:51,284 fail2ban.action         [678]: ERROR   iptables -w -D INPUT -p tcp -m multiport --dports smtp,465,submission -
j f2b-postfix
```

You can also manage and list the banned IPs with the setup.sh script.

Configure-LDAP

Introduction

Getting started with ldap and this mailserver we need to take 3 parts in account:

- POSTFIX
- DOVECOT
- SASLAUTHD (this can also be handled by dovecot above)

List with the variables to control the container provisioning

POSTFIX:

- `LDAP_QUERY_FILTER_USER`
- `LDAP_QUERY_FILTER_GROUP`
- `LDAP_QUERY_FILTER_ALIAS`
- `LDAP_QUERY_FILTER_DOMAIN`

SASLAUTHD:

- `SASLAUTHD_LDAP_FILTER`

DOVECOT:

- `DOVECOT_USER_FILTER`
- `DOVECOT_PASS_FILTER`

NOTE: This page will provide several use cases like recipes to show, how this project can be used with it's LDAP Features.

Ldap Setup - Kopano/Zarafa

```
---
version: '2'

services:
  mail:
    image: tvial/docker-mailserver:latest
    hostname: mail
    domainname: domain.com
    container_name: mail

    ports:
      - "25:25"
      - "143:143"
      - "587:587"
      - "993:993"

    volumes:
      - maildata:/var/mail
      - mailstate:/var/mail-state
      - ./config/:/tmp/docker-mailserver/

    environment:
      # We are not using dovecot here
      - SMTP_ONLY=1
      - ENABLE_SPAMASSASSIN=1
      - ENABLE_CLAMAV=1
      - ENABLE_FAIL2BAN=1
      - ENABLE_POSTGREY=1
      - SASLAUTHD_PASSWD=

      # >>> SASL Authentication
      - ENABLE_SASLAUTHD=1
      - SASLAUTHD_LDAP_SERVER=<yourLdapContainer/yourLdapServer>
      - SASLAUTHD_LDAP_PROTO=
      - SASLAUTHD_LDAP_BIND_DN=cn=Administrator,cn=Users,dc=mydomain,dc=loc
```

```
      - SASLAUTHD_LDAP_PASSWORD=mypassword
      - SASLAUTHD_LDAP_SEARCH_BASE=dc=mydomain,dc=loc
      - SASLAUTHD_LDAP_FILTER=(&(sAMAccountName=%U)(objectClass=person))
      - SASLAUTHD_MECHANISMS=ldap
      # <<< SASL Authentication

      # >>> Postfix Ldap Integration
      - ENABLE_LDAP=1
      - LDAP_SERVER_HOST=<yourLdapContainer/yourLdapServer>
      - LDAP_SEARCH_BASE=dc=mydomain,dc=loc
      - LDAP_BIND_DN=cn=Administrator,cn=Users,dc=mydomain,dc=loc
      - LDAP_BIND_PW=mypassword
      - LDAP_QUERY_FILTER_USER=(&(objectClass=user)(mail=%s))
      - LDAP_QUERY_FILTER_GROUP=(&(objectclass=group)(mail=%s))
      - LDAP_QUERY_FILTER_ALIAS=(&(objectClass=user)(otherMailbox=%s))
      - LDAP_QUERY_FILTER_DOMAIN=(&(|(mail=*@%s)(mailalias=*@%s)(mailGroupMember=*@%s))(mailEnabled=TRUE))
      # <<< Postfix Ldap Integration

      # >>> Kopano Integration
      - ENABLE_POSTFIX_VIRTUAL_TRANSPORT=1
      - POSTFIX_DAGENT=lmtp:kopano:2003
      # <<< Kopano Integration

      - ONE_DIR=1
      - DMS_DEBUG=0
      - SSL_TYPE=letsencrypt
      - PERMIT_DOCKER=host

    cap_add:
      - NET_ADMIN

volumes:
  maildata:
    driver: local
  mailstate:
    driver: local
```

If your directory has not the postfix-book schema installed, then you must change the internal attribute handling for dovecot. For this you have to change the `pass_attr` and the `user_attr` mapping, as shown in the example below:

```
      - DOVECOT_PASS_ATTR=<YOUR_USER_IDENTIFYER_ATTRIBUTE>=user,<YOUR_USER_PASSWORD_ATTRIBUTE>=password
      - DOVECOT_USER_ATTR=<YOUR_USER_HOME_DIRECTORY_ATTRIBUTE>=home,<YOUR_USER_MAILSTORE_ATTRIBUTE>=mail,<YOUR_USER_M
```

The following example illustrates this for a directory that has the qmail-schema installed and that uses `uid`:

```
      - DOVECOT_PASS_ATTRS=uid=user,userPassword=password
      - DOVECOT_USER_ATTRS=homeDirectory=home,qmailUID=uid,qmailGID=gid,mailMessageStore=mail
      - DOVECOT_PASS_FILTER=(&(objectClass=qmailUser)(uid=%u)(accountStatus=active))
      - DOVECOT_USER_FILTER=(&(objectClass=qmailUser)(uid=%u)(accountStatus=active))
```

Configure-POP3

We do not recommend using POP. Use IMAP instead.

If you really want to have POP3 running, add 3 lines to the docker-compose.yml :
Add the ports 110 and 995, and add environment variable ENABLE_POP :

```
mail:
  [...]
  ports:
  - "25:25"
  - "143:143"
  - "587:587"
  - "993:993"
  - "110:110"
  - "995:995"
  environment:
  - ENABLE_POP3=1
```

Configure-Relay-Hosts

Introduction

Rather than having Postfix deliver mail directly, you can configure Postfix to send mail via another mail relay (smarthost). Examples include Mailgun, Sendgrid and AWS SES.

Depending on the domain of the sender, you may want to send via a different relay, or authenticate in a different way.

Basic Configuration

Basic configuration is done via environment variables:

- RELAY_HOST *default host to relay mail through, empty will disable this feature*
- RELAY_PORT *port on default relay, defaults to port 25*
- RELAY_USER *username for the default relay*
- RELAY_PASSWORD *password for the default user*

Setting these environment variables will cause mail for all sender domains to be routed via the specified host, authenticating with the user/password combination.

Note for users of the previous AWS_SES_* variables: please update your configuration to use these new variables, no other configuration is required.

Advanced Configuration

Sender-dependent Authentication

Sender dependent authentication is done in `config/postfix-sasl-password.cf`. You can create this file manually, or use

```
setup.sh relay add-auth <domain> <username> [<password>]
```

An example configuration file looks like this:

```
@domain1.com        relay_user_1:password_1
@domain2.com        relay_user_2:password_2
```

If there is no other configuration, this will cause Postfix to deliver email throught the relay specified in `RELAY_HOST` env variable, authenticating as `relay_user_1` when sent from domain1.com and authenticating as `relay_user_2` when sending from domain2.com.

NOTE to activate the configuration you must either restart the container, or you can also trigger an update by modifying a mail account.

Sender-dependent Relay Host

Sender dependent relay hosts are configured in `config/postfix-relaymap.cf`. You can create this file manually, or use

```
setup.sh relay add-domain <domain> <host> [<port>]
```

An example configuration file looks like this:

```
@domain1.com      [relay1.org]:587
@domain2.com      [relay2.org]:2525
```

Combined with the previous configuration in `config/postfix-sasl-password.cf`, this will cause Postfix to deliver mail sent from domain1.com via `relay1.org:587`, authenticating as `relay_user_1`, and mail sent from domain2.com via `relay2.org:2525` authenticating as `relay_user_2`.
NOTE You still have to define RELAY_HOST to activate the feature

Excluding Sender Domains

If you want mail sent from some domains to be delivered directly, you can exclude them from being delivered via the default relay by adding them to `config/postfix-relaymap.cf` with no destination. You can also do this via

```
setup.sh relay exclude-domain <domain>
```

Extending the configuration file from above:

```
@domain1.com      [relay1.org]:587
@domain2.com      [relay2.org]:2525
@domain3.com
```

This will cause email sent from domain3.com to be delivered directly.

References

Thanks to the author of this article for the inspiration. This is also worth reading to understand a bit more about how to set up Mailgun to work with this.

Configure-SPF

From Wikipedia:

> Sender Policy Framework (SPF) is a simple email-validation system designed to detect email spoofing by providing a mechanism to allow receiving mail exchangers to check that incoming mail from a domain comes from a host authorized by that domain's administrators. The list of authorized sending hosts for a domain is published in the Domain Name System (DNS) records for that domain in the form of a specially formatted TXT record. Email spam and phishing often use forged "from" addresses, so publishing and checking SPF records can be considered anti-spam techniques.

For a more technical review: https://github.com/internetstandards/toolbox-wiki/blob/master/SPF-how-to.md

Add a SPF record

To add a SPF record in your DNS, insert the following line in your DNS zone:

```
; MX record must be declared for SPF to work
domain.com. IN  MX 1 mail.domain.com.

; SPF record
domain.com. IN TXT "v=spf1 mx ~all"
```

This enables the *Softfail* mode for SPF. You could first add this SPF record with a very low TTL.
SoftFail is a good setting for getting started and testing, as it lets all email through, with spams tagged as such in the mailbox.

After verification, you *might* want to change your SPF record to `v=spf1 mx -all` so as to enforce the *HardFail* policy. See http://www.open-spf.org/SPF_Record_Syntax/ for more details about SPF policies.

In any case, increment the SPF record's TTL to its final value.

Backup MX, Secondary MX

For whitelisting a IP-Address from the SPF test, you can create a config file (see policyd-spf.conf) and mount that file into `/etc/postfix-policyd-spf-python/policyd-spf.conf`.

Example:

Create and edit a policyd-spf.conf file here `/<your Docker-Mailserver dir>/config/postfix-policyd-spf.conf`:

```
debugLevel = 1
#0(only errors)-4(complete data received)

skip_addresses = 127.0.0.0/8,::ffff:127.0.0.0/104,::1

# Preferably use IP-Addresses for whitelist lookups:
Whitelist = 192.168.0.0/31,192.168.1.0/30
# Domain_Whitelist = mx1.mybackupmx.com,mx2.mybackupmx.com
```

Then add this line to `docker-compose.yml` below the `volumes:` section

```
- ./config/postfix-policyd-spf.conf:/etc/postfix-policyd-spf-python/policyd-spf.conf
```

Configure-SSL

There are multiple options to enable SSL:

- using letsencrypt (recommended)
- using Caddy
- using Traefik
- using self-signed certificates with the provided tool
- using your own certificates

After installation, you can test your setup with:

- checktls.com
- testssl.sh

Let's encrypt (recommended)

To enable Let's Encrypt on your mail server, you have to:

- get your certificate using letsencrypt client
- add an environment variable `SSL_TYPE` with value `letsencrypt` (see `docker-compose.yml.dist`)
- mount your whole `letsencrypt` folder to `/etc/letsencrypt`
- the certs folder name located in `letsencrypt/live/` must be the `fqdn` of your container responding to the `hostname` command. The full qualified domain name (`fqdn`) inside the docker container is built combining the `hostname` and `domainname` values of the docker-compose file, e. g.: hostname: `mail`; domainname: `myserver.tld`; fqdn: `mail.myserver.tld`

You don't have anything else to do. Enjoy.

Example using docker for letsencrypt

Make a directory to store your letsencrypt logs and configs.

In my case

```
mkdir -p /home/ubuntu/docker/letsencrypt
cd /home/ubuntu/docker/letsencrypt
```

Now get the certificate (modify `mail.myserver.tld`) and following the certbot instructions. This will need access to port 80 from the internet, adjust your firewall if needed

```
docker run --rm -ti -v $PWD/log/:/var/log/letsencrypt/ -v $PWD/etc/:/etc/letsencrypt/ -p 80:80 certbot/certbot certonly --standalone -d mail.myserve
```

You can now mount /home/ubuntu/docker/letsencrypt/etc/ in /etc/letsencrypt of `docker-mailserver`

To renew your certificate just run (this will need access to port 443 from the internet, adjust your firewall if needed)

```
docker run --rm -ti -v $PWD/log/:/var/log/letsencrypt/ -v $PWD/etc/:/etc/letsencrypt/ -p 80:80 -p 443:443 certbot/certbot renew
```

Example using docker, nginx-proxy and letsencrypt-nginx-proxy-companion

If you are running a web server already, it is non-trivial to generate a Let's Encrypt certificate for your mail server using `certbot`, because port 80 is already occupied. In the following example, we show how `docker-mailserver` can be run alongside the docker containers `nginx-proxy` and `letsencrypt-nginx-proxy-companion`.

There are several ways to start `nginx-proxy` and `letsencrypt-nginx-proxy-companion`. Any method should be suitable here. For example start `nginx-proxy` as in the `letsencrypt-nginx-proxy-companion` documentation:

```
docker run --detach \
  --name nginx-proxy \
  --restart always \
  --publish 80:80 \
  --publish 443:443 \
```

```
  --volume /server/letsencrypt/etc:/etc/nginx/certs:ro \
  --volume /etc/nginx/vhost.d \
  --volume /usr/share/nginx/html \
  --volume /var/run/docker.sock:/tmp/docker.sock:ro \
  jwilder/nginx-proxy
```

Then start `nginx-proxy-letsencrypt`:

```
docker run --detach \
  --name nginx-proxy-letsencrypt \
  --restart always \
  --volume /server/letsencrypt/etc:/etc/nginx/certs:rw \
  --volumes-from nginx-proxy \
  --volume /var/run/docker.sock:/var/run/docker.sock:ro \
  jrcs/letsencrypt-nginx-proxy-companion
```

Start the rest of your web server containers as usual.

Start another container for your `mail.myserver.tld`. This will generate a Let's Encrypt certificate for your domain, which can be used by `docker-mailserver`. It will also run a web server on port 80 at that address.:

```
docker run -d \
  --name webmail \
  -e "VIRTUAL_HOST=mail.myserver.tld" \
  -e "LETSENCRYPT_HOST=mail.myserver.tld" \
  -e "LETSENCRYPT_EMAIL=foo@bar.com" \
  library/nginx
```

You may want to add `-e LETSENCRYPT_TEST=true` to the above while testing to avoid the Let's Encrypt certificate generation rate limits.

Finally, start the mailserver with the docker-compose.yml Make sure your mount path to the letsencrypt certificates is correct. Inside your /path/to/mailserver/docker-compose.yml (for the mailserver from this repo) make sure volumes look like below example;

```
  volumes:
  - maildata:/var/mail
  - mailstate:/var/mail-state
  - ./config/:/tmp/docker-mailserver/
  - /server/letsencrypt/etc:/etc/letsencrypt/live
```

Then

/path/to/mailserver/docker-compose up -d mail

Example using docker, nginx-proxy and letsencrypt-nginx-proxy-companion with docker-compose

The following docker-compose.yml is the basic setup you need for using letsencrypt-nginx-proxy-companion. It is mainly derived from its own wiki/documenation.

```
version: "2"

services:
  nginx:
    image: nginx
    container_name: nginx
    ports:
      - 80:80
      - 443:443
    volumes:
      - /mnt/data/nginx/htpasswd:/etc/nginx/htpasswd
      - /mnt/data/nginx/conf.d:/etc/nginx/conf.d
      - /mnt/data/nginx/vhost.d:/etc/nginx/vhost.d
      - /mnt/data/nginx/html:/usr/share/nginx/html
      - /mnt/data/nginx/certs:/etc/nginx/certs:ro
    networks:
      - proxy-tier
```

```
    restart: always

  nginx-gen:
    image: jwilder/docker-gen
    container_name: nginx-gen
    volumes:
      - /var/run/docker.sock:/tmp/docker.sock:ro
      - /mnt/data/nginx/templates/nginx.tmpl:/etc/docker-gen/templates/nginx.tmpl:ro
    volumes_from:
      - nginx
    entrypoint: /usr/local/bin/docker-gen -notify-sighup nginx -watch -wait 5s:30s /etc/docker-gen/templates/nginx.tmpl /etc/nginx/conf.d/default.co
    restart: always

  letsencrypt-nginx-proxy-companion:
    image: jrcs/letsencrypt-nginx-proxy-companion
    container_name: letsencrypt-companion
    volumes_from:
      - nginx
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock:ro
      - /mnt/data/nginx/certs:/etc/nginx/certs:rw
    environment:
      - NGINX_DOCKER_GEN_CONTAINER=nginx-gen
      - DEBUG=false
    restart: always

networks:
  proxy-tier:
    external:
      name: nginx-proxy
```

The second part of the setup is the actual mail container. So, in another folder, create another docker-compose.yml with the following content (Removed all ENV variables for this example):

```
version: '2'
services:
  mail:
    image: tvial/docker-mailserver:latest
    hostname: ${HOSTNAME}
    domainname: ${DOMAINNAME}
    container_name: ${CONTAINER_NAME}
    ports:
    - "25:25"
    - "143:143"
    - "465:465"
    - "587:587"
    - "993:993"
    volumes:
    - ./mail:/var/mail
    - ./mail-state:/var/mail-state
    - ./config/:/tmp/docker-mailserver/
    - /mnt/data/nginx/certs/:/etc/letsencrypt/live/:ro
    cap_add:
    - NET_ADMIN
    - SYS_PTRACE
    restart: always

  cert-companion:
    image: nginx
    environment:
      - "VIRTUAL_HOST="
      - "VIRTUAL_NETWORK=nginx-proxy"
      - "LETSENCRYPT_HOST="
      - "LETSENCRYPT_EMAIL="
    networks:
      - proxy-tier
    restart: always

networks:
```

```
  proxy-tier:
    external:
      name: nginx-proxy
```

The mail container needs to have the letsencrypt certificate folder mounted as a volume. No further changes are needed. The second container is a dummy-sidecar we need, because the mail-container do not expose any web-ports. Set your ENV variables as you need. (VIRTUAL_HOST and LETSENCRYPT_HOST are mandandory, see documentation)

Example using the letsencrypt certificates on a Synology NAS

Version 6.2 and later of the Synology NAS DSM OS now come with an interface to generate and renew letencrypt certificates. Navigation into your DSM control panel and go to Security, then click on the tab Certificate to generate and manage letsencrypt certificates. Amongst other things, you can use these to secure your mail server. DSM locates the generated certificates in a folder below `/usr/syno/etc/certificate/_archive/` . Navigate to that folder and note the 6 character random folder name of the certificate you'd like to use. Then, add the following to your `docker-compose.yml` declaration file:

```
volumes:
     - /usr/syno/etc/certificate/_archive/YOUR_FOLDER/:/tmp/ssl
...
environment:
     - SSL_TYPE=manual
     - SSL_CERT_PATH=/tmp/ssl/fullchain.pem
     - SSL_KEY_PATH=/tmp/ssl/privkey.pem
```

DSM-generated letsencrypt certificates get auto-renewed every three months.

Caddy

If you are using Caddy to renew your certificates, please note that only RSA certificates work. Read issue 1440 for details. In short for Caddy v1 the Caddyfile should look something like:

```
https://mail.domain.com {
   tls yourcurrentemail@gmail.com {
      key_type  rsa2048
   }
}
```

For Caddy v2 you can specify the key_type in your server's global settings, which would end up looking something like this if you're using a Caddyfile:

```
{
debug
admin localhost:2019
http_port 80
https_port 443
default_sni mywebserver.com
key_type rsa4096

}
```

If you are instead using a json config for Caddy v2, you can set it in your site's TLS automation policies:

```
{
 "apps": {
  "http": {
   "servers": {
    "srv0": {
     "listen": [
      ":443"
     ],
     "routes": [
      {
       "match": [
```

```
            {
              "host": [
                "mail.domain.com",
              ]
            }
          ],
          "handle": [
            {
              "handler": "subroute",
              "routes": [
                {
                  "handle": [
                    {
                      "body": "",
                      "handler": "static_response"
                    }
                  ]
                }
              ]
            }
          ],
          "terminal": true
        },
      ]
    }
  }
 },
 "tls": {
  "automation": {
   "policies": [
    {
     "subjects": [
      "mail.domain.com",
     ],
     "key_type": "rsa2048",
     "issuer": {
      "email": "email@email.com",
      "module": "acme"
     }
    },
    {
     "issuer": {
      "email": "email@email.com",
      "module": "acme"
     }
    }
   ]
  }
 }
}
```

The generated certificates can be mounted:

```
volumes:
  - ${CADDY_DATA_DIR}/certificates/acme-v02.api.letsencrypt.org-directory/mail.domain.com/mail.domain.com.crt:/etc/letsencrypt/live/mail.doma
  - ${CADDY_DATA_DIR}/certificates/acme-v02.api.letsencrypt.org-directory/mail.domain.com/mail.domain.com.key:/etc/letsencrypt/live/mail.dom
```

EC certificates fail in the TLS handshake:

```
CONNECTED(00000003)
140342221178112:error:14094410:SSL routines:ssl3_read_bytes:sslv3 alert handshake failure:ssl/record/rec_layer_s3.c:1543:SSL alert number 4
no peer certificate available
No client certificate CA names sent
```

Traefik

Traefik is an open-source Edge Router which handles ACME protocol using lego.

Traefik can request certificates for domains through the ACME protocol (see Traefik's documentation about its ACME negotiation & storage mechanism). Traefik's router will take care of renewals, challenge negotiations, etc.

Traefik v2

(For Traefik v1 see next section)

Traefik's V2 storage format is natively supported if the `acme.json` store is mounted into the container at `/etc/letsencrypt/acme.json`. The file is also monitored for changes and will trigger a reload of the mail services. Lookup of the certificate domain happens in the following order:

1. $SSL_DOMAIN
2. $HOSTNAME
3. $DOMAINNAME

This allows for support of wild card certificates: `"SSL_DOMAIN=*.example.com"`. Here is an example setup for docker-compose:

```
version: '3.8'
services:
  mail:
    image: tvial/docker-mailserver:stable
    hostname: mail
    domainname: example.com
    volumes:
    - /etc/ssl/acme-v2.json:/etc/letsencrypt/acme.json:ro
    environment:
      SSL_TYPE: letsencrypt
      # SSL_DOMAIN: "*.example.com"
  traefik:
    image: traefik:v2.2
    restart: always
    ports:
    - "80:80"
    - "443:443"
    command:
    - --providers.docker
    - --entrypoints.web.address=:80
    - --entrypoints.web.http.redirections.entryPoint.to=websecure
    - --entrypoints.web.http.redirections.entryPoint.scheme=https
    - --entrypoints.websecure.address=:443
    - --entrypoints.websecure.http.middlewares=hsts@docker
    - --entrypoints.websecure.http.tls.certResolver=le
    - --certificatesresolvers.le.acme.email=admin@example.net
    - --certificatesresolvers.le.acme.storage=/acme.json
    - --certificatesresolvers.le.acme.httpchallenge.entrypoint=web
    volumes:
    - /var/run/docker.sock:/var/run/docker.sock:ro
    - /etc/ssl/acme-v2.json:/acme.json

  whoami:
    image: containous/whoami
    labels:
    - "traefik.http.routers.whoami.rule=Host(`mail.example.com`)"
```

This setup only comes with one caveat: The domain has to be configured on another service for traefik to actually request it from lets-encrypt (`whoami` in this case).

Traefik V1

If you are using Traefik v1, you might want to *push* your Traefik-managed certificates to the mailserver container, in order to reuse them. Not an easy task, but fortunately, youtous/mailserver-traefik is a certificate renewal service for docker-mailserver.

Depending of your Traefik configuration, certificates may be stored using a file or a KV Store (consul, etcd...) Either way, certificates will be renewed by Traefik, then automatically pushed to the mailserver thanks to the cert-renewer service. Finally, dovecot and postfix will be restarted.

Documentation: https://github.com/youtous/docker-mailserver-traefik

Self-signed certificates (testing only)

You can easily generate a self-signed SSL certificate by using the following command:

```
docker run -ti --rm -v "$(pwd)"/config/ssl:/tmp/docker-mailserver/ssl -h mail.my-domain.com -t tvial/docker-mailserver generate-ssl-certificate

# Press enter
# Enter a password when needed
# Fill information like Country, Organisation name
# Fill "my-domain.com" as FQDN for CA, and "mail.my-domain.com" for the certificate.
# They HAVE to be different, otherwise you'll get a `TXT_DB error number 2`
# Don't fill extras
# Enter same password when needed
# Sign the certificate? [y/n]:y
# 1 out of 1 certificate requests certified, commit? [y/n]y

# will generate:
# config/ssl/mail.my-domain.com-key.pem (used in postfix)
# config/ssl/mail.my-domain.com-req.pem (only used to generate other files)
# config/ssl/mail.my-domain.com-cert.pem (used in postfix)
# config/ssl/mail.my-domain.com-combined.pem (used in courier)
# config/ssl/demoCA/cacert.pem (certificate authority)
```

Note that the certificate will be generate for the container `fqdn`, that is passed as `-h` argument. Check the following page for more information regarding postfix and SSL/TLS configuration.

To use the certificate:

- add `SSL_TYPE=self-signed` to your container environment variables
- if a matching certificate (files listed above) is found in `config/ssl`, it will be automatically setup in postfix and dovecot. You just have to place them in `config/ssl` folder.

Custom certificate files

You can also provide your own certificate files. Add these entries to your `docker-compose.yml`:

```
volumes:
  - /etc/ssl:/tmp/ssl:ro
environment:
- SSL_TYPE=manual
- SSL_CERT_PATH=/tmp/ssl/cert/public.crt
- SSL_KEY_PATH=/tmp/ssl/private/private.key
```

This will mount the path where your ssl certificates reside as read-only under `/tmp/ssl`. Then all you have to do is to specify the location of your private key and the certificate.

Please note that you may have to restart your mailserver once the certificates change.

Testing certificate

From your host:

```
docker exec mail openssl s_client -connect 0.0.0.0:25 -starttls smtp -CApath /etc/ssl/certs/
```

or

```
docker exec mail openssl s_client -connect 0.0.0.0:143 -starttls imap -CApath /etc/ssl/certs/
```

And you should see the certificate chain, the server certificate and:

```
Verify return code: 0 (ok)
```

In addition, to verify certificate dates:

```
docker exec mail openssl s_client -connect 0.0.0.0:25 -starttls smtp -CApath /etc/ssl/certs/ 2>/dev/null | openssl x509 -noout  -dates
```

Plain text access

Not recommended for purposes other than testing.

Just add this to config/dovecot.cf:

```
ssl = yes
disable_plaintext_auth=no
```

These options in conjunction mean:

```
ssl=yes and disable_plaintext_auth=no: SSL/TLS is offered to the client, but the client isn't required to use it. The client is allowed to login with pla
```

Importing certificates obtained via another source

If you have another source for SSL/TLS certificates you can import them into the server via an external script. The external script can be found here: external certificate import script

The steps to follow are these:

1. Transport the new certificates to ./config/sll (/tmp/ssl in the container)
2. You should provide fullchain.key and privkey.pem
3. Place the script in ./config/ (or /tmp/docker-mailserver/ inside the container)
4. Make the script executable (chmod +x tomav-renew-certs.sh)
5. Run the script: docker exec mail /tmp/docker-mailserver/tomav-renew-certs.sh

If an error occurs the script will inform you. If not you will see both postfix and dovecot restart.

After the certificates have been loaded you can check the certificate:

```
openssl s_client -servername mail.mydomain.net -connect 192.168.0.72:465 2>/dev/null | openssl x509

# or

openssl s_client -servername mail.mydomain.net -connect mail.mydomain.net:465 2>/dev/null | openssl x509
```

Or you can check how long the new certificate is valid with commands like:

```
export SITE_URL="mail.mydomain.net"
export SITE_IP_URL="192.168.0.72"  ## can also be  mail.mydomain.net
export SITE_SSL_PORT="465"  ##imap port dovecot

##works: check if certificate will expire in two weeks
#2 weeks is 1209600 seconds
#3 weeks is 1814400
#12 weeks is 7257600
#15 weeks is 9072000

certcheck_2weeks=`openssl s_client -connect ${SITE_IP_URL}:${SITE_SSL_PORT} \
  -servername ${SITE_URL} 2> /dev/null |  openssl x509 -noout  -checkend 1209600`

####################################
#notes:  output can be
#Certificate will not expire
#Certificate will expire
####################
```

What does the script that imports the certificates do:

1. Check if there are new certs in the /tmp/ssl folder
2. check with the ssl cert fingerprint if they differ from the current certificates
3. if so it will copy the certs to the right places
4. and restart postfix and dovecot

You can ofcourse run the script by cron once a week or something. In that way you could automate cert renewal. If you do so it is probably wise to run an automated check on certificate expiry as well. Such a check could look something like this:

```
## code below will alert if certificate expires in less than two weeks
## please adjust varables!
## make sure the mail -s command works! Test!

export SITE_URL="mail.mydomain.net"
export SITE_IP_URL="192.168.2.72"  ## can also be  mail.mydomain.net
export SITE_SSL_PORT="465"  ##imap port dovecot
export ALERT_EMAIL_ADDR="bill@gates321boom.com"

certcheck_2weeks=`openssl s_client -connect ${SITE_IP_URL}:${SITE_SSL_PORT} \
  -servername ${SITE_URL} 2> /dev/null |  openssl x509 -noout  -checkend 1209600`

#####################################
#notes:  output can be
#Certificate will not expire
#Certificate will expire
#####################

#echo "certcheck 2 weeks gives $certcheck_2weeks"

##automated check you might run by cron or something
## does tls/ssl certificate expire within two weeks?

if [  "$certcheck_2weeks" = "Certificate will not expire" ]; then
  echo "all is wel, certwatch 2 weeks says $certcheck_2weeks"
  else
    echo "Cert seems to be expiring pretty soon, within two weeks: $certcheck_2weeks"
    echo "we will send an alert email and log as well"
    logger Certwatch: cert $SITE_URL will expire in two weeks
    echo "Certwatch: cert $SITE_URL will expire in two weeks" | mail -s "cert $SITE_URL expires in two weeks " $ALERT_EMAIL_ADDR
fi
```

Configure-Sieve-filters

User-defined sieve filters

Sieve allows to specify filtering rules for incoming emails that allow for example sorting mails into different folders depending on the title of an email. There are global and user specific filters which are filtering the incoming emails in the following order:

- Global-before -> User specific -> Global-after

Global filters are applied to EVERY incoming mail for EVERY email address. To specify a global Sieve filter provide a `config/before.dovecot.sieve` or a `config/after.dovecot.sieve` file with your filter rules. If any filter in this filtering chain discards an incoming mail, the delivery process will stop as well and the mail will not reach any following filters(e.g. global-before stops an incoming spam mail: The mail will get discarded and a user-specific filter won't get applied.)

To specify a user-defined Sieve filter place a `.dovecot.sieve` file into a virtual user's mail folder e.g. `/var/mail/domain.com/user1/.dovecot.sieve`. I this file exists dovecot will apply the filtering rules.

It's even possible to install a user provided Sieve filter at startup during users setup: simply include a Sieve file in the `config` path for each user login that need a filter. The file name provided should be in the form <user_login>.dovecot.sieve, so for example for `user1@domain.tld` you should provide a Sieve file named `config/user1@domain.tld.dovecot.sieve`.

An example of a sieve filter that moves mails to a folder `INBOX/spam` depending on the sender address:

```
require ["fileinto", "reject"];

if address :contains ["From"] "spam@spam.com" {
  fileinto "INBOX.spam";
} else {
    keep;
}
```

Note: that folders have to exist beforehand if sieve should move them.

Another example of a sieve filter that forward mails to a different address:

```
require ["copy"];

redirect :copy "user2@otherdomain.tld";
```

Just forward all incoming emails and do not save them locally:

```
redirect "user2@otherdomain.tld";
```

You can also use external programs to filter or pipe (process) messages by adding executable scripts in `config/sieve-pipe` or `config/sieve-filter`. This can be used in lieu of a local alias file, for instance to forward an email to a webservice. These programs can then be referenced by filename, by all users. Note that the process running the scripts run as a privileged user. For further information see Dovecot's wiki.

```
require ["vnd.dovecot.pipe"];
pipe "external-program";
```

For more examples or a detailed description of the Sieve language have a look at the official site. Other resources are available on the internet where you can find several examples.

Manage Sieve

The Manage Sieve extension allows users to modify their Sieve script by themselves. The authentication mechanisms are the same as for the main dovecot service. ManageSieve runs on port `4190` and needs to be enabled using the `ENABLE_MANAGESIEVE=1` environment variable.

```
(docker-compose.yml)
ports:
 - ...
 - "4190:4190"
environment:
```

```
  - ...
  - ENABLE_MANAGESIEVE=1
```

All user defined sieve scripts that are managed by ManageSieve are stored in the user's home folder in `/var/mail/domain.com/user1/sieve`. Just one sieve script might be active for a user and is sym-linked to `/var/mail/domain.com/user1/.dovecot.sieve` automatically.

Note: ManageSieve makes sure to not overwrite an existing `.dovecot.sieve` file. If a user activates a new sieve script the old one is backuped and moved to the `sieve` folder.

The extension is known to work with the following ManageSieve clients:

- Sieve Editor a portable standalone application based on the former Thunderbird plugin (https://github.com/thsmi/sieve).

Configure-autodiscover

Email auto-discovery means a client email is able to automagically find out about what ports and security options to use, based on the mail server URL. It can help simplify the tedious / confusing task of adding own's email account for non-tech savvy users.

Basically, email clients will search for auto-discoverable settings and prefill almost everything when a user enters its email address ❤️

There exists autodiscover-email-settings on hub.docker.com which provides IMAP/POP/SMTP/LDAP autodiscover capabilities on Microsoft Outlook/Apple Mail, autoconfig capabilities for Thunderbird or kmail and configuration profiles for iOS/Apple Mail.

Debugging

..todo.. - Please contribute more to help others debug this package

Enable verbose debugging output

You may find it useful to enable the DMS_DEBUG environment variable.

Invalid username or Password

1. Login Container

```
docker exec -it <mycontainer> bash
```

2. Check log files

`/var/log/mail` could not find any mention of incorrect logins here neither in the dovecot logs

3. Check the supervisors logfiles `/var/log/supervisor` You can find the logs for startup of fetchmail, postfix and others here - they might indicate problems during startup

4. Make sure you set your hostname to 'mail' or whatever you specified in your docker-compose.yml file or else your FQDN will be wrong

Installation Errors

1. During setup, if you get errors trying to edit files inside of the container, you likely need to install vi:

```
sudo su
docker exec -it <mycontainer> apt-get install -y vim
```

Testing Connection

I spent HOURS trying to debug "Connection Refused" and "Connection closed by foreign host" errors when trying to use telnet to troubleshoot my connection. I was also trying to connect from my email client (macOS mail) around the same time. Telnet had also worked earlier, so I was extremely confused as to why it suddenly stopped working. I stumbled upon fail2ban.log in my container. In short, when trying to get my macOS client working, I exceeded the number of failed login attempts and fail2ban put dovecot and postfix in jail! I got around it by whitelisting my ipaddresses (my ec2 instance and my local computer)

```
sudo su
docker exec -ti mail bash
cd /var/log
cat fail2ban.log | grep dovecot

# Whitelist ip addresses:
fail2ban-client set dovecot addignoreip <server ip>  # Server
fail2ban-client set postfix addignoreip <server ip>
fail2ban-client set dovecot addignoreip <client ip>  # Client
fail2ban-client set postfix addignoreip <client ip>

# this will delete the jails entirely - nuclear option
fail2ban-client stop dovecot
fail2ban-client stop postfix
```

Send email is never received

Some hosting provides have a stealth block on port 25. Make sure to check with your hosting provider that traffic on port 25 is allowed

Common hosting providers known to have this issue:

- Azure
- AWS EC2

FAQ-and-Tips

What kind of database are you using?

None! No database is required. Filesystem is the database.
This image is based on config files that can be persisted using Docker volumes, and as such versioned, backed up and so forth.

Where are emails stored?

Mails are stored in `/var/mail/${domain}/${username}`.
You should use a data volume container for `/var/mail` to persist data. Otherwise, your data may be lost.

How to alter the running mailserver instance *without* relaunching the container?

docker-mailserver aggregates multiple "sub-services", such as Postfix, Dovecot, Fail2ban, SpamAssasin, etc. In many cases, on may edit a sub-service's config and reload that very sub-service, without stopping and relaunching the whole mail server.

In order to do so, you'll probably want to push your config updates to your server through a Docker volume, then restart the sub-service to apply your changes, using `supervisorctl`. For instance, after editing fail2ban's config: `supervisorctl restart fail2ban`.

See supervisorctl's documentation.

Tips: to add/update/delete an email account, there is no need to restart postfix/dovecot service inside the container after using setup.sh script.
For more information, see issues/1639

How can I sync container with host date/time? Timezone?

Share the host's `/etc/localtime` with the docker-mailserver container, using a Docker volume:

```
volumes:
  - /etc/localtime:/etc/localtime:ro
```

(optional) Add one line to `.env` or `env-mailserver` to set timetzone for container, for example:

```
TZ=Europe/Berlin
```

check here for `tz name list`

What is the file format?

All files are using the Unix format with `LF` line endings. Please do not use `CRLF`.

What about backups?

Assuming that you use `docker-compose` and a data volumes, you can backup your user mails like this:

```
docker run --rm -ti \
  -v maildata:/var/mail \
  -v mailstate:/var/mail-state \
  -v /backup/mail:/backup \
  alpine:3.2 \
  tar czf /backup/mail-`date +%y%m%d-%H%M%S`.tgz /var/mail /var/mail-state

find /backup/mail -type f -mtime +30 -exec rm -f {} \;
```

What about **mail-state** folder?

This folder consolidates all data generated by the server itself to persist when you upgrade. Example of data folder persisted: lib-amavis, lib-clamav, lib-fail2ban, lib-postfix, lib-postgrey, lib-spamassasin, lib-spamassassin, spool-postfix, ...

How can I configure my email client?

Login are full email address (`user@domain.com`).

```
# imap
username:         <user1@domain.tld>
password:         <mypassword>
server:          <mail.domain.tld>
imap port:        143 or 993 with ssl (recommended)
imap path prefix:   INBOX

# smtp
smtp port:        25 or 587 with ssl (recommended)
username:         <user1@domain.tld>
password:         <mypassword>
```

Please use `STARTTLS`.

How can I manage my custom Spamassassin rules?

Antispam rules are managed in `config/spamassassin-rules.cf`.

What are acceptable `SA_SPAM_SUBJECT` values?

For no subject set `SA_SPAM_SUBJECT=undef`.

For a trailing white-space subject one can define the whole variable with quotes in `docker-compose.yml`:

```
environment:
 - "SA_SPAM_SUBJECT=[SPAM] "
```

Can I use naked/bare domains (no host name)?

Yes, but not without some configuration changes. Normally it is assumed that docker-mailserver runs on a host with a name, so the fully qualified host name might be `mail.example.com` with the domain `example.com`. The MX records point to `mail.example.com`. To use a bare domain where the host name is `example.com` and the domain is also `example.com`, change mydestination from:

`mydestination = $myhostname, localhost.$mydomain, localhost`

To:

`mydestination = localhost.$mydomain, localhost`

Add the latter line to config/postfix-main.cf. That should work. Without that change there will be warnings in the logs like:

`warning: do not list domain example.com in BOTH mydestination and virtual_mailbox_domains`

Plus of course mail delivery fails.

Why are Spamassassin x-headers not inserted into my sample.domain.com subdomain emails?

In the default setup, amavis only applies Spamassassin x-headers into domains matching the template listed in the config file 05-domain_id (in the amavis defaults). The default setup @local_domains_acl = (".$mydomain"); does not match subdomains. To match subdomains, you can override the @local_domains_acl directive in the amavis user config file 50-user with @local_domains_maps = ("."); to match any sort of domain template.

How can I make SpamAssassin learn spam?

Put received spams in `.Junk/` imap folder using `SPAMASSASSIN_SPAM_TO_INBOX=1` and `MOVE_SPAM_TO_JUNK=1` and add a *user* cron like the following:

```
# This assumes you're having `environment: ONE_DIR=1` in the env-mailserver,
# with a consolidated config in `/var/mail-state`
#
# m h dom mon dow command
# Everyday 2:00AM, learn spam from a specific user
0 2 * * * docker exec mail sa-learn --spam /var/mail/domain.com/username/.Junk --dbpath /var/mail-state/lib-amavis/.spamassassin
```

If you run the server with docker-compose, you can leverage on docker configs and the mailserver's own cron. This is less problematic than the simple solution shown above, because it decouples the learning from the host on which the mailserver is running and avoids errors if the server is

not running.

The following configuration works nicely:

create a *system* cron file:

```
# in the docker-compose.yml root directory
mkdir cron
touch cron/sa-learn
chown root:root cron/sa-learn
chmod 0644 cron/sa-learn
```

edit the system cron file `nano cron/sa-learn`, and set an appropriate configuration:

```
# This assumes you're having `environment: ONE_DIR=1` in the env-mailserver,
# with a consolidated config in `/var/mail-state`
#
# m h dom mon dow user command
#
# Everyday 2:00AM, learn spam from a specific user
# spam: junk directory
0  2 * * * root  sa-learn --spam /var/mail/domain.com/username/.Junk --dbpath /var/mail-state/lib-amavis/.spamassassin
# ham: archive directories
15 2 * * * root  sa-learn --ham /var/mail/domain.com/username/.Archive* --dbpath /var/mail-state/lib-amavis/.spamassassin
# ham: inbox subdirectories
30 2 * * * root  sa-learn --ham /var/mail/domain.com/username/cur* --dbpath /var/mail-state/lib-amavis/.spamassassin
#
# Everyday 3:00AM, learn spam from all users of a domain
# spam: junk directory
0  3 * * * root  sa-learn --spam /var/mail/otherdomain.com/*/.Junk --dbpath /var/mail-state/lib-amavis/.spamassassin
# ham: archive directories
15 3 * * * root  sa-learn --ham /var/mail/otherdomain.com/*/.Archive* --dbpath /var/mail-state/lib-amavis/.spamassassin
# ham: inbox subdirectories
30 3 * * * root  sa-learn --ham /var/mail/otherdomain.com/*/cur* --dbpath /var/mail-state/lib-amavis/.spamassassin
```

with plain docker-compose:

```
version: "2"

services:
  mail:
    image: tvial/docker-mailserver:latest
    # ...
    volumes:
      - ./cron/sa-learn:/etc/cron.d/sa-learn
```

with docker swarm:

```
version: "3.3"

services:
  mail:
    image: tvial/docker-mailserver:latest
    # ...
    configs:
      - source: my_sa_crontab
        target: /etc/cron.d/sa-learn

configs:
  my_sa_crontab:
    file: ./cron/sa-learn
```

With the default settings, Spamassassin will require 200 mails trained for spam (for example with the method explained above) and 200 mails trained for ham (using the same command as above but using `--ham` and providing it with some ham mails). Until you provided these 200+200 mails, Spamassasin will not take the learned mails into account. For further reference, see the Spamassassin Wiki.

How can I configure a catch-all?

Considering you want to redirect all incoming e-mails for the domain `domain.tld` to `user1@domain.tld`, add the following line to `config/postfix-virtual.cf`:

```
@domain.tld user1@domain.tld
```

How can I delete all the e-mails for a specific user?

First of all, create a special alias named `devnull` by editing `config/postfix-aliases.cf`:

```
devnull: /dev/null
```

Considering you want to delete all the e-mails received for `baduser@domain.tld`, add the following line to `config/postfix-virtual.cf`:

```
baduser@domain.tld devnull
```

How do I have more control about what SPAMASSASIN is filtering?

By default, SPAM and INFECTED emails are put to a quarantine which is not very straight forward to access. Several config settings are affecting this behavior:

First, make sure you have the proper thresholds set:

```
SA_TAG=-100000.0
SA_TAG2=3.75
SA_KILL=100000.0
```

The very negative vaue in `SA_TAG` makes sure, that all emails have the Spamassasin headers included. `SA_TAG2` is the actual threshold to set the YES/NO flag for spam detection. `SA_KILL` needs to be very high, to make sure nothing is bounced at all (`SA_KILL` superseeds `SPAMASSASSIN_SPAM_TO_INBOX`)

Make sure everything (including SPAM) is delivered to the inbox and not quarantined.

```
SPAMASSASSIN_SPAM_TO_INBOX=1
```

Use `MOVE_SPAM_TO_JUNK=1` or create a sieve script which puts spam to the Junk folder.

```
require ["comparator-i;ascii-numeric","relational","fileinto"];

if header :contains "X-Spam-Flag" "YES" {
  fileinto "Junk";
} elsif allof (
   not header :matches "x-spam-score" "-*",
   header :value "ge" :comparator "i;ascii-numeric" "x-spam-score" "3.75" ) {
  fileinto "Junk";
}
```

Create a dedicated mailbox for emails which are infected/bad header and everything amavis is blocking by default and put its address into `config/amavis.cf`

```
$clean_quarantine_to      = "amavis\@domain.com";
$virus_quarantine_to      = "amavis\@domain.com";
$banned_quarantine_to     = "amavis\@domain.com";
$bad_header_quarantine_to = "amavis\@domain.com";
$spam_quarantine_to       = "amavis\@domain.com";
```

What kind of SSL certificates can I use?

You can use the same certificates you use with another mail server.
The only thing is that we provide a `self-signed` certificate tool and a `letsencrypt` certificate loader.

I just moved from my old mail server but "it doesn't work".

If this migration implies a DNS modification, be sure to wait for DNS propagation before opening an issue. Few examples of symptoms can be found here or here.
This could be related to a modification of your `MX` record, or the IP mapped to `mail.my-domain.tld`. Additionally, validate your DNS configuration.

If everything is OK regarding DNS, please provide formatted logs and config files. This will allow us to help you.

If we're blind, we won't be able to do anything.

Which system requirements needs my container to run **`docker-mailserver`** effectively?

1 core and 1GB of RAM + swap partition is recommended to run `docker-mailserver` with clamav. Otherwise, it could work with 512M of RAM.

Please note that clamav can consume a lot of memory, as it reads the entire signature database into RAM. Current figure is about 850M and growing. If you get errors about clamav or amavis failing to allocate memory you need more RAM or more swap and of course docker must be allowed to use swap (not always the case). If you can't use swap at all you may need 3G RAM.

Is **`docker-mailserver`** running in a rancher environment?

Yes, by Adding the Environment Variable `PERMIT_DOCKER: network`.

WARNING: Adding the docker network's gateway to the list of trusted hosts, e.g. using the `network` or `connected-networks` option, can create an open relay, for instance if IPv6 is enabled on the host machine but not in Docker. (#1405)

How can I authenticate users with SMTP_ONLY?

See https://github.com/tomav/docker-mailserver/issues/1247 for an example.

ToDo: Write a HowTo/UseCase/Tutorial about authentication with SMTP_ONLY.

Common errors

```
warning: connect to Milter service inet:localhost:8893: Connection refused
# DMARC not running
# => /etc/init.d/opendmarc restart

warning: connect to Milter service inet:localhost:8891: Connection refused
# DKIM not running
# => /etc/init.d/opendkim restart

mail amavis[1459]: (01459-01) (!)connect to /var/run/clamav/clamd.ctl failed, attempt #1: Can't connect to a UNIX socket /var/run/clamav/clamd.
mail amavis[1459]: (01459-01) (!)ClamAV-clamd: All attempts (1) failed connecting to /var/run/clamav/clamd.ctl, retrying (2)
mail amavis[1459]: (01459-01) (!)ClamAV-clamscan av-scanner FAILED: /usr/bin/clamscan KILLED, signal 9 (0009) at (eval 100) line 905.
mail amavis[1459]: (01459-01) (!!)AV: ALL VIRUS SCANNERS FAILED
# Clamav is not running (not started or because you don't have enough memory)
# => check requirements and/or start Clamav
```

Using behind proxy

Add to `/etc/postfix/main.cf` :

```
proxy_interfaces = X.X.X.X (your public IP)
```

What about updates

You can of course use a own script or every now and then pull && stop && rm && start the images but there are tools available for this. There is a page in the Update and cleanup wiki page that explains how to use it the docker way.

Howto adjust settings with the user-patches.sh script

Suppose you want to change a number of settings that are not listed as variables or add things to the server that are not included?

This docker-container has a built-in way to do post-install processes. If you place a script called `user-patches.sh` in the config directory it will be run after all configuration files are set up, but before the postfix, amavis and other daemons are started.

The config file I am talking about is this volume in the yml file:

`- ./config/:/tmp/docker-mailserver/`

To place such a script you can just make it in the config dir, for instance like this:

`cd ./config`

`touch user-patches.sh`

`chmod +x user-patches.sh`

and then fill it with suitable code.

If you want to test it you can move into the running container, run it and see if it does what you want. For instance:

`./setup.sh debug login # start shell in container`

`cat /tmp/docker-mailserver/user-patches.sh #check the file`

`/tmp/docker-mailserver/user-patches.sh ## run the script`

`exit`

You can do a lot of things with such a script. You can find an example user-patches.sh script here: example user-patches.sh script

Special case patching supervisord config

It seems worth noting, that the `user-patches.sh` gets executed trough supervisord. If you need to patch some supervisord config (e.g. `/etc/supervisor/conf.d/saslauth.conf`), the patching happens too late. An easy workaround is to make the `user-patches.sh` reload the supervisord config after patching it:

```
#!/bin/bash
sed -i 's/rimap -r/rimap/' /etc/supervisor/conf.d/saslauth.conf
supervisorctl update
```

Forward-Only-mailserver-with-LDAP-authentication

Building a Forward-Only mailserver

A forward-only mailserver does not have any local mailboxes. Instead, it has only aliases that forward emails to external email accounts (for example to a gmail account). You can also send email from the localhost (the computer where the mailserver is installed), using as sender any of the alias addresses.

The important settings for this setup (on `mailserver.env`) are these:

```
PERMIT_DOCKER=host
ENABLE_POP3=
ENABLE_CLAMAV=0
SMTP_ONLY=1
ENABLE_SPAMASSASSIN=0
ENABLE_FETCHMAIL=0
```

Since there are no local mailboxes, we use `SMTP_ONLY=1` to disable `dovecot`. We disable as well the other services that are related to local mailboxes (`POP3` , `ClamAV` , `SpamAssassin` , etc.)

We can create aliases with `./setup.sh` , like this:

```
./setup.sh alias add <alias-address> <external-email-account>
```

Authenticating with LDAP

If you want to send emails from outside the mailserver you have to authenticate somehow (with a username and password). One way of doing it is described in this discussion. However if there are many user accounts, it is better to use authentication with LDAP. The settings for this on `mailserver.env` are:

```
ENABLE_LDAP=1
LDAP_START_TLS=yes
LDAP_SERVER_HOST=ldap.example.org
LDAP_SEARCH_BASE=ou=users,dc=example,dc=org
LDAP_BIND_DN=cn=mailserver,dc=example,dc=org
LDAP_BIND_PW=pass1234

ENABLE_SASLAUTHD=1
SASLAUTHD_MECHANISMS=ldap
SASLAUTHD_LDAP_SERVER=ldap.example.org
SASLAUTHD_LDAP_SSL=0
SASLAUTHD_LDAP_START_TLS=yes
SASLAUTHD_LDAP_BIND_DN=cn=mailserver,dc=example,dc=org
SASLAUTHD_LDAP_PASSWORD=pass1234
SASLAUTHD_LDAP_SEARCH_BASE=ou=users,dc=example,dc=org
SASLAUTHD_LDAP_FILTER=(&(uid=%U)(objectClass=inetOrgPerson))
```

My LDAP data structure is very basic, containing only the username, password, and the external email address where to forward emails for this user. An entry looks like this

```
add uid=username,ou=users,dc=example,dc=org
uid: username
objectClass: inetOrgPerson
sn: username
cn: username
userPassword: {SSHA}abcdefghi123456789
email: real-email-address@external-domain.com
```

This structure is different from what is expected/assumed from the configuration scripts of the mailserver, so it doesn't work just by using the `LDAP_QUERY_FILTER_...` settings. Instead, I had to do custom configuration. I created the script `config/user-patches.sh` , with a content like this:

```
#!/bin/bash
```

```
rm -f /etc/postfix/{ldap-groups.cf,ldap-domains.cf}

postconf \
   "virtual_mailbox_domains = /etc/postfix/vhost" \
   "virtual_alias_maps = ldap:/etc/postfix/ldap-aliases.cf texthash:/etc/postfix/virtual" \
   "smtpd_sender_login_maps = ldap:/etc/postfix/ldap-users.cf"

sed -i /etc/postfix/ldap-users.cf \
   -e '/query_filter/d' \
   -e '/result_attribute/d' \
   -e '/result_format/d'
cat <<EOF >> /etc/postfix/ldap-users.cf
query_filter = (uid=%u)
result_attribute = uid
result_format = %s@example.org
EOF

sed -i /etc/postfix/ldap-aliases.cf \
   -e '/domain/d' \
   -e '/query_filter/d' \
   -e '/result_attribute/d'
cat <<EOF >> /etc/postfix/ldap-aliases.cf
domain = example.org
query_filter = (uid=%u)
result_attribute = mail
EOF

postfix reload
```

You see that besides `query_filter`, I had to customize as well `result_attribute` and `result_format`.

For more details about using LDAP see: LDAP managed mail server with Postfix and Dovecot for multiple domains

Another solution that serves as a forward-only mailserver is this: https://gitlab.com/docker-scripts/postfix

Full-text-search

Overview

Full-text search allows all messages to be indexed, so that mail clients can quickly and efficiently search messages by their full text content.

The dovecot-solr Plugin is used in conjunction with Apache Solr running in a separate container. This is quite straightforward to setup using the following instructions.

Setup Steps

1. docker-compose.yml:

```
  solr:
    image: lmmdock/dovecot-solr:latest
    volumes:
     - solr-dovecot:/opt/solr/server/solr/dovecot
    restart: always

  mailserver:
    image: tvial/docker-mailserver:latest
    ...
    volumes:
      ...
      - ./etc/dovecot/conf.d/10-plugin.conf:/etc/dovecot/conf.d/10-plugin.conf:ro
    ...

volumes:
  solr-dovecot:
    driver: local
```

2. `etc/dovecot/conf.d/10-plugin.conf` :

```
mail_plugins = $mail_plugins fts fts_solr

plugin {
  fts = solr
  fts_autoindex = yes
  fts_solr = url=http://solr:8983/solr/dovecot/
}
```

3. Start the solr container: `docker-compose up -d --remove-orphans solr`
4. Restart the mailserver container: `docker-compose restart mailserver`
5. Flag all user mailbox FTS indexes as invalid, so they are rescanned on demand when they are next searched

```
docker-compose exec mailserver doveadm fts rescan -A
```

Further discussion

See issue #905

Home

Welcome to the extended documentation for docker-mailserver!

Please first have a look at the `README.md` to setup and configure this server. This wiki provides you with advanced configuration, detailed examples, hints - see navigation on the right side.

To get you started

1. The script `setup.sh` is supplied with this project. It supports you in configuring and administrating your server. Information on how to get it and how to use it is available on a dedicated page.
2. Be aware that advanced tasks may still require tweaking environment variables, reading through documentation and sometimes inspecting your running container for debugging purposes. After all, a mail server is a complex arrangement of various programs.
3. A list of all configuration options is provided in `ENVIRONMENT.md`. The `README.md` is a good starting point to understand what this image is capable of.
4. A list of all optional and automatically created configuration files and directories is available on the dedicated page.
5. See the FAQ for some more tips!

IPv6

Background

If your container host supports IPv6, then `docker-mailserver` will automatically accept IPv6 connections by way of the docker host's IPv6. However, incoming mail will fail SPF checks because they will appear to come from the IPv4 gateway that docker is using to proxy the IPv6 connection (172.20.0.1 is the gateway).

This can be solved by supporting IPv6 connections all the way to the `docker-mailserver` container.

Setup steps

```
+++ b/serv/docker-compose.yml
@@ -1,4 +1,4 @@
-version: '2'
+version: '2.1'

@@ -32,6 +32,16 @@ services:

+  ipv6nat:
+    image: robbertkl/ipv6nat
+    restart: always
+    network_mode: "host"
+    cap_add:
+      - NET_ADMIN
+      - SYS_MODULE
+    volumes:
+      - /var/run/docker.sock:/var/run/docker.sock:ro
+      - /lib/modules:/lib/modules:ro

@@ -306,4 +316,13 @@ networks:

+  default:
+    driver: bridge
+    enable_ipv6: true
+    ipam:
+      driver: default
+      config:
+        - subnet: fd00:0123:4567::/48
+          gateway: fd00:0123:4567::1
```

Further discussion

See issue #1438

Installation-Examples

Building a simple mailserver

WARNING: Adding the docker network's gateway to the list of trusted hosts, e.g. using the `network` or `connected-networks` option, can create an open relay, for instance if IPv6 is enabled on the host machine but not in Docker. (#1405)

We are going to use this docker based mailserver:

- First create a directory for the mailserver and get the setup script:

```
mkdir -p /var/ds/mail.example.org
cd /var/ds/mail.example.org/

curl -o setup.sh \
    https://raw.githubusercontent.com/tomav/docker-mailserver/master/setup.sh
chmod a+x ./setup.sh
```

- Create the file `docker-compose.yml` with a content like this:

```
version: '2'

services:
  mail:
    image: tvial/docker-mailserver:latest
    hostname: mail
    domainname: example.org
    container_name: mail
    ports:
    - "25:25"
    - "587:587"
    - "465:465"
    volumes:
    - ./data/:/var/mail/
    - ./state/:/var/mail-state/
    - ./config/:/tmp/docker-mailserver/
    - /var/ds/wsproxy/letsencrypt/:/etc/letsencrypt/
    environment:
    - PERMIT_DOCKER=network
    - SSL_TYPE=letsencrypt
    - ONE_DIR=1
    - DMS_DEBUG=1
    - SPOOF_PROTECTION=0
    - REPORT_RECIPIENT=1
    - ENABLE_SPAMASSASSIN=0
    - ENABLE_CLAMAV=0
    - ENABLE_FAIL2BAN=1
    - ENABLE_POSTGREY=0
    cap_add:
    - NET_ADMIN
    - SYS_PTRACE
```

  For more details about the environment variables that can be used, and their meaning and possible values, check also these:

  - https://github.com/tomav/docker-mailserver#environment-variables
  - https://github.com/tomav/docker-mailserver/blob/master/.env.dist

  Make sure to set the proper `domainname` that you will use for the emails. We forward only SMTP ports (not POP3 and IMAP) because we are not interested in accessing the mailserver directly (from a client). We also use these settings:

  - `PERMIT_DOCKER=network` because we want to send emails from other docker containers.
  - `SSL_TYPE=letsencrypt` because we will manage SSL certificates with letsencrypt.

- We need to open these ports on the firewall: `25`, `587`, `465`

```
ufw allow 25
```

```
ufw allow 587
ufw allow 465
```

On your server you may have to do it differently.

- Pull the docker image:

```
docker pull tvial/docker-mailserver:latest
```

- Now generate the DKIM keys with `./setup.sh config dkim` and copy the content of the file `config/opendkim/keys/domain.tld/mail.txt` on the domain zone configuration at the DNS server. I use bind9 for managing my domains, so I just paste it on `example.org.db`:

```
mail._domainkey IN      TXT     ( "v=DKIM1; h=sha256; k=rsa; "
      "p=MIIBIjANBgkqhkiG9w0BAQEFACAQ8AMIIBCgKCAQEAaH5KuPYPSF3Ppkt466BDMAFGOA4mgqn4oPjZ5BbFlYA9l5jU3bgzRj3l6/Q1n5a9lQ
      "iqq3bD/BVlwKRp5gH6TEYEmx8EBJUuDxrJhkWRUk2VDl1fqhVBy8A9O7Ah+85nMrlOHIFsTaYo9o6+cDJ6t1i6G1gu+bZD0d3/3bqGLPBQV9L
```

- Add these configurations as well on the same file on the DNS server:

```
mail     IN  A   10.11.12.13

; mailservers for example.org
   3600  IN  MX  1  mail.example.org.

; Add SPF record
        IN TXT "v=spf1 mx ~all"
```

Then don't forget to change the serial number and to restart the service.

- Get an SSL certificate from letsencrypt. I use wsproxy for managing SSL letsencrypt certificates of my domains:

```
cd /var/ds/wsproxy
ds domains-add mail mail.example.org
ds get-ssl-cert myemail@gmail.com mail.example.org --test
ds get-ssl-cert myemail@gmail.com mail.example.org
```

Now the certificates will be available on `/var/ds/wsproxy/letsencrypt/live/mail.example.org`.

- Start the mailserver and check for any errors:

```
apt install docker-compose
docker-compose up mail
```

- Create email accounts and aliases with `SPOOF_PROTECTION=0`:

```
./setup.sh email add admin@example.org passwd123
./setup.sh email add info@example.org passwd123
./setup.sh alias add admin@example.org myemail@gmail.com
./setup.sh alias add info@example.org myemail@gmail.com
./setup.sh email list
./setup.sh alias list
```

Aliases make sure that any email that comes to these accounts is forwarded to my real email address, so that I don't need to use POP3/IMAP in order to get these messages. Also no anti-spam and anti-virus software is needed, making the mailserver lighter.

- Or create email accounts and aliases with `SPOOF_PROTECTION=1`:

```
./setup.sh email add admin.gmail@example.org passwd123
./setup.sh email add info.gmail@example.org passwd123
./setup.sh alias add admin@example.org admin.gmail@example.org
./setup.sh alias add info@example.org info.gmail@example.org
./setup.sh alias add admin.gmail@example.org myemail@gmail.com
```

```
./setup.sh alias add info.gmail@example.org myemail@gmail.com
./setup.sh email list
./setup.sh alias list
```

This extra step is required to avoid the `553 5.7.1 Sender address rejected: not owned by user` error (the account used for setting up gmail is `admin.gmail@example.org` and `info.gmail@example.org`)

- Send some test emails to these addresses and make other tests. Then stop the container with `Ctrl+c` and start it again as a daemon: `docker-compose up -d mail`.
- Now save on Moodle configuration the SMTP settings and test by trying to send some messages to other users:
  - SMTP hosts: `mail.example.org:465`
  - SMTP security: `SSL`
  - SMTP username: `info@example.org`
  - SMTP password: `passwd123`

Using docker-mailserver behind proxy

Information

If you are hiding your container behind a proxy service you might have discovered that the proxied requests from now on contain the proxy IP as the request origin. Whilst this behavior is technical correct it produces certain problems on the containers behind the proxy as they cannot distinguish the real origin of the requests anymore.

To solve this problem on TCP connections we can make use of the proxy protocol. Compared to other workarounds that exist (`X-Forwarded-For` which only works for HTTP requests or `Tproxy` that requires you to recompile your kernel) the proxy protocol:

- it is protocol agnostic (can work with any layer 7 protocols, even when encrypted).
- it does not require any infrastructure changes
- nat-ing firewalls have no impact it
- it is scalable The is only one condition: both endpoints of the connection MUST be compatible with proxy protocol.

Luckily `dovecot` and `postfix` are both Proxy-Protocol ready softwares so it depends only on your used reverse-proxy/loadbalancer.

Configuration of the used proxy software

The configuration depends on the used proxy system. I will provide the configuration examples of traefik v2 using IMAP and SMTP with implicit TLS. Feel free to add your configuration if you achived the same goal using different proxy software below:

▶ traefik v2

Configuration of the backend (**`dovecot`** and **`postfix`**)

The following changes can be achived completely by adding the content to the appropriate files by using the projects function to overwrite config files.

Changes for `postfix` can be applied by adding the following content to `config/postfix-main.cf` :

```
postscreen_upstream_proxy_protocol = haproxy
```

and to `config/postfix-master.cd` :

```
submission/inet/smtpd_upstream_proxy_protocol=haproxy
smtps/inet/smtpd_upstream_proxy_protocol=haproxy
```

Changes for `dovecot` can be applied by adding the following content to `config/dovecot.cf` :

```
haproxy_trusted_networks = <your-proxy-ip>, <optional-cidr-notation>
haproxy_timeout = 3 secs
service imap-login {
  inet_listener imaps {
    haproxy = yes
```

```
    ssl = yes
    port = 10993
  }
}
```

Note that port `10993` is used here to avoid conflicts with internal systems like `postscreen` and `amavis` as they will exchange messages on the default port and obviously have a different origin then compared to the proxy.

List-of-optional-config-files-&-directories

This is a list of all configuration files and directories which are optional or automatically generated in your `config` directory.

Directories:

- sieve-filter: directory for sieve filter scripts. See wiki
- sieve-pipe: directory for sieve pipe scripts. See wiki
- opendkim: DKIM directory. Autoconfigurable via setup.sh config dkim. See wiki for further info
- ssl: SSL Certificate directory. Autoconfigurable via setup.sh config ssl. Make sure to read the wiki as well to get a working mail server.

Files:

- {user_email_address}.dovecot.sieve: User specific Sieve filter file. See wiki
- before.dovecot.sieve: Global Sieve filter file, applied prior to the ${login}.dovecot.sieve filter. See wiki
- after.dovecot.sieve: Global Sieve filter file, applied after the ${login}.dovecot.sieve filter. See wiki
- postfix-main.cf: Every line will be added to the postfix main configuration. See wiki
- postfix-master.cf: Every line will be added to the postfix master configuration. See wiki
- postfix-accounts.cf: User accounts file. Modify via the setup.sh email script.
- postfix-send-access.cf: List of users denied sending. Modify via setup.sh email restrict
- postfix-receive-access.cf: List of users denied receiving. Modify via setup.sh email restrict
- postfix-virtual.cf: Alias configuration file. Modify via setup.sh alias
- postfix-sasl-password.cf: listing of relayed domains with their respective username:password. Modify via `setup.sh relay add-auth <domain> <username> [<password>]` . See wiki
- postfix-relaymap.cf: domain-specific relays and exclusions Modify via `setup.sh relay add-domain` and `setup.sh relay exclude-domain` . See wiki
- postfix-regexp.cf: Regular expression alias file. See wiki
- ldap-users.cf: Configuration for the virtual user mapping (virtual_mailbox_maps). See the start-mailserver.sh script
- ldap-groups.cf: Configuration for the virtual alias mapping (virtual_alias_maps). See the start-mailserver.sh script
- ldap-aliases.cf: Configuration for the virtual alias mapping (virtual_alias_maps). See the start-mailserver.sh script
- ldap-domains.cf: Configuration for the virtual domain mapping (virtual_mailbox_domains). See the start-mailserver.sh script
- whitelist_clients.local: Whitelisted domains, not considered by postgrey. Enter one host or domain per line.
- spamassassin-rules.cf: Antispam rules for Spamassassin. See wiki
- fail2ban-fail2ban.cf: Additional config options for fail2ban.cf. See wiki
- fail2ban-jail.cf: Additional config options for fail2ban's jail behaviour. See wiki
- amavis.cf: replaces the /etc/amavis/conf.d/50-user file
- dovecot.cf: replaces /etc/dovecot/local.conf. See wiki
- dovecot-quotas.cf: list of custom quotas per mailbox. See wiki

Override-Default-Dovecot-Configuration

Add configuration

The Dovecot default configuration can easily be extended providing a `config/dovecot.cf` file. Dovecot documentation remains the best place to find configuration options.

Your `docker-mailserver` folder should look like this example:

```
├── config
│   ├── dovecot.cf
│   ├── postfix-accounts.cf
│   └── postfix-virtual.cf
├── docker-compose.yml
└── README.md
```

One common option to change is the maximum number of connections per user:

```
mail_max_userip_connections = 100
```

Another important option is the `default_process_limit` (defaults to `100`). If high-security mode is enabled you'll need to make sure this count is higher than the maximum number of users that can be logged in simultaneously. This limit is quickly reached if users connect to the mail server with multiple end devices.

Override configuration

For major configuration changes it's best to override the `dovecot` configuration files. For each configuration file you want to override, add a list entry under the `volumes:` key.

```
version: '2'

services:
  mail:
  ...
    volumes:
      - maildata:/var/mail
      ...
      - ./config/dovecot/10-master.conf:/etc/dovecot/conf.d/10-master.conf
```

Debugging

To debug your dovecot configuration you can use this command:

```
./setup.sh debug login doveconf | grep <some-keyword>
```

setup.sh is included in the `docker-mailserver` repository.

or

```
docker exec -ti <your-container-name> doveconf | grep <some-keyword>
```

The `config/dovecot.cf` is copied to `/etc/dovecot/local.conf`. To check this file run:

```
docker exec -ti <your-container-name> cat /etc/dovecot/local.conf
```

Override-Default-Postfix-Configuration

The Postfix default configuration can easily be extended by providing a `config/postfix-main.cf` in postfix format. This can also be used to add configuration that is not in our default configuration.

For example, one common use of this file is for increasing the default maximum message size:

```
# increase maximum message size
  message_size_limit = 52428800
```

That specific example is now supported and can be handled by setting POSTFIX_MESSAGE_SIZE_LIMIT.

Postfix documentation remains the best place to find configuration options.

Each line in the provided file will be loaded into postfix.

In the same way it is possible to add a custom `config/postfix-master.cf` file that will override the standard `master.cf`. Each line in the file will be passed to `postconf -P`. The expected format is service_name/type/parameter, for example:

```
submission/inet/smtpd_reject_unlisted_recipient=no
```

Run `postconf -P` in the container without arguments to see the active master options.

Note! There should be no space between the parameter and the value.

Have a look at the code for more information.

Retrieve-emails-from-a-remote-mail-server-(using-builtin-fetchmail)

To enable the fetchmail service to retrieve e-mails set the environment variable `ENABLE_FETCHMAIL` to `1`. Your `docker-compose.yml` file should look like following snippet:

```
...
environment:
  - ENABLE_FETCHMAIL=1
  - FETCHMAIL_POLL=300
...
```

Generate a file called `fetchmail.cf` and place it in the `config` folder. Your `docker-mailserver` folder should look like this example:

```
├── config
│   ├── dovecot.cf
│   ├── fetchmail.cf
│   ├── postfix-accounts.cf
│   └── postfix-virtual.cf
├── docker-compose.yml
└── README.md
```

Configuration

A detailed description of the configuration options can be found in the online version of the manual page.

Example IMAP configuration

```
poll 'imap.example.com' proto imap
 user 'username'
 pass 'secret'
 is 'user1@domain.tld'
 ssl
```

Example POP3 configuration

```
poll 'pop3.example.com' proto pop3
 user 'username'
 pass 'secret'
 is 'user2@domain.tld'
 ssl
```

IMPORTANT: Don't forget the last line: e. g. `is 'user1@domain.tld'`. After `is` you have to specify one email address from the configuration file `config/postfix-accounts.cf`.

More details how to configure fetchmail can be found in the fetchmail man page in the chapter "The run control file".

Polling interval

By default the fetchmail service searches every 5 minutes for new mails on your external mail accounts. You can override this default value by changing the ENV variable `FETCHMAIL_POLL`.

```
  - FETCHMAIL_POLL=60
```

You must specify a numeric argument which is a polling interval in seconds. The example above polls every minute for new mails.

Debugging

To debug your `fetchmail.cf` configuration run this command:

```
./setup.sh debug fetchmail
```

For more informations about the configuration script `setup.sh` [[read the corresponding wiki page|Setup-docker-mailserver-using-the-script-setup.sh]].

Here a sample output of `./setup.sh debug fetchmail`:

```
fetchmail: 6.3.26 querying outlook.office365.com (protocol POP3) at Mon Aug 29 22:11:09 2016: poll started
Trying to connect to 132.245.48.18/995...connected.
fetchmail: Server certificate:
fetchmail: Issuer Organization: Microsoft Corporation
fetchmail: Issuer CommonName: Microsoft IT SSL SHA2
fetchmail: Subject CommonName: outlook.com
fetchmail: Subject Alternative Name: outlook.com
fetchmail: Subject Alternative Name: *.outlook.com
fetchmail: Subject Alternative Name: office365.com
fetchmail: Subject Alternative Name: *.office365.com
fetchmail: Subject Alternative Name: *.live.com
fetchmail: Subject Alternative Name: *.internal.outlook.com
fetchmail: Subject Alternative Name: *.outlook.office365.com
fetchmail: Subject Alternative Name: outlook.office.com
fetchmail: Subject Alternative Name: attachment.outlook.office.net
fetchmail: Subject Alternative Name: attachment.outlook.officeppe.net
fetchmail: Subject Alternative Name: *.office.com
fetchmail: outlook.office365.com key fingerprint: 3A:A4:58:42:56:CD:BD:11:19:5B:CF:1E:85:16:8E:4D
fetchmail: POP3< +OK The Microsoft Exchange POP3 service is ready. [SABFADEAUABSADAAMQBDAEEAMAAwADAANwAuAGUAdQByAHAAcgBkAD
fetchmail: POP3> CAPA
fetchmail: POP3< +OK
fetchmail: POP3< TOP
fetchmail: POP3< UIDL
fetchmail: POP3< SASL PLAIN
fetchmail: POP3< USER
fetchmail: POP3< .
fetchmail: POP3> USER user1@outlook.com
fetchmail: POP3< +OK
fetchmail: POP3> PASS *
fetchmail: POP3< +OK User successfully logged on.
fetchmail: POP3> STAT
fetchmail: POP3< +OK 0 0
fetchmail: No mail for user1@outlook.com at outlook.office365.com
fetchmail: POP3> QUIT
fetchmail: POP3< +OK Microsoft Exchange Server 2016 POP3 server signing off.
fetchmail: 6.3.26 querying outlook.office365.com (protocol POP3) at Mon Aug 29 22:11:11 2016: poll completed
fetchmail: normal termination, status 1
```

Understanding-the-ports

Quick Reference

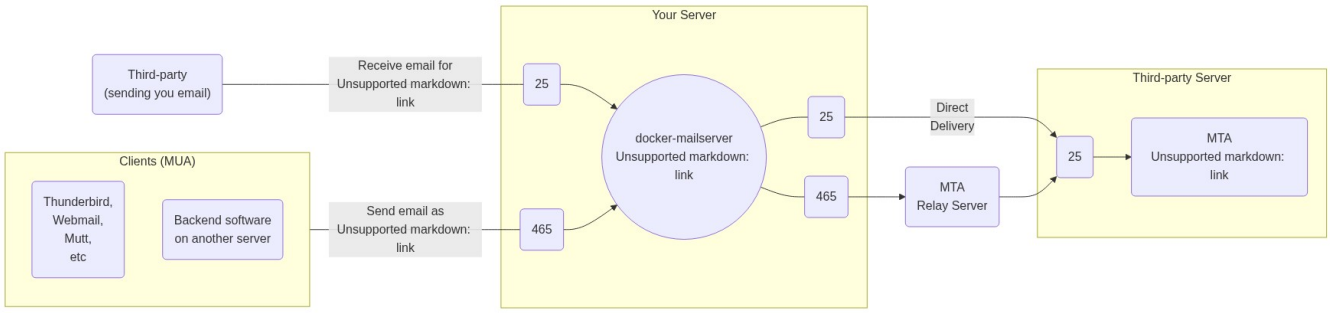
Prefer Implicit TLS ports, they're more secure and if you use a Reverse Proxy, should be less hassle (although it's probably wiser to expose these ports directly to `docker-mailserver`).

Overview of email ports

Protocol	Explicit TLS[1]	Implicit TLS	Purpose
SMTP	25	N/A	Transfer[2]
ESMTP	587	465[3]	Submission
POP3	110	995	Retrieval
IMAP4	143	993	Retrieval

1. A connection *may* be secured over TLS when both ends support `STARTTLS`. On ports 110, 143 and 587, `docker-mailserver` will reject a connection that cannot be secured. Port 25 is required to support insecure connections.
2. Receives email, `docker-mailserver` additionally filters for spam and viruses. For submitting email to the server to be sent to third-parties, you should prefer the *submission* ports(465, 587) - which require authentication. Unless a relay host is configured(eg SendGrid), outgoing email will leave the server via port 25(thus outbound traffic must not be blocked by your provider or firewall).
3. A *submission* port since 2018 (RFC 8314). Previously a secure variant of port 25.

What ports should I use? (SMTP)



▶ Flowchart - Mermaid.js source:

Inbound Traffic (On the left):

- Port 25: Think of this like a physical mailbox, it is open to receive email from anyone who wants to. `docker-mailserver` will actively filter email delivered on this port for spam or viruses and refuse mail from known bad sources. While you could also use this port internally to send email outbound without requiring authentication, you really should prefer the *Submission* ports(587, 465).
- Port 465(*and 587*): This is the equivalent of a post office box where you would send email to be delivered on your behalf(`docker-mailserver` is that metaphorical post office, aka the MTA). Unlike port 25, these two ports are known as the *Submission* ports and require a valid email account on the server with a password to be able to send email to anyone outside of the server(an MTA you do not control, eg Outlook or Gmail). Prefer port 465 which provides Implicit TLS.

Outbound Traffic (On the Right):

- Port 25: Send the email directly to the given email address MTA as possible. Like your own `docker-mailserver` port 25, this is the standard port for receiving email on, thus email will almost always arrive to the final MTA on this port. Note that, there may be additional MTAs further in the chain, but this would be the public facing one representing that email address.
- Port 465(*and 587*): SMTP Relays are a popular choice to hand-off delivery of email through. Services like SendGrid are useful for bulk email(marketing) or when your webhost or ISP are preventing you from using standard ports like port 25 to send out email(which can be abused by spammers).

  `docker-mailserver` can serve as a relay too, but the difference between a DIY relay and a professional service is reputation, which is referenced by MTAs you're delivering to such as Outlook, Gmail or others(perhaps another `docker-mailserver` server!), when deciding if email should be marked as junked or potentially not delivered at all. As a service like SendGrid has a reputation to maintain, relay is restricted to

registered users who must authenticate(even on port 25), they do not store email, merely forward it to another MTA which could be delivered on a different port like 25.

Explicit vs Implicit TLS

Explicit TLS (aka Opportunistic TLS) - Opt-in Encryption

Communication on these ports begin in cleartext, indicating support for `STARTTLS`. If both client and server support `STARTTLS` the connection will be secured over TLS, otherwise no encryption will be used.

Support for `STARTTLS` is not always implemented correctly, which can lead to leaking credentials(client sending too early) prior to a TLS connection being established. Third-parties such as some ISPs have also been known to intercept the `STARTTLS` exchange, modifying network traffic to prevent establishing a secure connection.

Due to these security concerns, RFC 8314 (Section 4.1) encourages you to prefer Implicit TLS ports where possible.

Implicit TLS - Enforced Encryption

Communication is always encrypted, avoiding the above mentioned issues with Explicit TLS.

You may know of these ports as SMTPS, POP3S, IMAPS, which indicate the protocol in combination with a TLS connection. However, Explicit TLS ports provide the same benefit when `STARTTLS` is successfully negotiated; Implicit TLS better communicates the improved security to all three protocols (SMTP/POP3/IMAP over Implicit TLS).

Additionally, referring to port 465 as *SMTPS* would be incorrect, as it is a submissions port requiring authentication to proceed via *ESMTP*, whereas ESMTPS has a different meaning(STARTTLS supported). Port 25 may lack Implicit TLS, but can be configured to be more secure between trusted parties via MTA-STS, STARTTLS Policy List, DNSSEC and DANE.

Security

TODO: *This section should provide any related configuration advice, and probably expand on and link to resources about DANE, DNSSEC, MTA-STS and STARTTLS Policy list, with advice on how to configure/setup these added security layers.*

TODO: *A related section or page on ciphers used may be useful, although less important for users to be concerned about.*

TLS connections on mail servers, compared to web browsers

Unlike with HTTP where a web browser client communicates directly with the server providing a website, a secure TLS connection as discussed below is not the equivalent safety that HTTPS provides when the transit of email (receiving or sending) is sent through third-parties, as the secure connection is only between two machines, any additional machines (MTAs) between the MUA and the MDA depends on them establishing secure connections between one another successfully.

Other machines that facilitate a connection that generally aren't taken into account can exist between a client and server, such as those where your connection passes through your ISP provider are capable of compromising a cleartext connection through interception.

Update-and-cleanup

Automatic update

Docker images are handy but it can get a a hassle to keep them updated. Also when a repository is automated you want to get these images when they get out.

One could setup a complex action/hook-based workflow using probes, but there is a nice, easy to use docker image that solves this issue and could prove useful: watchtower.

A docker-compose example:

```
services:
  watchtower:
    restart: always
    image: containrrr/watchtower:latest
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
```

For more details, see the manual

Automatic cleanup

When you are pulling new images in automatically, it would be nice to have them cleaned up as well. There is also a docker image for this: spotify/docker-gc.

A docker-compose example:

```
services:
  docker-gc:
    restart: always
    image: spotify/docker-gc:latest
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
```

For more details, see the manual

Or you can just use the `--cleanup` option provided by containrrr/watchtower.

Using-in-Kubernetes

Deployment example

There is nothing much in deploying mailserver to Kubernetes itself. The things are pretty same as in `docker-compose.yml`, but with Kubernetes syntax.

```
apiVersion: v1
kind: Namespace
metadata:
  name: mailserver
---
kind: ConfigMap
apiVersion: v1
metadata:
  name: mailserver.env.config
  namespace: mailserver
  labels:
    app: mailserver
data:
  OVERRIDE_HOSTNAME: example.com
  ENABLE_FETCHMAIL: "0"
  FETCHMAIL_POLL: "120"
  ENABLE_SPAMASSASSIN: "0"
  ENABLE_CLAMAV: "0"
  ENABLE_FAIL2BAN: "0"
  ENABLE_POSTGREY: "0"
  ONE_DIR: "1"
  DMS_DEBUG: "0"

---
kind: ConfigMap
apiVersion: v1
metadata:
  name: mailserver.config
  namespace: mailserver
  labels:
    app: mailserver
data:
  postfix-accounts.cf: |
    user1@example.com|{SHA512-CRYPT}$6$2YpW1nYtPBs2yLYS$z.5PGH1OEzsHHNhl3gJrc3D.YMZkvKw/vp.r5WIiwya6z7P/CQ9GDEJDr2G2V0cAfjD

  postfix-virtual.cf: |
    alias1@example.com user1@dexample.com

  #dovecot.cf: |
  #  service stats {
  #    unix_listener stats-reader {
  #      group = docker
  #      mode = 0666
  #    }
  #    unix_listener stats-writer {
  #      group = docker
  #      mode = 0666
  #    }
  #  }

  SigningTable: |
    *@example.com mail._domainkey.example.com

  KeyTable: |
    mail._domainkey.example.com example.com:mail:/etc/opendkim/keys/example.com-mail.key

  TrustedHosts: |
    127.0.0.1
    localhost

  #user-patches.sh: |
```

```
  #  #!/bin/bash

  #fetchmail.cf: |

---
kind: Secret
apiVersion: v1
metadata:
  name: mailserver.opendkim.keys
  namespace: mailserver
  labels:
    app: mailserver
type: Opaque
data:
  example.com-mail.key: 'base64-encoded-DKIM-key'

---
kind: Service
apiVersion: v1
metadata:
  name: mailserver
  namespace: mailserver
  labels:
    app: mailserver
spec:
  selector:
    app: mailserver
  ports:
    - name: smtp
      port: 25
      targetPort: smtp
    - name: smtp-secure
      port: 465
      targetPort: smtp-secure
    - name: smtp-auth
      port: 587
      targetPort: smtp-auth
    - name: imap
      port: 143
      targetPort: imap
    - name: imap-secure
      port: 993
      targetPort: imap-secure
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mailserver
  namespace: mailserver
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mailserver
  template:
    metadata:
      labels:
        app: mailserver
        role: mail
        tier: backend
    spec:
      #nodeSelector:
      #  kubernetes.io/hostname: local.k8s
      #initContainers:
      #- name: init-myservice
      #  image: busybox
      #  command: ["/bin/sh", "-c", "cp /tmp/user-patches.sh /tmp/files"]
      #  volumeMounts:
      #    - name: config
      #      subPath: user-patches.sh
      #      mountPath: /tmp/user-patches.sh
```

```
      #       readOnly: true
      #     - name: tmp-files
      #       mountPath: /tmp/files
      containers:
      - name: docker-mailserver
        image: tvial/docker-mailserver:latest
        imagePullPolicy: Always
        volumeMounts:
          - name: config
            subPath: postfix-accounts.cf
            mountPath: /tmp/docker-mailserver/postfix-accounts.cf
            readOnly: true
          #- name: config
          #  subPath: postfix-main.cf
          #  mountPath: /tmp/docker-mailserver/postfix-main.cf
          #  readOnly: true
          - name: config
            subPath: postfix-virtual.cf
            mountPath: /tmp/docker-mailserver/postfix-virtual.cf
            readOnly: true
          - name: config
            subPath: fetchmail.cf
            mountPath: /tmp/docker-mailserver/fetchmail.cf
            readOnly: true
          - name: config
            subPath: dovecot.cf
            mountPath: /tmp/docker-mailserver/dovecot.cf
            readOnly: true
          #- name: config
          #  subPath: user1.example.com.dovecot.sieve
          #  mountPath: /tmp/docker-mailserver/user1@example.com.dovecot.sieve
          #  readOnly: true
          #- name: tmp-files
          #  subPath: user-patches.sh
          #  mountPath: /tmp/docker-mailserver/user-patches.sh
          - name: config
            subPath: SigningTable
            mountPath: /tmp/docker-mailserver/opendkim/SigningTable
            readOnly: true
          - name: config
            subPath: KeyTable
            mountPath: /tmp/docker-mailserver/opendkim/KeyTable
            readOnly: true
          - name: config
            subPath: TrustedHosts
            mountPath: /tmp/docker-mailserver/opendkim/TrustedHosts
            readOnly: true
          - name: opendkim-keys
            mountPath: /tmp/docker-mailserver/opendkim/keys
            readOnly: true
          - name: data
            mountPath: /var/mail
            subPath: data
          - name: data
            mountPath: /var/mail-state
            subPath: state
          - name: data
            mountPath: /var/log/mail
            subPath: log
        ports:
          - name: smtp
            containerPort: 25
            protocol: TCP
          - name: smtp-secure
            containerPort: 465
            protocol: TCP
          - name: smtp-auth
            containerPort: 587
          - name: imap
            containerPort: 143
            protocol: TCP
```

```
        - name: imap-secure
          containerPort: 993
          protocol: TCP
      envFrom:
        - configMapRef:
            name: mailserver.env.config
    volumes:
      - name: config
        configMap:
          name: mailserver.config
      - name: opendkim-keys
        secret:
          secretName: mailserver.opendkim.keys
      - name: data
        persistentVolumeClaim:
          claimName: mail-storage
      - name: tmp-files
        emptyDir: {}
```

Note: Any sensitive data (keys, etc) should be deployed via Secrets. Other configuration just fits well into ConfigMaps.

Note: Make sure that Pod is assigned to specific Node in case you're using volume for data directly with `hostPath`. Otherwise Pod can be rescheduled on a different Node and previous data won't be found. Except the case when you're using some shared filesystem on your Nodes.

Exposing to outside world

The hard part with Kubernetes is to expose deployed mailserver to outside world. Kubernetes provides multiple ways for doing that. Each has its downsides and complexity.

The major problem with exposing mailserver to outside world in Kubernetes is to preserve real client IP. Real client IP is required by mailserver for performing IP-based SPF checks and spam checks.

Preserving real client IP is relatively non-trivial in Kubernetes and most exposing ways do not provide it. So, it's up to you to decide which exposing way suits better your needs in a price of complexity.

If you do not require SPF checks for incoming mails you may disable them in Postfix configuration by dropping following line (which removes `check_policy_service unix:private/policyd-spf` option):

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: mailserver.config
  labels:
    app: mailserver
data:
  postfix-main.cf: |
    smtpd_recipient_restrictions = permit_sasl_authenticated, permit_mynetworks, reject_unauth_destination, reject_unauth_pipelining, reject_inva
# ...

---

kind: Deployment
apiVersion: extensions/v1beta1
metadata:
  name: mailserver
# ...
        volumeMounts:
          - name: config
            subPath: postfix-main.cf
            mountPath: /tmp/docker-mailserver/postfix-main.cf
            readOnly: true
# ...
```

External IPs Service

The simplest way is to expose mailserver as a Service with external IPs.

```
kind: Service
apiVersion: v1
metadata:
  name: mailserver
  labels:
    app: mailserver
spec:
  selector:
    app: mailserver
  ports:
    - name: smtp
      port: 25
      targetPort: smtp
# ...
  externalIPs:
    - 80.11.12.10
```

Downsides

- Real client IP is not preserved, so SPF check of incoming mail will fail.
- Requirement to specify exposed IPs explicitly.

Proxy port to Service

The Proxy Pod helps to avoid necessity of specifying external IPs explicitly. This comes in price of complexity: you must deploy Proxy Pod on each Node you want to expose mailserver on.

Downsides

- Real client IP is not preserved, so SPF check of incoming mail will fail.

Bind to concrete Node and use host network

The simplest way to preserve real client IP is to use `hostPort` and `hostNetwork: true` in the mailserver Pod. This comes in price of availability: you can talk to mailserver from outside world only via IPs of Node where mailserver is deployed.

```
kind: Deployment
apiVersion: extensions/v1beta1
metadata:
  name: mailserver
# ...
    spec:
      hostNetwork: true
# ...
      containers:
# ...
          ports:
            - name: smtp
              containerPort: 25
              hostPort: 25
            - name: smtp-auth
              containerPort: 587
              hostPort: 587
            - name: imap-secure
              containerPort: 993
              hostPort: 993
# ...
```

Downsides

- Not possible to access mailserver via other cluster Nodes, only via the one mailserver deployed at.
- Every Port within the Container is exposed on the Host side, regardless of what the `ports` section in the Configuration defines.

Proxy port to Service via PROXY protocol

This way is ideologically the same as using Proxy Pod, but instead of a separate proxy pod, you configure your ingress to proxy TCP traffic to the

mailserver pod using the PROXY protocol, which preserves the real client IP.

Configure your ingress

With an NGINX ingress controller, set `externalTrafficPolicy: Local` for its service, and add the following to the TCP services config map (as described here):

```
# ...
  25:  "mailserver/mailserver:25::PROXY"
  465: "mailserver/mailserver:465::PROXY"
  587: "mailserver/mailserver:587::PROXY"
  993: "mailserver/mailserver:993::PROXY"
# ...
```

With HAProxy, the configuration should look similar to the above. If you know what it actually looks like, add an example here. :)

Configure the mailserver

Then, configure both Postfix and Dovecot to expect the PROXY protocol:

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: mailserver.config
  labels:
    app: mailserver
data:
  postfix-main.cf: |
    postscreen_upstream_proxy_protocol = haproxy
  postfix-master.cf: |
    submission/inet/smtpd_upstream_proxy_protocol=haproxy
    smtps/inet/smtpd_upstream_proxy_protocol=haproxy
  dovecot.cf: |
    haproxy_trusted_networks = 10.0.0.0/8, 127.0.0.0/8   # Assuming your ingress controller is bound to 10.0.0.0/8
    service imap-login {
      inet_listener imaps {
        haproxy = yes
      }
    }
# ...
---

kind: Deployment
apiVersion: extensions/v1beta1
metadata:
  name: mailserver
spec:
  template:
    spec:
      containers:
        - name: docker-mailserver
          volumeMounts:
            - name: config
              subPath: postfix-main.cf
              mountPath: /tmp/docker-mailserver/postfix-main.cf
              readOnly: true
            - name: config
              subPath: postfix-master.cf
              mountPath: /tmp/docker-mailserver/postfix-master.cf
              readOnly: true
            - name: config
              subPath: dovecot.cf
              mountPath: /tmp/docker-mailserver/dovecot.cf
              readOnly: true
# ...
```

Downsides

- Not possible to access mailserver via inner cluster Kubernetes DNS, as PROXY protocol is required for incoming connections.

Let's Encrypt certificates

Kube-Lego may be used for a role of Let's Encrypt client. It works with Kubernetes Ingress Resources and automatically issues/manages certificates/keys for exposed services via Ingresses.

```
kind: Ingress
apiVersion: extensions/v1beta1
metadata:
  name: mailserver
  labels:
    app: mailserver
  annotations:
    kubernetes.io/tls-acme: 'true'
spec:
  rules:
    - host: example.com
      http:
        paths:
          - path: /
            backend:
              serviceName: default-backend
              servicePort: 80
  tls:
    - secretName: mailserver.tls
      hosts:
        - example.com
```

Now, you can use Let's Encrypt cert and key from `mailserver.tls` Secret in your Pod spec.

```
# ...
          env:
            - name: SSL_TYPE
              value: 'manual'
            - name: SSL_CERT_PATH
              value: '/etc/ssl/mailserver/tls.crt'
            - name: SSL_KEY_PATH
              value: '/etc/ssl/mailserver/tls.key'
# ...
          volumeMounts:
            - name: tls
              mountPath: /etc/ssl/mailserver
              readOnly: true
# ...
      volumes:
        - name: tls
          secret:
            secretName: mailserver.tls
# ...
```

_Footer

_Sidebar

- Home
- Newcomers: An Introduction to Mail Servers

Configuration

- Your best friend: setup.sh
- Your users:
  - Accounts
  - Aliases
- Mail delivery: POP3
- Best practices:
  - DKIM
  - DMARC
  - SPF
  - Autodiscovery
- Security:
  - Understanding the ports
  - SSL/TLS
  - Fail2ban
- Something went wrong?
  - Debugging
  - FAQ

Advanced config / admin

- Optional config files & directories
- Maintenance:
  - Update & cleanup
- Override default config of:
  - Dovecot
  - Postfix
- LDAP authentication
- Email filtering w/ Sieve
- Email gathering w/ fetchmail
- Email forwarding with:
  - Relay Hosts
  - AWS SES
- Full-text search
- Kubernetes
- IPv6

Tutorials

- FAQ
- Installation examples

Use Cases

- Forward-Only mailserver with LDAP authentication

setup.sh

`setup.sh` is an administration script that helps with the most common tasks, including initial configuration. It is intented to be used from the host machine, *not* from within your running container.

The latest version of the script is included in the `docker-mailserver` repository. You may retrieve it at any time by running this command in your console:

```
wget https://raw.githubusercontent.com/docker-mailserver/docker-mailserver/master/setup.sh
chmod a+x ./setup.sh
```

Usage

Run `./setup.sh -h` and you'll get some usage information:

```
setup.sh Bootstrapping Script

Usage: ./setup.sh [-i IMAGE_NAME] [-c CONTAINER_NAME] <subcommand> <subcommand> [args]

OPTIONS:

 -i IMAGE_NAME     The name of the docker-mailserver image
                   The default value is
                   'docker.io/mailserver/docker-maiserver:latest'

 -c CONTAINER_NAME The name of the running container.

 -p PATH           Config folder path (default: /home/georg/github/docker-mailserver/config)

 -h                Show this help dialogue

 -z                Allow container access to the bind mount content
                   that is shared among multiple containers
                   on a SELinux-enabled host.

 -Z                Allow container access to the bind mount content
                   that is private and unshared with other containers
                   on a SELinux-enabled host.

SUBCOMMANDS:

 email:

   ./setup.sh email add <email> [<password>]
   ./setup.sh email update <email> [<password>]
   ./setup.sh email del <email>
   ./setup.sh email restrict <add|del|list> <send|receive> [<email>]
   ./setup.sh email list

 alias:
   ./setup.sh alias add <email> <recipient>
   ./setup.sh alias del <email> <recipient>
   ./setup.sh alias list

 quota:
   ./setup.sh quota set <email> [<quota>]
   ./setup.sh quota del <email>

 config:

   ./setup.sh config dkim <keysize> (default: 4096) <domain.tld> (optional - for LDAP setups)
   ./setup.sh config ssl <fqdn>

 relay:

   ./setup.sh relay add-domain <domain> <host> [<port>]
   ./setup.sh relay add-auth <domain> <username> [<password>]
```

```
  ./setup.sh relay exclude-domain <domain>

debug:

  ./setup.sh debug fetchmail
  ./setup.sh debug fail2ban [<unban> <ip-address>]
  ./setup.sh debug show-mail-logs
  ./setup.sh debug inspect
  ./setup.sh debug login <commands>

help: Show this help dialogue
```