curl -sSL https://get.docker.com/ | CHANNEL=stable sh
After the installation process is finished, you may need to enable the service and make sure it is started (e.g. CentOS 7)
systemctl enable docker.service
systemctl start docker.service

Warning

mailcow requires the latest version of docker-compose. It is highly recommended to use the commands below to install docker-compose. Package managers (e.g. apt, yum) likely won't give you the latest version. Note: This command downloads docker-compose from the official Docker Github repository and is a safe method. The snippet will determine the latest supported version by mailcow. In almost all cases this is the latest version available (exceptions are broken releases or major changes not yet supported by mailcow).

curl -L https://github.com/docker/compose/releases/download/$(curl -Ls https://www.servercow.de/docker-compose/latest.php)/docker-compose-$(uname -s)-$(uname -m) > /usr/local/bin/docker-compose
chmod +x /usr/local/bin/docker-compose

2. Clone the master branch of the repository, make sure your umask equals 0022. Please clone the repository as root user and also control the stack as root. We will modify attributes - if necessary - while boostrapping the containers automatically and make sure everything is secured. The update.sh script must therefore also be run as root. It might be necessary to change ownership and other attributes of files you will otherwise not have access to. We drop permissions for every exposed application and will not run an exposed service as root! Controlling the Docker daemon as non-root user does not give you additional security. The unprivileged user will spawn the containers as root likewise. The behaviour of the stack is identical.

$ su
# umask
0022 # <- Verify it is 0022
cd /opt
git clone https://github.com/mailcow/mailcow-dockerized
cd mailcow-dockerized

3. Generate a configuration file. Use a FQDN (host.domain.tld) as hostname when asked.

./generate_config.sh

4. Change configuration if you want or need to.

nano mailcow.conf

4.1. Users with a MTU not equal to 1500 (e.g. OpenStack):

Whenever you run into trouble and strange phenomena, please check your MTU.

Edit docker-compose.yml and change the network settings according to your MTU. Add the new driver_opts parameter like this:

```
networks:
  mailcow-network:
    ...
    driver_opts:
      com.docker.network.driver.mtu: 1450
    ...
```

4.2. Users without an IPv6 enabled network on their host system:

Enable IPv6. Finally.

If you do not have an IPv6 enabled network on your host and you don't care for a better internet (thehe), it is recommended to disable IPv6 for the mailcow network to prevent unforeseen issues.

5. Pull the images and run the compose file. The parameter -d will start mailcow: dockerized detached:

```
docker-compose pull
docker-compose up -d
```

Redirect HTTP to HTTPS (**$ cd /opt/mailcow-dockerized/**)
Since February the 28th 2017 mailcow does come with port 80 and 443 enabled.

Do not use the config below for reverse proxy setups, please see our reverse proxy guide for this, which includes a redirect from HTTP to HTTPS.

Open mailcow.conf and set HTTP_BIND= - if not already set.

Create a new file data/conf/nginx/redirect.conf and add the following server config to the file:

```
server {
  root /web;
  listen 80 default_server;
  listen [::]:80 default_server;
  include /etc/nginx/conf.d/server_name.active;
  if ( $request_uri ~* "%0A|%0D" ) { return 403; }
  location ^~ /.well-known/acme-challenge/ {
    allow all;
    default_type "text/plain";
  }
  location / {
    return 301 https://$host$uri$is_args$args;
  }
}
```

In case you changed the HTTP_BIND parameter, recreate the container:

```
docker-compose up -d
```

Otherwise restart Nginx:

```
$ cd /opt/mailcow-dockerized/
$ docker-compose restart nginx-mailcow
```

dvanced SSL

Let's Encrypt (out-of-the-box)¶

The "acme-mailcow" container will try to obtain a LE certificate for ${MAILCOW_HOSTNAME}, autodiscover.ADDED_MAIL_DOMAIN and autoconfig.ADDED_MAIL_DOMAIN.

Warning
mailcow must be available on port 80 for the acme-client to work. Our reverse proxy example configurations do cover that. You can also use any external ACME client (certbot for example) to obtain certificates, but you will need to make sure, that they are copied to the correct location and a post-hook reloads affected containers. See more in the Reverse Proxy documentation.

By default, which means 0 domains are added to mailcow, it will try to obtain a certificate for ${MAILCOW_HOSTNAME}.

For each domain you add, it will try to resolve autodiscover.ADDED_MAIL_DOMAIN and autoconfig.ADDED_MAIL_DOMAIN to its IPv6 address or - if IPv6 is not configured in your domain - IPv4 address. If it succeeds, a name will be added as SAN to the certificate request.

Only names that can be validated, will be added as SAN.

For every domain you remove, the certificate will be moved and a new certificate will be requested. It is not possible to keep domains in a certificate, when we are not able validate the challenge for those.

If you want to re-run the ACME client, use docker-compose restart acme-mailcow and monitor its logs with docker-compose logs --tail=200 -f acme-mailcow.

Additional domain names¶

Edit "mailcow.conf" and add a parameter ADDITIONAL_SAN like this:

Do not use quotes (") and do not use spaces between the names!

ADDITIONAL_SAN=smtp.*,cert1.example.com,cert2.example.org,whatever.*

Each name will be validated against its IPv6 address or - if IPv6 is not configured in your domain - IPv4 address.

A wildcard name like smtp.* will try to obtain a smtp.DOMAIN_NAME SAN for each domain added to mailcow.

Run docker-compose up -d to recreate affected containers automatically.

Info
Using names other name MAILCOW_HOSTNAME to access the mailcow UI may need further configuration.

If you plan to use a server name that is not MAILCOW_HOSTNAME to access the mailcow UI (for example by adding mail.* to ADDITIONAL_SAN make sure to populate that name in

mailcow.conf via ADDITIONAL_SERVER_NAMES. Names must be separated by commas and must not contain spaces. If you skip this step, mailcow may respond with an incorrect site.

ADDITIONAL_SERVER_NAMES=webmail.domain.tld,other.example.tld

Run docker-compose up -d to apply.

Force renewal¶

To force a renewal, you need to create a file named force_renew and restart the acme-mailcow container:

cd /opt/mailcow-dockerized

touch data/assets/ssl/force_renew

docker-compose restart acme-mailcow

Now check the logs for a renewal

docker-compose logs --tail=200 -f acme-mailcow

The file will be deleted automatically.

Validation errors and how to skip validation¶

You can skip the IP verification by setting SKIP_IP_CHECK=y in mailcow.conf (no quotes). Be warned that a misconfiguration will get you ratelimited by Let's Encrypt! This is primarily useful for multi-IP setups where the IP check would return the incorrect source IP address. Due to using dynamic IPs for acme-mailcow, source NAT is not consistent over restarts.

If you encounter problems with "HTTP validation", but your IP address confirmation succeeds, you are most likely using firewalld, ufw or any other firewall, that disallows connections from br-mailcow to your external interface. Both firewalld and ufw disallow this by default. It is often not enough to just stop these firewall services. You'd need to stop mailcow (docker-compose down), stop the firewall service, flush the chains and restart Docker.

You can also skip this validation method by setting SKIP_HTTP_VERIFICATION=y in "mailcow.conf". Be warned that this is discouraged. In most cases, the HTTP verification is skipped to workaround unknown NAT reflection issues, which are not resolved by ignoring this specific network misconfiguration. If you encounter problems generating TLSA records in the DNS overview within mailcow, you are most likely having issues with NAT reflection you should fix.

If you changed a SKIP_* parameter, run docker-compose up -d to apply your changes.

Disable Let's Encrypt¶

Disable Let's Encrypt completely¶

Set SKIP_LETS_ENCRYPT=y in "mailcow.conf" and recreate "acme-mailcow" by running docker-compose up -d.

Skip all names but ${MAILCOW_HOSTNAME}¶

Add ONLY_MAILCOW_HOSTNAME=y to "mailcow.conf" and recreate "acme-mailcow" by running docker-compose up -d.

The Let's Encrypt subjectAltName limit of 100 domains¶

Let's Encrypt currently has a limit of 100 Domain Names per Certificate.

By default, "acme-mailcow" will create a single SAN certificate for all validated domains (see the first section and Additional domain names). This provides best compatibility but means the Let's Encrypt limit exceeds if you add too many domains to a single mailcow installation.

To solve this, you can configure ENABLE_SSL_SNI to generate:

- A main server certificate with MAILCOW_HOSTNAME and all fully qualified domain names in the ADDITIONAL_SAN config
- One additional certificate for each domain found in the database with autodiscover., autoconfig. and any other ADDITIONAL_SAN configured in this format (subdomain.*).
- Limitations: A certificate name ADDITIONAL_SAN=test.example.com will be added as SAN to the main certificate. A separate certificate/key pair will not be generated for this format.

Postfix, Dovecot and Nginx will then serve these certificates with SNI.

Set ENABLE_SSL_SNI=y in "mailcow.conf" and recreate "acme-mailcow" by running docker-compose up -d.

> **Warning**
> Not all clients support SNI, see Dovecot documentation or Wikipedia. You should make sure these clients use the MAILCOW_HOSTNAME for secure connections if you enable this feature.

Here is an example:

- MAILCOW_HOSTNAME=server.email.tld
- ADDITIONAL_SAN=webmail.email.tld,mail.*
- Mailcow email domains: "domain1.tld" and "domain2.tld"

The following certificates will be generated:

- server.email.tld, webmail.email.tld -> this is the default certificate, all clients can connect with these domains
- mail.domain1.tld, autoconfig.domain1.tld, autodiscover.domain1.tld -> individual certificate for domain1.tld, cannot be used by clients without SNI support
- mail.domain2.tld, autoconfig.domain2.tld, autodiscover.domain2.tld -> individual certificate for domain2.tld, cannot be used by clients without SNI support

How to use your own certificate¶

Make sure you disable mailcows internal LE client (see above).

To use your own certificates, just save the combined certificate (containing the certificate and intermediate CA/CA if any) to data/assets/ssl/cert.pem and the corresponding key to data/assets/ssl/key.pem.

IMPORTANT: Do not use symbolic links! Make sure you copy the certificates and do not link them to data/assets/ssl.

Restart affected services afterwards:

```
docker restart $(docker ps -qaf name=postfix-mailcow)
docker restart $(docker ps -qaf name=nginx-mailcow)
docker restart $(docker ps -qaf name=dovecot-mailcow)
```

See Post-hook script for non-mailcow ACME clients for a full example script.

Test against staging ACME directory¶

Edit mailcow.conf and add LE_STAGING=y.

Run docker-compose up -d to activate your changes.

Custom directory URL¶

Edit mailcow.conf and add the corresponding directory URL to the new variable DIRECTORY_URL:

```
DIRECTORY_URL=https://acme-custom-v9000.api.letsencrypt.org/directory
```

You cannot use LE_STAGING with DIRECTORY_URL. If both are set, only LE_STAGING is used.

Run docker-compose up -d to activate your changes.

Check your configuration¶

Run docker-compose logs acme-mailcow to find out why a validation fails.

To check if nginx serves the correct certificate, simply use a browser of your choice and check the displayed certificate.

To check the certificate served by Postfix, Dovecot and Nginx we will use openssl:

```
# Connect via SMTP (587)
echo "Q" | openssl s_client -starttls smtp -crlf -connect mx.mailcow.email:587
# Connect via IMAP (143)
echo "Q" | openssl s_client -starttls imap -showcerts -connect mx.mailcow.email:143
# Connect via HTTPS (443)
echo "Q" | openssl s_client -connect mx.mailcow.email:443
```

To validate the expiry dates as returned by openssl against MAILCOW_HOSTNAME, you are able to use our helper script:

```
cd /opt/mailcow-dockerized
bash helper-scripts/expiry-dates.sh
```

https://www.mail-tester.com/

https://pi-hole.net/
Установка: curl -sSL https://install.pi-hole.net | bash

Heimdall. Крутой Dashboard (панель приложений)
Heimdall - это настраиваемая панель приложений с очень классным дизайном и простой настройкой.

Heimdall: https://heimdall.site/

Установка и настройка Контейнера Heimdall.

Команды в виде:
apt update

DOCKER Insatall:
https://docs.docker.com/engine/instal...
docker --version
docker run hello-world

Compose Install:
https://docs.docker.com/compose/install/
docker-compose --version

Пользователь и папки:

useradd -s /bin/false heimdall
id heimdall

cd /opt/
mkdir -p heimdall/config
chown -R heimdall.heimdall heimdall

mkdir -p composeFiles/heimdall
cd composeFiles/heimdall

vi docker-compose.yml

Временные зоны: https://en.wikipedia.org/wiki/List_of...

```
version: "3.3"
services:
  heimdall:
    image: linuxserver/heimdall
    container_name: heimdall
    environment:
      - PUID=####
      - PGID=####
      - TZ=Europe/Berlin
    volumes:
      - /opt/heimdall/config:/config
    ports:
      - 9001:80
      - 9002:443
    restart: always
```