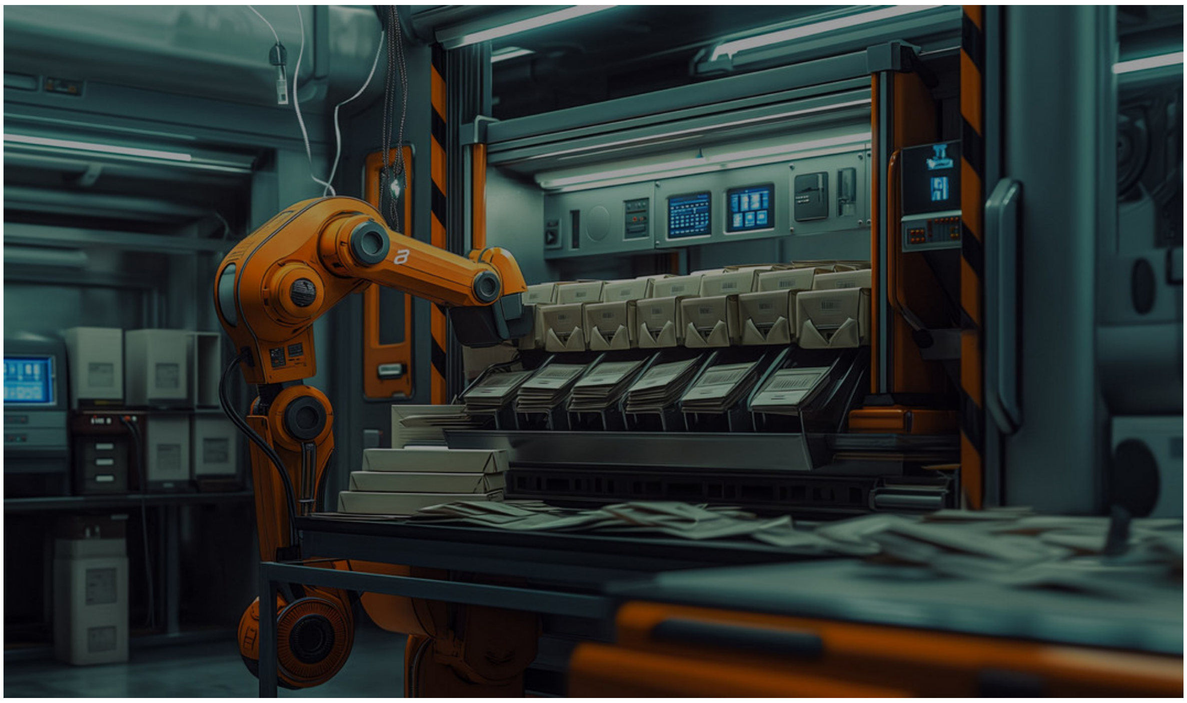


Postal User Manual English Version

Author：网管小贾 / sysadm.cc (8/4/2024)

Welcome

Feature List

This is a list of features (in no real particular order) of things that Postal can do.

General features:

- Support for multiple organizations with mail servers and users within.
- Graphs and stats showing volume of incoming and outgoing mail.
- Access to view historical messages.
- Access to view the full outgoing and incoming message queue.
- Set up webhooks to receive live information about delivery information in realtime. Full access to the last 7 days of webhook requests are also stored for debugging purposes.
- Built-in DNS checking and monitoring to ensure domains you send mail from are configured correctly for maximum deliverability.
- Per server retention configuration to set how long messages should be kept in the database and the maximum size to keep on disk.
- Complete logging so delivery issues can easily be identified.
- Mail server wide search tools to find messages that need investigation.

Outgoing e-mails:

- Send messages to the SMTP server or using the HTTP API.
- Manage multiple credentials per server.
- Support for DKIM signing of outbound messages.
- Enable development to hold messages in Postal without actually delivering them to recipients (message can be viewed in the Postal interface).
- Built-in suppression list to avoid sending mail to recipients that don't exist or can't accept e-mail.
- Click and open tracking to keep track of when recipients open your e-mails and click links within them.
- Configure per-server send limits to avoid abuse on mail servers.
- Management of multiple pools of sending IP addresses.
- Configure different senders or recipients to have mail delivered from certain IP addresses.
- Mail tagging so certain e-mails can be given a tag to allow them to be grouped when needed. For example, you may tag `receipts` or `password-reset` e-mails as such.

Incoming e-mails:

- Ability to forward incoming e-mail to HTTP endpoints.
- Ability to forward incoming e-mail to other SMTP servers.
- Ability to forward incoming e-mail to other e-mail addresses.
- Spam & thread checking with SpamAssassin and ClamAV with configurable thresholds and different methods for dealing with spam messages.

This is a list of features (in no real particular order) of things that Postal can do.

FAQs

This hasn't been around long enough for anyone to have asked any question frequently enough for it to be considered an FAQ. We will update this page as soon as we have some questions to answer.

Should I use this instead of cloud provider?

That's really up to you. There are advantages and disadvantages of both solutions and you should pick the one that suits each individual situation. Don't take running your own e-mail platform lightly though, there are many considerations that need to be taken into account to ensure you achieve good deliverability (including correct DNS configuration).

E-Mails sent through Postal are going to spam.

- Check you've configured your DNS correctly. To start you need reverse DNS for your IPs, you need to configure DKIM & SPF, you need to make sure your rDNS matches the HELO given to the recipient's mail server.
- Ensure your sending IPs have reverse DNS (PTR) records configured.
- Check that the IP address you're sending mail from isn't on any blacklists.
- Check that your actual e-mail doesn't look like spam.
- New IPs sending large volumes of e-mail will likely not deliver well initially.

You can run your message through something like Mail Tester which will give you a good idea of the spammy-ness of your messages and ensure you have everything configured correctly.

Can you add mailing list feature?

No. Postal is a mail transport agent and not a mailing list manager. We don't want to add features that are better suited in another application, for example, address books or handling the un-subscription of people from a database.

Getting Started

Getting Started

Postal is fairly easy to install but running your own mail server isn't something for everyone. If you use Postal, you will be responsible for configuring your DNS as well as maintaining the platform (including running upgrades). If this doesn't sound like something you're up for, try a hosted platform.

The process for installing Postal is outlined below. When you're ready to get started go to the Pre-Requisites page.

1. Install Docker and other pre-requisites
2. Add appropriate configuration and DNS entries
3. Start Postal
4. Make your first user
5. Login to the web interface to create your first virtual mail server

A visual learner?

You can watch this video which walks through the installation process.

Pre-requisites

There are some things you'll need to do before you can install Postal.

Servers

We **strongly** recommend installing Postal on its own dedicated server (i.e. a server running no other software). The minimum specification for Postal is as follows:

- At least 4GB of RAM
- At least 2 CPU cores
- An appropriate amount of disk space (at least 25GB) for your use case

Most people install Postal on virtual servers. There are lots of providers to choose from including Digital Ocean and Linode.

One thing to be aware of is you'll need to ensure that your provider does not block port 25 outbound. This is quite common and is used to prevent abuse from spammers.

It doesn't matter what operating system you choose as long as you are able to install Docker on it (see next section). Nothing in these instructions will make assumptions about your operating system.

Docker

Postal runs entirely using containers which means to run Postal you'll need some software to run these containers. We recommend using Docker for this purpose but you can use whatever software you wish.

You'll need to install Docker Engine on your server to begin with. Follow the instructions on the Docker website to install Docker.

You'll also need to ensure you have the Docker Compose plugin installed.

Before continuing ensure that you can run both `docker` and `docker compose` from your prompt.

System utilities

There are a few system utilities that you need to have installed before you'll be able to run some of the Postal commands.

On Ubuntu/Debian:

```
apt install git curl jq
```

On CentOS/RHEL:

```
yum install git curl jq
```

Git & installation helper repository

You'll need to make sure you have `git` installed on your server. You'll then need to clone the Postal installation helper repository. This contains some bootstrapping config and other useful things which will speed along your installation.

```
git clone https://github.com/postalserver/install /opt/postal/install
sudo ln -s /opt/postal/install/bin/postal /usr/bin/postal
```

MariaDB (10.6 or higher)

Postal requires a database engine to store all email and other essential configuration data. You will need to provide credentials that allow full access to create and delete databases as well as having full access to any databases created. Postal will provision a database automatically for each mail server that you create.

We do not support using MySQL in place of MariaDB.

You can run MariaDB in a container, assuming you have Docker, using this command:

```
docker run -d \
   --name postal-mariadb \
   -p 127.0.0.1:3306:3306 \
   --restart always \
   -e MARIADB_DATABASE=postal \
   -e MARIADB_ROOT_PASSWORD=postal \
   mariadb
```

- This will run a MariaDB instance and have it listen on port 3306.
- Be sure to choose a secure password. You'll need to put this in your Postal configuration when you install it so be sure to make a (secure) note of it.
- If you are unable or unwilling to grant root access, the database user you create separately needs all permissions on databases called `postal` and `postal-*` (this prefix can be configured in the `message_db` part of your configuration).

```
Whilst you can configure the maximum message size however you wish, you will
need to verify that MariaDB is configured with `innodb_log_file_size` to at
least 10 times the biggest message you wish to send (150MB for 15MB email, 250MB
for 25MB email, etc).

If you have a legacy v1 database you might also want to check that raw tables in
the database have the type `LONGBLOB`.
```

Installation

Once you've completed all the prerequisites, you can go ahead and start to install Postal.

Configuration

Before you can start Postal, you'll need some configuration. The repository you cloned includes a tool to automatically generate some initial configuration files.

Run the command below and replace `postal.yourdomain.com` with the actual hostname you want to access your Postal web interface at. Make sure you have set up this domain with your DNS provider before continuing.

```
postal bootstrap postal.yourdomain.com
```

This will generate three files in `/opt/postal/config`.

- `postal.yml` is the main postal configuration file
- `signing.key` is the private key used to sign various things in Postal
- `Caddyfile` is the configuration for the Caddy webserver

Once generated, you should open up `/opt/postal/config/postal.yml` and add all the appropriate values for your installation (database passwords etc...).

Note that the docker setup mounts `/opt/postal/config` as `/config` so any full directory paths mentioned in `postal.yml` will likely need to start with `/config` and not `/opt/postal/config`

Initializing the database

Once you've added your configuration, you need to initialize your database by adding all the appropriate tables. Run the following commands to create the schema and then create your first admin user.

```
postal initialize
postal make-user
```

Running postal

You're now ready to actually run Postal itself. You can go ahead and do this by running:

```
postal start
```

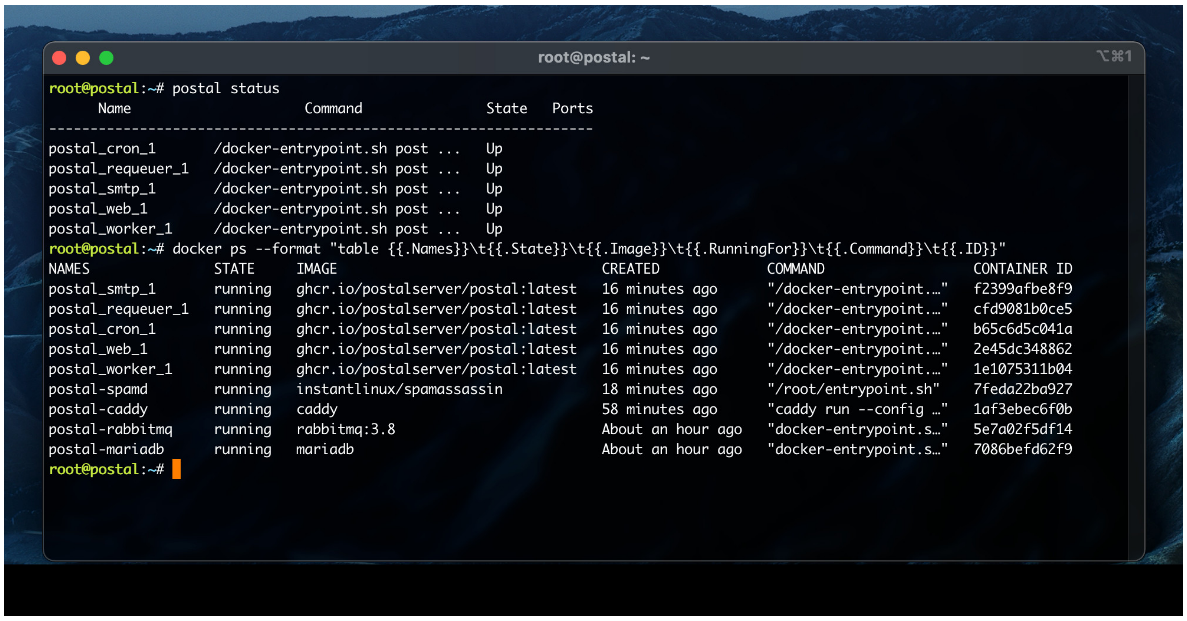
This will run a number of containers on your machine. You can use `postal status` to see details of these components.

Caddy

To handle SSL termination and all web traffic, you'll need to configure a web proxy. You can use anything that takes your fancy here - nginx, Apache, HAProxy, anything - but in this example, we're going to use Caddy. It's a great little server that requires very little configuration and is very easy to set up.

```
docker run -d \
   --name postal-caddy \
   --restart always \
   --network host \
   -v /opt/postal/config/Caddyfile:/etc/caddy/Caddyfile \
   -v /opt/postal/caddy-data:/data \
   caddy
```

Once this has started, Caddy will issue an SSL certificate for your domain and you'll be able to immediately access the Postal web interface and login with the user you created in one of the previous steps.



```
root@postal:~# postal status
      Name                     Command               State   Ports
-------------------------------------------------------------------
postal_cron_1       /docker-entrypoint.sh post ...   Up
postal_requeuer_1   /docker-entrypoint.sh post ...   Up
postal_smtp_1       /docker-entrypoint.sh post ...   Up
postal_web_1        /docker-entrypoint.sh post ...   Up
postal_worker_1     /docker-entrypoint.sh post ...   Up
root@postal:~# docker ps --format "table {{.Names}}\t{{.State}}\t{{.Image}}\t{{.RunningFor}}\t{{.Command}}\t{{.ID}}"
NAMES               STATE     IMAGE                                CREATED             COMMAND                    CONTAINER ID
postal_smtp_1       running   ghcr.io/postalserver/postal:latest   16 minutes ago      "/docker-entrypoint.…"   f2399afbe8f9
postal_requeuer_1   running   ghcr.io/postalserver/postal:latest   16 minutes ago      "/docker-entrypoint.…"   cfd9081b0ce5
postal_cron_1       running   ghcr.io/postalserver/postal:latest   16 minutes ago      "/docker-entrypoint.…"   b65c6d5c041a
postal_web_1        running   ghcr.io/postalserver/postal:latest   16 minutes ago      "/docker-entrypoint.…"   2e45dc348862
postal_worker_1     running   ghcr.io/postalserver/postal:latest   16 minutes ago      "/docker-entrypoint.…"   1e1075311b04
postal-spamd        running   instantlinux/spamassassin            18 minutes ago      "/root/entrypoint.sh"      7feda22ba927
postal-caddy        running   caddy                                58 minutes ago      "caddy run --config …"   1af3ebec6f0b
postal-rabbitmq     running   rabbitmq:3.8                         About an hour ago   "docker-entrypoint.s…"   5e7a02f5df14
postal-mariadb      running   mariadb                              About an hour ago   "docker-entrypoint.s…"   7086befd62f9
root@postal:~#
```

Upgrading

If you're not currently running Postal v2, you'll need to follow the Upgrading from 1.x documentation before you can use these instructions.

Once you have installed Postal, you can upgrade it by running this command. This will always upgrade you to the latest version of Postal that is available.

```
cd /opt/postal/install
git pull origin
postal upgrade
```

This will do a few things in the following order:

- Fetch the latest copy of the installation helpers repository using Git.
- Pull the latest version of the Postal containers.
- Perform any necessary updates to the database schema.
- Restart all running containers

This is not a zero downtime upgrade so it is recommended to do this at a time when traffic will be low and you have scheduled the maintenance as appropriate. If you need zero downtime upgrades, you will need to look for alternative container orchestration systems that can handle this (such as Kubernetes).

Changing to a specific version

By default, running `postal upgrade` will install the latest version available from the Postal container registry. If you need to change the version of Postal to a specific version, you can specify a version for the `postal upgrade` command as follows:

```
postal upgrade [version]
```

Configuration

Postal can be configured through its configuration file or environment variables. There are a fair number of areas which can be configured.

You can review all the available configuration options.

- Full Postal Configuration file - this is an example configuration file that contains all the configuration options along with their defaults and a description. This file would usually exist in `/opt/postal/config/postal.yml`.
- All environment variables - this page lists all the environment variables. All configuration that can be set in the config file can also be set by an environment variable.

Note: If you change any configuration, you should be sure to restart Postal

Ports and bind addresses

The web & SMTP server listen on ports and addresses. The defaults for these can be set through configuration however, if you're running multiple instances of these on a single host you will need to specify different ports for each one.

You can use the `PORT` and `BIND_ADDRESS` environment variables to provide instance-specific values for these processes.

Legacy configuration

The current version for the Postal configuration file is `2`. This is shown by the `version: 2` in the configuration file itself.

Postal still supports the version 1 (or legacy) configuration format from Postal v2 and earlier. If you are using this config file, you will receive a warning in the logs when starting Postal. We recommend changing your configuration to follow the new v2 format which is documented above.

The key differences between v1 and v2 configuration is shown below.

- `web.host` changes to `postal.web_hostname`
- `web.protocol` changes to `postal.web_protocol`
- `web_server.port` changes to `web_server.default_port`
- `web_server.bind_address` changes to `web_server.default_bind_address`
- `smtp_server.port` changes to `smtp_server.default_port`
- `smtp_server.bind_address` changes to `smtp_server.default_bind_address`
- `dns.return_path` changes to `dns.return_path_domain`
- `dns.smtp_server_hostname` changes to `postal.smtp_hostname`
- `general.use_ip_pools` changes to `postal.use_ip_pools`
- `general.*` changes to various new names under the `postal.` namespace
- `smtp_relays` changes to `postal.smtp_relays` and now uses an array of strings which should be in the format of `smtp://{host}:{port}?ssl_mode={mode}`

- `logging.graylog.*` changes to `gelf.*`

DNS configuration

To work properly, you'll need to configure a number of DNS records for your Postal installation. Review the table below and create appropriate DNS records with your DNS provider. You will need to enter the record names you choose in your `postal.yml` configuration file.

We'll assume for the purposes of this documentation you have both IPv4 and IPv6 available to your server. We'll use the following values in this documentation, you'll need to replace them as appropriate.

- `192.168.1.3` - IPv4 address
- `2a00:1234:abcd:1::3` - IPv6 address
- `postal.example.com` - the hostname you wish to use to run Postal

A Records

You'll need these records for accessing the API, management interface & SMTP server.

Hostname	Type	Value
postal.example.com	A	`192.168.1.3`
postal.example.com	AAAA	`2a00:1234:abcd:1::3`

SPF Record

You can configure a global SPF record for your mail server which means domains don't need to each individually reference your server IPs. This allows you to make changes in the future.

Hostname	Type	Value
spf.postal.example.com	TXT	`v=spf1 ip4:192.168.1.3 ip6:2a00:1234:abcd:1::3 ~all`

You may wish to replace `~all` with `-all` to make the SPF record stricter.

Return Path

The return path domain is the default domain that is used as the `MAIL FROM` for all messages sent through a mail server. You should add DNS records as below.

Hostname	Type	Value
rp.postal.example.com	MX	`10 postal.example.com`
rp.postal.example.com	TXT	`v=spf1 a mx include:spf.postal.example.com ~all`
postal._domainkey.rp.postal.example.com	TXT	Value from `postal default-dkim-record`

Route domain

If you wish to receive incoming e-mail by forwarding messages directly to routes in Postal, you'll need to configure a domain for this just to point to your server using an MX record.

Hostname	Type	Value
routes.postal.example.com	MX	`10 postal.example.com`

Example Postal Configuration

In your `postal.yml` you should have something that looks like the below to cover the key DNS records.

```
dns:
  mx_records:
    - mx1.postal.example.com
    - mx2.postal.example.com
  spf_include: spf.postal.example.com
  return_path_domain: rp.postal.example.com
  route_domain: routes.postal.example.com
  track_domain: track.postal.example.com
```

Upgrading to v3

If you are currently running a version of Postal less than 2.0.0, you should upgrade to v2 before v3.

Postal v3 was released in March 2024 and introduced some changes to way that Postal runs. The noteable changes between v2 and v3 are as follows:

- No need to use RabbitMQ.
- No need to run `cron` or `requeuer` processes.
- Improved logging.
- Improve configuration management (including the ability to configure with environment variables or a config file).

Database considerations

It is important that any pre-existing tables in your database are set up with the `DYNAMIC` row format. If not, you may receive errors during the database migrations. This has been the default since MariaDB 10.2.1.

You can check the format of your tables using `SHOW TABLE STATUS FROM postal`. If you have tables which are incorrect, you can change them with the following:

```
ALTER TABLE `table_name` ROW_FORMAT=DYNAMIC;
```

Upgrading

To upgrade, you can follow the same instructions as provided on the upgrade page

Configuration

Postal v3 introduces a new format for its configuration file. An example of the full, new, configuration file format can be found in our repository.

While v3 is still compatible with configuration from earlier versions but you should change your configuration to the new format to ensure continued compatibility. Any newly added configuration options are not available in the v1 configuration format.

RabbitMQ

Once you have upgraded to v3, you can remove any RabbitMQ services you have that solely support your Postal installation.

Cron & Requeuer Processes

These processes are not required in Postal v3 and should not be running.

Upgrading to v2

In July 2021, we changed the way that Postal is installed. The only supported method for installing Postal is now using a container that we provide. You can follow these instructions to upgrade your 1.x installation to use containers.

How do I know if I'm using Postal v1?

There are a few changes between the two versions which should help identify your version.

- The Postal web interface now has a footer on all pages (except the login page) which show the current version. If you have no footer, you're not using Postal v2.
- If you installed Postal without using containers, you are most likely using Postal v1.
- If you run `ps aux | grep procodile` and get any results, you are using Postal v1.
- If you run `docker ps` and get no results, you are using Postal v1.
- If you installed Postal before July 2021, you are using Postal v1.
- If you have an `/opt/postal/app` directory you are using Postal v1 (or you have already upgraded to Postal v2 but not tidied up).

Assumptions

For the purposes of this guide, we're going to make some assumptions about your installation. If any of these assumptions are not true, you will need to determine the appropriate route for you to upgrade.

- You have Postal installed on a single server.
- Your server has a MariaDB (or MySQL) database server running on it and listening port 3306.
- Your server has a RabbitMQ server running on it and listening on port 5672.
- Your current installation is located at `/opt/postal` and your configuration is in `/opt/postal/config`.
- You use a web proxy (such as nginx, Caddy or Apache) in front of the Postal web server.

Performing this upgrade will mean that your Postal services will be unavailable for a short period of time. We recommend scheduling some maintenance and performing the upgrade when traffic is low.

Preparation

There are a few extra system dependencies that you need to install.

- Docker
- docker-compose

Important: use the latest versions of these rather than simply just installing the latest package available from your operating system vendor's repositories. Instructions are linked above.

If you're running an old or unsupported version of your operating system, you may wish to use this as an opportunity to upgrade. The method for doing so is outside of the scope of this documentation.

Stopping Postal

Start by stopping the Postal processes using `postal stop`.

Configuring web proxy for open/click tracking

In Postal 2.x onwards, we no longer provide a dedicated server process for serving requests for open & click tracking. If you don't use this, you can skip to the next section. However, if you do, you need to add some configuration to your web proxy and issue some SSL certificates.

For all the **Tracking Domains** that you have configured (for example `track.yourdomain.com`) you will need to do the following:

1. Configure a virtual host in your web proxy to route all requests for each tracking domain to the Postal web server (on port 5000).
2. Ensure that all requests going through the proxy have the `X-Postal-Track-Host: 1` header.
3. Issue an SSL certificate for all these hosts.
4. Ensure that your web proxy is listening on the IP address that you previously used for the Postal `fast_server`.
5. As there is no longer a requirement for Postal to have two IP addresses, you can update all your DNS records that reference your secondary IP to point to the main IP that you use for Postal.

Checking configuration

Your existing configuration for Postal can remain in the same place as it was before at `/opt/postal/config`. If you have referenced any other files in your `postal.yml`, you will need to ensure that these files are within the `/opt/postal/config` folder and you replace the path with `/config`. For example, if you have this:

```yaml
smtp_server:
  tls_enabled: true
  tls_certificate_path: /opt/postal/config/smtp.crt
  tls_private_key_path: /opt/postal/config/smtp.key
```

You will need to update `/opt/postal/config` to `/config` as follows:

```yaml
smtp_server:
  tls_enabled: true
  tls_certificate_path: /config/smtp.crt
  tls_private_key_path: /config/smtp.key
```

Important: if you have referenced files in other parts of your operating system (such as in `/etc`), you must ensure these are now within the `/opt/postal/config` directory otherwise they won't be available within the container that Postal runs within.

Removing the old Postal helper script

Run the following command to backup the old Postal helper script.

```
mv /usr/bin/postal /usr/bin/postal.v1
```

Installing Postal v2

The next thing to do is to download the new Postal installation helpers repo and set up the new `postal` command.

```
git clone https://github.com/postalserver/install /opt/postal/install
sudo ln -s /opt/postal/install/bin/postal /usr/bin/postal
```

Next, run a normal upgrade using the new Postal command line helper. This will run a new image to upgrade your database schema to that required for Postal v2.

```
postal upgrade
```

Finally, you can start the Postal components.

```
postal start
```

You should now find that Postal is running again and working as normal. Confirm that all process types are running using `postal status`. Your output should look like this:

```
      Name                      Command               State   Ports
-------------------------------------------------------------------
postal_cron_1       /docker-entrypoint.sh post ...   Up
postal_requeuer_1   /docker-entrypoint.sh post ...   Up
postal_smtp_1       /docker-entrypoint.sh post ...   Up
postal_web_1        /docker-entrypoint.sh post ...   Up
postal_worker_1     /docker-entrypoint.sh post ...   Up
```

A note about SMTP ports

If you were previously running the Postal SMTP server on any port other than 25, you can revert this configuration and have Postal listen on this port directly. To do this, you can remove any `iptables` rules you might have and update your `postal.yml` with the new port number.

Rolling back

If something goes wrong and you need to rollback to the previous version you can still do that by reverting the `postal` helper and starting it again.

```
postal stop
unlink /usr/bin/postal
mv /usr/bin/postal.v1 /usr/bin/postal
postal start
```

Tidying up

When you're happy that everything is running nicely, there are some final things you should do:

- Remove `/opt/postal/app`. This is where the application itself lived and is no longer required.
- Remove `/opt/postal/log`. Logs are no longer stored here.
- Remove `/opt/postal/vendor`. This is no longer used.

- Remove the backup Postal helper tool from `/usr/bin/postal.v1`.
- If you changed any tracking domains to use your main IP address, you can remove the additional IP from the server after checking that all DNS updates have applied.

Installing on a new server with existing data

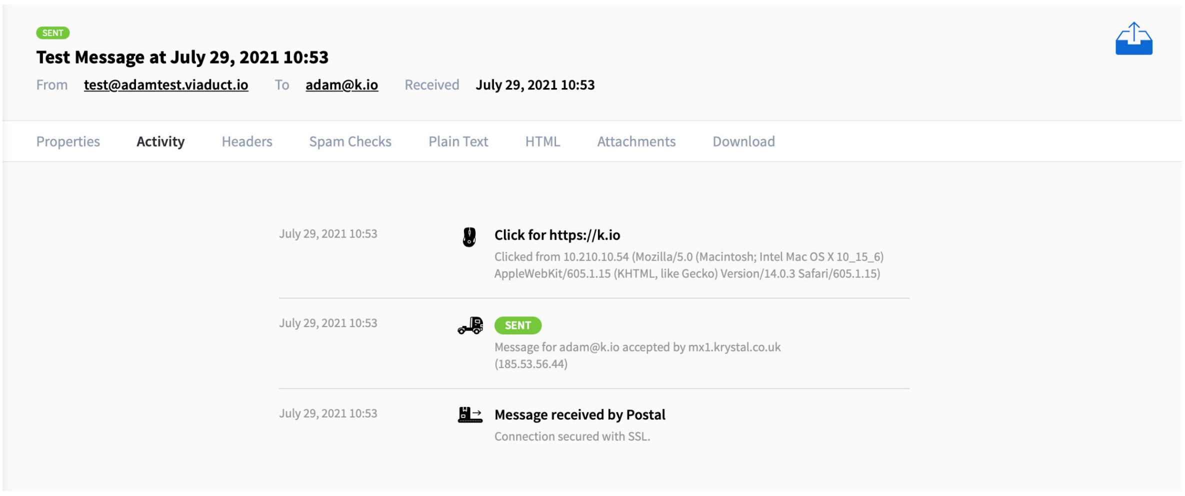
If you want to simply install Postal on a new server and copy your data over, you can do so by following these instructions.

1. Create your new server and follow the instructions for installing Postal. You should have a working installation at this point.
2. On your old server, stop Postal using `postal stop`. Make sure it has fully stopped before continuing using `postal status`.
3. On your new server, stop Postal using `postal stop`.
4. Use whatever tool takes your fancy (`mysqldump`, `Mariabackup` etc...) to copy your databases to your new server. Make sure you copy the `postal` database as well as all other databases prefixed with `postal` (or whatever you have configured your prefix to be in the `message_db` part of your configuration).
5. On your new server, run `postal upgrade-db` to update the copied database with the changed table structures
6. Ensure that your `postal.yml` is merged appropriately. For example, make sure your `dns` section is correct. There is no need to copy the `rails.secret` - a new secret on the new host won't be a problem.
7. If you stopped Postal cleanly before beginning, there is no need to copy any persisted data from RabbitMQ.
8. Shutdown your old Postal server.
9. Move the IP address(es) from the old server to the new one (if both old and new servers are with the same provider).
10. Start Postal on the new server using `postal start`.

Features

Click & Open Tracking

Postal supports tracking opens and clicks from e-mails. This allows you to see when people open messages or they click links within them.



Test Message at July 29, 2021 10:53

From test@adamtest.viaduct.io To adam@k.io Received July 29, 2021 10:53

July 29, 2021 10:53 — **Click for https://k.io**
Clicked from 10.210.10.54 (Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_6) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/14.0.3 Safari/605.1.15)

July 29, 2021 10:53 — SENT
Message for adam@k.io accepted by mx1.krystal.co.uk (185.53.56.44)

July 29, 2021 10:53 — **Message received by Postal**
Connection secured with SSL.

How it works

Once enabled, Postal will automatically scan your outgoing messages and replace any links and images with new URLs that go via your Postal web server. When the link is clicked, Postal will log the click and redirect to the user to the original URL automatically. The links that are included in the e-mail should be on the same domain as the sender and therefore you need to configure a subdomain like `click.yourdomain.com` and point it to your Postal server.

Configuring your web server

To avoid messages being marked as spam, it's important that the subdomain that Postal uses in the re-written URLs is on the same domain as that sending the message. This means if you are sending mail from `example.com`, you'll need to setup `click.example.com` (or whatever you choose) to point to your Postal server.

You'll need to add an appropriate virtual host on your web server that proxies traffic from that domain to the Postal web server. The web server must add the `X-Postal-Track-Host: 1` header so the Postal web server knows to treat requests as tracking requests.

Once you have configured this, you should be able to visit your chosen domain in a browser and see `Hello.` printed back to you. If you don't see this, review your configuration until you do. If you still don't see this and you enable the tracking, your messages will be sent with broken links and images.

If you're happy things are working, you can enable tracking as follows:

1. Find the web server you wish to enable tracking on in the Postal web interface
2. Go to the **Domains** item
3. Select **Tracking Domains**
4. Click **Add a tracking domain**

5. Enter the domain that you have configured and choose the configuration you want to use. It is **highly** recommended that you use SSL for these connections. Anything else is likely to cause problems with reputation and user experience.

Disabling tracking on a per e-mail basis

If you don't wish to track anything in an email you can add a header to your e-mails before sending it.

```
X-AMP: skip
```

Disabling tracking for certain link domains

If there are certain domains you don't wish to track links from, you can define these on the tracking domain settings page. For example, if you list `yourdomain.com` no links to this domain will be tracked.

Disabling tracking on a per link basis

If you wish to disable tracking for a particular link, you can do so by inserting `+notrack` as shown below. The `+notrack` will be removed leaving a plain link.

- `https+notrack://postalserver.io`
- `http+notrack://katapult.io/signup`

Health & Metrics

The Postal worker and SMTP server processes come with additional functionality that allows you to monitor the health of the process as well as look at live metrics about their performance.

Port numbers

By default, the health server listens on a different port for each type of process.

- Worker - listens on port `9090`
- SMTP server - listens on port `9091`

Unlike other services, if these ports are in use when the process starts, the health server will simply not start but the rest of the process will run as normal. This will be shown in the logs.

To configure these ports you can set the `HEALTH_SERVER_PORT` and `HEALTH_SERVER_BIND_ADDRESS` environment variables.

Metrics

The metrics are exposed at `/metrics` and are in a standard Prometheus exporter format. This means they can be scraped by any tool that can ingest Prometheus metrics. This will then allow them to be turned in to graphs as appropriate.

Health checks

The `/health` endpoint will return "OK" when the process is running. This can be used for health check monitoring.

IP Pools

Postal supports sending messages from different IP addresses. This allows you to configure certain sets of IPs for different mail servers or send from different IPs based on the sender or recipient addresses.

Enabling IP pools

By default, IP pools are disabled and all email is sent from any IP address on the host running the workers. To use IP pools, you'll need to enable them in the configuration file. You can do this by setting the following in your `postal.yml` configuration file. You'll then need to restart Postal using `postal stop` and `postal start`.

```
postal:
  use_ip_pools: true
```

Configuring IP pools

Once you have enabled IP pools, you'll need to set them up within the web interface. You'll see an **IP Pools** link in the top right of the interface. From here you can add pools and then add IP addresses within them.

Once an IP pool has been added, you'll need to assign it any organization that should be permitted to use it. Open up the organization and choose **IPs** and then tick the pools you want to allocate.

Once allocated to an organization, you can assign the IP pool to servers from the server's **Settings** page. You can also use the IP pool to configure IP rules for the organization or server.

It's **very important** to make sure that the IP addresses you add in the web interface are actually configured on your Postal servers. If the IPs don't exist on the server, message delivery may fail or messages will not be dequeued correctly.

Logging

All Postal processes log to STDOUT and STDERR which means their logs are managed by whatever engine is used to run the container. In the default case, this is Docker.

Redirecting logs to the host syslog

If you want to send your log data to the host system's syslog then you can configure this. This is useful if you wish to use external tools like `fail2ban` to block users from accessing your system.

The quickest way to achieve this is to use a docker compose overide file in `/opt/postal/install/docker-compose.override.yml`. The contents of this file, would contain the following:

```yaml
version: "3.9"
services:
  smtp:
    logging:
      driver: syslog
      options:
        tag: postal-smtp
```

If you wanted to put worker and web server logs there too, you can define those. The example above demonstrates using the `smtp` server process.

Limiting the size of logs

Docker cam be configured to limit the size of the log files it stores. To avoid storing large numbers of log files, you should configure this appropriately. This can be achieved by setting a maximum size in your `/etc/docker/daemon.json` file.

```json
{
  "log-driver": "local",
  "log-opts": {
    "max-size": "100m"
  }
}
```

Sending logs to Graylog

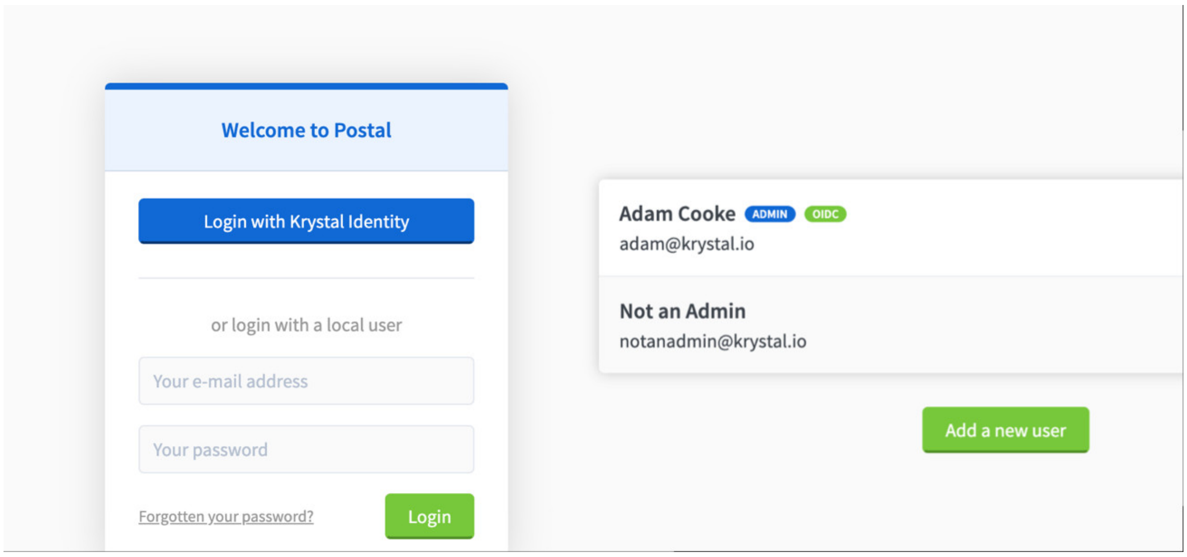
Postal includes support for sending log output to a central Graylog server over UDP. This can be configured using the following options:

```yaml
gelf:
  # GELF-capable host to send logs to
  host:
  # GELF port to send logs to
  port: 12201
  # The facility name to add to all log entries sent to GELF
  facility: postal
```

OpenID Connect

Postal supports OpenID Connect (OIDC) allowing you to delegate authentication to an external service. When enabled, there are various changes:

- You are not required to enter a password when you add new users.
- When a user first logs in with OIDC, they will be matched to a local user based on their e-mail address.
- On subsequent logins, the user will be matched based on their unique identifier provided by the OIDC issuer.
- Users without local passwords cannot reset their password through Postal.
- Users cannot change their local password when associated with an OIDC identity.
- Existing users that currently have a password will continue to be able to use that password until it is linked with an OIDC identity.



Configuration

To get started, you'll need to find an OpenID Connect enabled provider. You should create your application within the provider in order to obtain a identifier (client ID) and a secret (client secret).

You may be prompted to provide a "redirect URI" during this process. You should enter `https://postal.yourdomain.com/auth/oidc/callback`.

Finally, you'll need to place your configuration in the Postal config file as normal.

```yaml
oidc:
  # Start by enabling OIDC for your installation.
  enabled: true

  # The name of the OIDC provider as shown in the UI. For example:
  # "Login with My Proivder".
  name: My Provider

  # The OIDC issuer URL provided to you by your Identity provider.
  # The provider must support OIDC Discovery by hosting their configuration
  # at https://identity.example.com/.well-known/openid-configuration.
  issuer: https://identity.example.com

  # The client ID for OIDC
  identifier: abc1234567890

  # The client secret for OIDC
  secret: zyx0987654321

  # Scopes to request from the OIDC server. You'll need to find these from your
  # provider. You should ensure you request enough scopes to ensure the user's
  # email address is returned from the provider.
  scopes:
    - openid
    - email
```

If your Identity Provider does not support OpenID Connect discovery (which is enabled by default, you can manually configure it.) For full details of the options available see the example config file.

By default, Postal will look for an email address in the `email` field and a name in the `name` field. These can be overriden using configuration if these values can be found elsewhere.

Logging in

Once enabled, you can log in by pressing the **Login with xxx** button on the login page. This will direct you to your chosen identity provider. Once authorised, you will be directed back to the application. If a user exists matching the e-mail address returned by the OpenID provider, it will be linked and you will be logged in. If not, an error will be displayed.

Debugging

Details about the user matching process will be displayed in the web server logs when the callback from the Identity provider happens.

Disabling local authentication

Once you have established your OpenID Connect set up, you can fully disable local authentication. This will change the login page as well as user management options.

```
oidc:
  # ...
  local_authentication_enabled: false
```

Using Google as an identity provider

Setting up Postal to authenticate with Google is fairly straight forward. You'll need to use the Google Cloud console to generate a client ID and secret (see docs). When prompted for a redirect URI, you should be `https://postal.yourdomain.com/auth/oidc/callback`. The following configuration can be used to enable this:

```
oidc:
  enabled: true
  name: Google
  issuer: https://accounts.google.com
  identifier: # your client ID from Google
  secret: # your client secret from Google
  scopes: [openid, email]
```

SMTP Authentication

For sending outgoing emails through the Postal SMTP server you will need to generate a **credential** through the Postal web interface. This credential is associated with a server and allows you to send mail from any domain associated with that domain (or the organization that owns the domain.)

Authentication types

When authenticating to the SMTP server, there are three supported authentication types.

- `PLAIN` - the credentials are passed in plain text to the server. When using this, you can provide any string as the username (e.g. `x`) and the password should contain your credential string.
- `LOGIN` - the credentials are passed Base64-encoded to the server. As above, you can use anything as the username and the password should contain the credential string (Base64-encoded).
- `CRAM-MD5` - this is a challenge-response mechanism based on the HMAC-MD5 algorithm. Unlike the above two mechanism, the username does matter and should contain the organization and server permalinks separated by a `/` or `_` character. The password used should be the value from your credential.

From/Sender validation

When sending outgoing email through the SMTP server, it is important that the `From` header contains a domain that is owned by the server or its organization. If this it not valid, you will receive a `530 From/Sender name is not valid` error.

If you have enabled "Allow Sender Header" for the server, you can include this domain in the `Sender` header instead and any value you wish in the `From` header.

IP-based authentication

Postal has the option to authenticate clients based on their IP address. To use this, you need to create an **SMTP-IP** credential for the IP or network you wish to allow to send mail. Use this carefully to avoid creating an open relay.

SMTP TLS

By default, Postal's SMTP server is not TLS enabled however you can enable it by generating and providing a suitable certificate. We recommend that you use a certificate issued by a regnognised certificate authority but this isn't essential to use this feature.

Key & certificate locations

Certificates should be placed in your `/opt/postal/config` directory.

- `/opt/postal/config/smtp.key` - the private key in PEM format
- `/opt/postal/config/smtp.cert` - the certificate in PEM format

Generating a self signed certificate

You can use the command below to generate a self signed certificate.

```
openssl req -x509 -newkey rsa:4096 -keyout /opt/postal/config/smtp.key -out
/opt/postal/config/smtp.cert -sha256 -days 365 -nodes
```

Configuration

Once you have a key and certificate you will need to enable TLS in the configuration file (`/opt/postal/config/postal.yml`). Additional options are available too.

```yaml
smtp_server:
  # ...
  tls_enabled: true
  # tls_certificate_path: other/path/to/cert/within/container
  # tls_private_key_path: other/path/to/cert/within/container
  # tls_ciphers:
  # ssl_version: SSLv23
```

You will need to run `postal restart` if you change the configuration or your key/certificate.

Spam & Virus Checking

Postal can integrate with SpamAssassin and ClamAV to automatically scan incoming and outgoing messages that pass through mail servers.

This functionality is disabled by default.

Setting up SpamAssassin

By default, Postal will talk to SpamAssassin's `spamd` using an TCP socket connection (port 783). You'll need to install SpamAssassin on your server and then enable it within Postal.

Installing SpamAssassin

```
sudo apt install spamassassin
```

Once you have installed this, you will need to open up `/etc/default/spamassassin` and change `ENABLED` to `1` and `CRON` to `1`. On some systems (such as Ubuntu 20.04 or newer), you might need to enable the SpamAssassin daemon with the following command.

```
update-rc.d spamassassin enable
```

Then you should restart SpamAssassin.

```
sudo systemctl restart spamassassin
```

Enabling in Postal

To enable spam checking, you'll need to add the following to your `postal.yml` configuration file and restart. If you have installed SpamAssassin on a different host to your Postal installation you can change the host here but be sure to ensure that the `spamd` is listening on your external interfaces.

```
spamd:
  enabled: true
  host: 127.0.0.1
  port: 783
postal stop
postal start
```

Classifying Spam

The spam system is based on a numeric scoring system and different scores are assigned to different issues which may appear in a message. You can configure different thresholds which define when a message is treated as spam. We recommend starting at 5 and updating this once you've seen how your incoming messages are classified.

You have three options which can be configured on a per route basis which defines how spam messages are treated:

- **Mark** - messages will be sent through to your endpoint but the spam information will be made available to you.
- **Quarantine** - the message will placed into your hold queue and you'll need to release them if you wish them to be passed to your application. They will only remain here for 7 days,
- **Fail** - the message will be marked as failed and will only be recorded in your message history without being sent.

Developer

Using the API

The HTTP API allows you to send messages to us using JSON over HTTP. You can either talk to the API using your current HTTP library or you can use one of the pre-built libraries.

Full API documentation

This API does not support managing all the functions of Postal. There are plans to introduce a new v2 API which will have more functionality and significantly better documentation. We do not have an ETA for this. Additionally, we will not be accepting any pull requests to extend the current API to have any further functionality than it currently does.

General API Instructions

- You should send POST requests to the URLs shown below.
- Parameters should be encoded in the body of the request and `application/json` should be set as the `Content-Type` header.
- The response will always be provided as JSON. The status of a request can be determined from the `status` attribute in the payload you receive. It will be `success` or `error`. Further details can be found in the `data` attribute.

An example response body is shown below:

```
{
  "status":"success",
  "time":0.02,
  "flags":{},
  "data":{"message_id":"xxxx"}
}
```

To authenticate to the API you'll need to create an API credential for your mail server through the web interface. This is a random string which is unique to your server.

To authenticate a request to the API, you need to pass this key in the `X-Server-API-Key` HTTP header.

Sending a message

There are two ways to send a message - you can either provide each attribute needed for the e-mail individually or you can craft your own RFC 2822 message and send this instead.

Full details about these two methods can be found in our API documentation:

- Sending a message
- Sending an RFC 2822 message

For both these methods, the API will return the same information as the result. It will contain the `message_id` of the message that was sent plus a `messages` hash with the IDs of the messages that were sent by the server to each recipient.

```
{
  "message_id":"message-id-in-uuid-format@rp.postal.yourdomain.com",
  "messages":{
    "john@example.com":{"id":37171, "token":"a4udnay1"},
    "mary@example.com":{"id":37172, "token":"bsfhjsdfs"}
  }
}
```

Client Libraries

There are a number of client libraries available to help send e-mail using the Postal platform. These aren't all developed by the Postal team.

- Ruby
- Rails
- Ruby (mail gem)
- PHP
- Node
- Java
- .Net
- Go

All of these libraries will make use of the API rather than using any SMTP protocol - this is considered to be best approach for delivering your messages.

If your framework makes use of SMTP, you do not need a client library however you will also miss out on some of Postals functionality.

Receiving e-mail by HTTP

One of the most useful features in Postal is the ability to have incoming messages delivered to your own application as soon as they arrive. To receive incoming messages from Postal you can set it up to pass them to an HTTP URL of your choosing.

Each endpoint has an HTTP URL (we highly recommend making use of HTTPS where possible) as well as a set of rules which defines how data is sent to you.

- You can choose whether data is encoded as normal form data or whether Postal sends JSON as the body of the request.
- You can choose whether to receive the raw message (raw) or have it as a JSON dictionary (processed).
- You can choose whether you'd like replies and signatures to be separated from the plain body of the message.

Your server should accept Postals incoming request and reply within 5 seconds. If it takes longer than this, Postal will assume it has failed and the request will be retried. Your server should send a `200 OK` status to signal to Postal that you've received the request.

Messages will be tried up to 18 times with an exponential back-off until a successful response is seen except in the case of `5xx` statuses which will fail immediately.

When a message permanently fails to be delivered to your endpoint (i.e. the server returned a 5xx status code or it wasn't accepted after 18 attempts), the recipient will be sent a bounce message.

You can view the attempts (along with debugging information) on the message page in the web interface.

The processed payload

When you chose to receive the message as JSON (processed), you'll receive a payload with the following attributes.

```
{
  "id":12345,
  "rcpt_to":"sales@awesomeapp.com",
  "mail_from":"test@example.com",
  "token":"rtmuzogUauKN",
  "subject":"Re: Welcome to AwesomeApp",
  "message_id":"81026759-68fb-4872-8c97-6dd2901cb33a@rp.postal.yourdomain.com",
  "timestamp":1478169798.924355,
  "size":"822",
  "spam_status":"NotSpam",
  "bounce":false,
  "received_with_ssl":false,
  "to":"sales@awesomeapp.com",
  "cc":null,
  "from":"John Doe <test@example.com>",
  "date":"Thu, 03 Nov 2016 10:43:18 +0000",
  "in_reply_to":null,
  "references":null,
  "plain_body":"Hello there!",
  "html_body":"<p>Hello there!</p>",
  "auto_submitted":"auto-reply",
  "attachment_quantity":1,
  "attachments":[
    {
      "filename":"test.txt",
      "content_type":"text/plain",
      "size":12,
      "data":"SGVsbG8gd29ybGQh"
    }
  ]
}
```

- You will only have the `attachments` attribute if you have enabled it.
- The `data` attribute for each attachment is Base64 encoded.

The raw message payload

When you choose to receive the full message, you will receive the following attributes.

```
{
  "id":12345,
  "message":"REtJTS1TaWduYXR1cmU6IHY9MTsgYT1yc2Etc2hhMjU2Oy...",
  "base64":true,
  "size":859
}
```

- The `base64` attribute specifies whether or not the `message` attribute is encoded with Base64. This is likely to be true all the time.

Webhooks

Postal supports sending webhooks over HTTP when various events occur during the lifecycle of a message.

This page lists all the different types of event along with an example JSON payload that you'll receive. In many cases, only a small amount of information will be sent, if you require more information you should use the API to obtain it.

Message Status Events

These events are triggered on various events in an e-mail's lifecycle. The payload format is identical for all messages however the `status` attribute may change. The following statuses may be delivered.

- `MessageSent` - when a message is successfully delivered to a recipient/endpoint.
- `MessageDelayed` - when a message's delivery has been delayed. This will be sent each time Postal attempts a delivery and a message is delayed further.
- `MessageDeliveryFailed` - when a message cannot be delivered.
- `MessageHeld` - when a message is held.

```
{
  "status":"Sent",
  "details":"Message sent by SMTP to aspmx.l.google.com (2a00:1450:400c:c0b::1b)
(from 2a00:67a0:a:15::2)",
  "output":"250 2.0.0 OK 1477944899 ly2si31746747wjb.95 - gsmtp",
  "time":0.22,
  "sent_with_ssl":true,
  "timestamp":1477945177.12994,
  "message":{
    "id":12345,
    "token":"abcdef123",
    "direction":"outgoing",
    "message_id":"5817a64332f44_4ec93ff59e79d154565eb@app34.mail",
    "to":"test@example.com",
    "from":"sales@awesomeapp.com",
    "subject":"Welcome to AwesomeApp",
    "timestamp":1477945177.12994,
    "spam_status":"NotSpam",
    "tag":"welcome"
  }
}
```

Message Bounces

If Postal receives a bounce message for a message that was previously accepted, you'll receive the `MessageBounced` event.

```
{
  "original_message":{
    "id":12345,
    "token":"abcdef123",
    "direction":"outgoing",
    "message_id":"5817a64332f44_4ec93ff59e79d154565eb@app34.mail",
    "to":"test@example.com",
    "from":"sales@awesomeapp.com",
    "subject":"Welcome to AwesomeApp",
    "timestamp":1477945177.12994,
    "spam_status":"NotSpam",
    "tag":"welcome"
  },
  "bounce":{
    "id":12347,
    "token":"abcdef124",
    "direction":"incoming",
    "message_id":"5817a64332f44_4ec93ff59e79d154565eb@someserver.com",
    "to":"abcde@psrp.postal.yourdomain.com",
    "from":"postmaster@someserver.com",
    "subject":"Delivery Error",
    "timestamp":1477945179.12994,
    "spam_status":"NotSpam",
    "tag":null
  }
}
```

Message Click Event

If you have click tracking enabled, the `MessageLinkClicked` event will tell you that a user has clicked on a link in one of your e-mails.

```
{
  "url":"https://atech.media",
  "token":"VJzsFA0S",
  "ip_address":"185.22.208.2",
  "user_agent":"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_6)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/54.0.2840.98 Safari/537.36",
  "message":{
    "id":12345,
    "token":"abcdef123",
    "direction":"outgoing",
    "message_id":"5817a64332f44_4ec93ff59e79d154565eb@app34.mail",
    "to":"test@example.com",
    "from":"sales@awesomeapp.com",
    "subject":"Welcome to AwesomeApp",
    "timestamp":1477945177.12994,
    "spam_status":"NotSpam",
    "tag":"welcome"
  }
}
```

Message Loaded/Opened Event

If you have open tracking enabled, the `MessageLoaded` event will tell you that a user has opened your e-mail (or, at least, have viewed the tracking pixel embedded within it.)

```
{
  "ip_address":"185.22.208.2",
  "user_agent":"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_6)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/54.0.2840.98 Safari/537.36",
  "message":{
    "id":12345,
    "token":"abcdef123",
    "direction":"outgoing",
    "message_id":"5817a64332f44_4ec93ff59e79d154565eb@app34.mail",
    "to":"test@example.com",
    "from":"sales@awesomeapp.com",
    "subject":"Welcome to AwesomeApp",
    "timestamp":1477945177.12994,
    "spam_status":"NotSpam",
    "tag":"welcome"
  }
}
```

DNS Error Event

Postal regularly monitors domains it knows about to ensure that your SPF/DKIM/MX records are correct. If you'd like to be notified when the checks fail, you can subscribe to the `DomainDNSError` event.

```
{
  "domain":"example.com",
  "uuid":"820b47a4-4dfd-42e4-ae6a-1e5bed5a33fd",
  "dns_checked_at":1477945711.5502,
  "spf_status":"OK",
  "spf_error":null,
  "dkim_status":"Invalid",
  "dkim_error":"The DKIM record at example.com does not match the record we have
provided. Please check it has been copied correctly.",
  "mx_status":"Missing",
  "mx_error":null,
  "return_path_status":"OK",
  "return_path_error":null,
  "server":{
    "uuid":"54529725-8807-4069-ab29-a3746c1bbd98",
    "name":"AwesomeApp Mail Server",
    "permalink":"awesomeapp",
    "organization":"atech"
  },
}
```

Other Notes

Auto-Responders & Bounces

When you send an e-mail you may be automatically be sent a bounce message or an auto responder by the destination mail server. These are handled slightly differently to normal incoming e-mail and it's worth understanding how these work.

Messages like these aren't usually sent to the e-mail address that sent the message but are sent to the **return path** address. This is an address which Postal assigns to your mail server and is provided to the destination mail server for the purpose of sending this type of message. The return to address will be something like `abcdef@psrp.yourdomain.com`.

If you wish to route mail which is sent to your return path address to your application, you can set up a **return path route**. This is the same as a normal route except you should enter the name as `__returnpath__` and leave the domain field blank. You can only choose HTTP endpoints for this type of route. Once added, messages sent to your return path that aren't detected as bounces will be sent to HTTP endpoint you choose.

A note about bounces

Messages that Postal detects as being bounces for a message you have already sent will not be delivered to your return path route. The original message will be updated and a `MessageBounced` webhook event will be triggered.

Our container image

This page contains information about how Postal actually is packaged and run within a container.

Where's the container?

Postal is packaged and hosted at `ghcr.io/postalserver/postal`.

The `latest` tag will always track the `main` branch within the repository and therefore will have the latest copy of the application. It is not recommended to use this tag in production because you may start using it at any time without noticing.

Each version of Postal will also be tagged, for example, `3.0.0`. We always recommend using a version tag in production. To upgrade, you simply need to start using the newer version of the container and be sure to run the `upgrade` command. You can view all the tags which exist on GitHub and in our CHANGELOG.

What processes need to run?

There are 3 processes that need to be running for a successful Postal installation. All processes are run within the same image (`ghcr.io/postalserver/postal`). The table below identifies the processes.

The web server

- **Command:** `postal web-server`
- **Ports:** 5000

This is the main web server that handles all web traffic for Postal's admin interface and open/click tracking requests. By default, it listens on port 5000 but this can be configured in the `postal.yml` file by changing the `web_server.default_port` option or setting the `PORT` environment variable.

You can run multiple web servers and load balance between them to add additional capacity for web requests.

The SMTP server

- **Command:** `postal smtp-server`
- **Ports:** 25
- **Capabilities required:** `NET_BIND_SERVICE`

This is the main SMTP server for receiving messages from clients and other MTAs. As with the web server, you can configure this to run on any port by changing the `smtp_server.default_port` option in the `postal.yml` config file or setting the `PORT` environment variable.

You can run multiple SMTP servers and load balance between them to add additional capacity for SMTP connections.

The worker(s)

- **Command:** `postal worker`

This runs a worker which will receive jobs from the message queue. Essentially, this handles processing all incoming and outgoing e-mail. If you're needing to process lots of e-mails, you may wish to run more than one of these. You can run as many of these as you wish.

Configuration

The image expects all configuration to be mounted at `/config`. At a minimum, this directory must contain a `postal.yml` and a `signing.key`. You can see a minimal example `postal.yml` in the installation tool repository. For a full example, see here.

The `signing.key` can be generated using the following command:

```
openssl genrsa -out path/to/signing.key 2018
```

Networking

If you wish to utilise IP pools, you will need to run Postal using host networking. This is because Postal will need to be able to determine which physical IPs are available to it and be able to send and receive traffic on those IPs.

If you are not using IP pools, there is no need to use host networking and you can expose the ports listed above as appropriate.

Waiting for services

The container's entrypoint supports waiting for external services to be ready before starting the underlying process. To use this you need to set the `WAIT_FOR_TARGETS` environment variable with a list of services and ports. For example, `mariadb:3306`, replacing `mariadb` with the hostname or IP of your MariaDB server. You can specify multiple endpoint by separating them with a space.

The default maximum time to wait is 30 seconds, you can override this using the `WAIT_FOR_TIMEOUT` environment variable.

Debugging

This page contains information on how to identify problems with your installation.

DNS

Whilst Postal can verify DNS records on its own, it can be useful to check what the rest of the internet is seeing.

You can make use of Whats My DNS for a quick global check or the `dig` command if you have it on a terminal, i.e. `dig txt yourdomain.com` or `dig a yourdomain.com`.

SMTP

The quickest way to ensure the internet can connect to your Postal SMTP is with `telnet postal.yourdomain.com 25`. You can proceed to attempt sending a message manually if you are familiar with the raw SMTP commands to further verify your Postal installation is working correctly.

SMTP SSL

If you aren't sure about whether the SSL certificate you have provided to Postal is set up correctly you can use `openssl s_client -connect postal.yourdomain.com:25 -starttls smtp` to get some information about your certificate from the point of view of SMTP.

Wildcards & Address Tags

Postal supports the use of wildcards and address tags in routes.

Using a wildcard

Wildcards will allow you to receive all e-mail for every address on a domain. However, for most use cases, this isn't recommended because it means that large volumes of spam may be processed by your mail server that would otherwise have been rejected by Postal without troubling you.

Just enter `*` in the name box to create a route for this.

Using address tags

Postal supports the use of "tags" on e-mail addresses which means you can add a single route but still receive from multiple addresses. For example, if you add a route for `email` you will also receive messages for `email+anything@yourdomain.com` without any additional configuration.

微信　　支付宝

来杯冰阔落，扫码捐赠网管小贾~

扫码关注微信公众号@网管小贾，个人微信：sysadmcc

网管小贾 / sysadm.cc