Receiving, Forwarding,and Storing Messages

Table of Contents

Introduction to Receiving, Forwarding and Storing Messages

Mailgun will allow you to receive emails sent to your Mailgun Domain using the Routes feature, which will perform action that can include:

- Forwarding emails to a different email address
- Send the data to a configured webhook/URL (http POST to a URL)
- Storing the email on Mailgun's servers temporarily to allow retrieval at a different time

Routes

You can define a list of routes to handle incoming emails. This idea of routes is borrowed from MVC web frameworks like Django or Ruby on Rails. If a message matches a route expression, Mailgun can perform an action, such as forward the email, or store the email.

You can define routes visually by clicking the **Receiving** tab in the Control Panel, or programmatically using the Routes API. For more on setting up Routes, see How Do I Setup a Route?.

A Route is a pair of **filter + action**. Each incoming message is passed to a filter expression, and if it evaluates to true, the action is executed.

Each Route can be assigned a priority, and are evaluated in the order of priority, with lower numbers having a higher priority. By default, all Routes are evaluated (even if a higher priority Route is triggered). To avoid this action, you can use a stop() action (see below).

Route Properties

Route	Description
`Priority`	Integer showing the priority of route execution. Lower numbers have higher priority.
`Filter`	Filters available in routes - match_recipient() match_header() catchall() (see Route Filters for description).
`Actions`	Type of action to take when a filter is triggered - forward() store() stop() (see below for description).
`Description`	Arbitrary string to describe the route (shown in the Control Panel UI)

Note: The length of the **Filter** or **Action** fields cannot exceed 4k. If you need more actions or filters than is allowed under the 4k limit, you can add additional routes. Multiple routes with the same Filter expression are allowed. This will allow you to add many more Actions for the same Filter but spread across multiple route entries.

Route Filters

Route filters are expressions that decide when an action is triggered. A filter is created based on the recipient of the incoming email, the headers in the incoming email or use a catch-all filter. Filters support regular expressions in the pattern to give you a lot of flexibility when creating them.

Match Recipient(pattern)

Matches the SMTP recipient of the incoming message against the regular expression pattern. For example, this filter will match messages going to foo@bar.com:

```
match_recipient("foo@bar.com")
```

You can use Python-style regular expressions in your filter. For example, this will match all messages coming to any recipient at @bar.com:

```
match_recipient(".*@bar.com")
```

Another example, handling plus addressing for a specific recipient:

```
match_recipient("^chris\+(.*)@example.com$")
```

Mailgun supports regexp captures in filters, which allows you to use captured values inside of your actions. The example below captures the local name (the part of email before @) and passes it as a mailbox parameter to an application URL:

```
route filter : match_recipient("(.*)@bar.com")
route action : forward("http://myhost.com/post/?mailbox=\1")
```

You can use named captures as well:

```
route filter : match_recipient("(?P<user>.*?)@(?P<domain>.*)")
route action : forward("http://mycallback.com/domains/\g<domain>/users/\g<user>")
```

Match Header(header,pattern)

This is similar to match-recipient, only instead of looking at a message recipient, it applies the pattern to an arbitrary MIME header for the message.

The example below matches any message with a word "support" in its subject:

```
match_header("subject", ".*support")
```

The example below matches any message against several keywords:

```
match_header('subject', '(.*)(urgent|help|asap)(.*)')
```

The example below will match any messages deemed spam (if spam filtering is enabled):

```
match_header('X-Mailgun-Sflag', 'Yes')
```

match*recipient(pattern) AND match*header(header, pattern)

The example below will match any recipient for a domain, then match if the message is in English:

```
match_recipient('^(.*)@example.com$') and match_header("Content-Language", "^(.*)en-US(.*)$")
```

catch_all()

Will create matches if no proceeding routes matched. Usually, you need to use it in a route with a lowest priority, to make sure it evaluates last.

Route Actions

If a route expression is evaluated to *true,* Mailgun executes the corresponding action. Currently you can use the following three actions in your routes: forward(), store() and stop().

Forward(destination)

Forwards the message to a specified destination, which can be another email address or a URL. A few examples:

```
forward("mailbox@myapp.com")
forward("http://myapp.com/messages")
```

You can combine multiple destinations by separating them with a comma.

```
forward("http://myapp.com/messages, mailbox@myapp.com")
```

Note: When forwarding messages to another email address, you should disable click tracking, and open tracking and unsubscribes, by editing your domain settings in the Control Panel. If these features are enabled, the content of each message is modified by Mailgun before forwarding, which invalidates the DKIM signature. If the message comes from a domain publishing a DMARC policy (like Yahoo! Mail), the message will be rejected as spam by the forwarding destination.

Store(notification endpoint)

This temporarily stores the message (for up to 3 days) on Mailgun's servers so that you can retrieve it later. This is helpful for large attachments that may cause time-outs, or if you want to retrieve them later to reduce the frequency of hits on your server.

When you specify a URL, Mailgun will notify you when the email arrives along with a URL which you can use to retrieve the message:

```
store(notify="http://mydomain.com/callback")
```

If you don't specify a URL with the notify parameter, the message will still be stored and you can get the message later through the Messages API. You can see a full list of parameters we will post/return to you below.

Stop()

Without a stop() action executed, all lower priority Routes will also be evaluated. This simply stops the priority waterfall so the subsequent routes won't be evaluated.

Receiving Messages via HTTP through a forward() action

When a URL is specified as a route destination through a forward() action, Mailgun will perform an HTTP POST request into the URL using one of the two formats:

- **Fully parsed** : Mailgun will parse the message, encode it into UTF-8, process the attachments, and attempt to separate quoted parts from the actual message. * Preferred Option.
- **Raw MIME** : The message will be posted as-is. You are responsible for parsing MIME. To receive raw MIME message, the destination URL must end with mime.

For Route POSTs, Mailgun listens to codes from your server and reacts accordingly:

Received by Mailgun	Code description
`200 (Success)`	When Mailgun receives this code, it will determine the webhook POST is successful and will not be retried.
`406 (Not Applicable)`	When this code is received, Mailgun will determine the POST is rejected and it will not be retried.
`Any other code`	Mailgun will try POSTing according to the schedule (below) for webhooks other than the delivery notification.

If a 406 error code is not returned and your application is unable to process the webhook request, Mailgun will attempt to retry (other than for delivery notification) in intervals for 8 hours before stopping to try. The intervals are 10 minutes, 15 minutes, 30 minutes, 1 hour, 2 hours, and 4 hours.

You can use these two tables of HTTP parameters to determine what you can expect to be posted into your applications through a forward() action.

Parsed Messages Parameters

Parameter	Type	Description
`recipient`	string	The recipient of the message as reported by MAIL TO during SMTP chat
`sender`	string	The sender of the message as reported by MAIL FROM during SMTP chat. Note: this value may differ from From MIME header
`from`	string	The sender of the message as reported by from message header, for example "Bob <bob@example.com>"
`subject`	string	Subject string
`Body-plain`	string	The text version of the email. This field is always present. If the incoming message only has HTML body, Mailgun will create a text representation for you.
`stripped-text`	string	The text version of the message without quoted parts and signature block (if found)
`stripped-signature`	string	The signature block stripped from the plain text message (if found)
`body-html`	string	The HTML version of the message, if message was multipart. Note that all parts of the message will be posted, not just text/html. For instance, if a message arrives with "foo" part it will be posted as "body-foo"
`stripped-html`	string	The HTML version of the message, without quoted parts.
`Attachment-count`	int	The number of attachments the message has.
`Attachment-x`	string	The attached file ('x' stands for number of the attachment). Attachments are handled as file uploads, encoded as multipart/form-data.
`timestamp`	int	The number of seconds passed since January 1, 1970 (*see securing web hooks*)
`token`	string	A randomly generated string with a length of 50 (*See securing webhooks)*
`signature`	string	A string with hexadecimal digits generated by HMAC algorithm (see securing webhooks).
`message-headers`	string	A list of MIME headers dumped to a JSON string (*order of headers is preserved*)
`Content-id-map`	string	JSON-encoded dictionary which maps Content-ID (CID) of each attachment to the corresponding attachment-x parameter. This allows you to map posted attachments to tags like <img src='cid'> in the message body.

Note: Not all web frameworks support multi-valued keys parameters, so the message-headers parameter was added.

Example: Ruby on Rails requires a special syntax to post params like that: you need to add [] to a key to collect its values on the server side as an array.

Below is a Ruby on Rails example of obtaining MIME headers via message-headers parameter:

MIME Messages Parameters

Parameter	Type	Description
`recipient`	string	The recipient of the message
`sender`	string	The sender of the message as reported by SMTP MAIL FROM
`from`	string	The sender of the message as reported by from message header, for example "Bob <bob@example.com>".
`subject`	string	The subject string
`Body-mime`	string	The full MIME envelope. You will need a MIME parsing library to process this data.
`timestamp`	int	The number of seconds passed since January 1, 1970 (*See Securing Webhooks)*
`token`	string	A randomly generated string with a length of 50 (*See Securing Webhooks)*
`signature`	string	A string with hexadecimal digits generated by HMAC algorithm (S*ee securing webhooks*).

Note: To receive raw MIME messages and perform your own parsing, you must configure a route with a URL ending with "mime". Example: **http://myhost/post_mime**

*Consider using **http:/bin.mailgun.net** to debug and play with your routes. This tool allows you to forward incoming messages to a temporary URL and inspect the posted data.

Storing and Retrieving Messages

When storing an email through a store() action in a Route, you can choose to be notified when the message is stored by including a URL with the notify parameter when setting up the store action or you can retrieve the message later by searching for the message through the Events API and retrieving it through the Messages API.

When you set a URL to be posted when the message is received:

(store(notify="http://mydomain.com/callback") or retrieve the message later through a GET request to the Messages API, the following parameters are posted/returned in JSON.

If at least one attachment is written, then the resulting body will have a multipart/form-data content type, otherwise it will have an application/x-www-form-urlencoded content type.

Parameter	Type	Description
`domain`	String	The domain name this message was received from.
`recipient`	string	The recipient of the message as reported by MAIL TO during SMTP chat
`sender`	string	The sender of the message as reported by MAIL FROM during SMTP chat. *(This value may differ from the MIME header)*
`from`	string	The sender of the message as reported by from message header, for example "Bob Lee <blee@mailgun.net>".
`subject`	string	The subject string
`body-plain`	string	The text version of the email. This field is always present. If the incoming message only has HTML body, Mailgun will create a text representation for you.
`stripped-text`	string	The text version of the message without the quoted parts and signature block (if found)
`stripped-signature`	string	The signature block stripped from the plain text message (if found)
`body-html`	string	The HTML version of the message, if message was multipart. Note that all parts of the message will be posted, not just text/html. For instance, if a message arrives with "foo" part it will be posted as "body-foo"
`stripped-html`	string	The HTML version of the message, without the quoted parts
`attachments`	string	The string that contains a JSON list of metadata objects, one for each attachment.
`message-url`	string	A URL that you can use to get and/or delete the message. Only present in the payload posted to the notification URL
`timestamp`	int	The number of seconds passed since January 1, 1970 *(See Securing Webhooks)*
`token`	string	A randomly generated string with a length of 50 *(See Securing Webhooks)*
`signature`	string	A string with hexadecimal digits generated by HMAC algorithm *(See securing webhooks).*
`message-headers`	string	A list of MIME headers dumped to a JSON string (The o*rder of headers is preserved)*
`Content-id-map`	string	

Alternatively, you can choose the following parameters when the Accept header is set to message/rfc2822

<table>
<tr><th>Parameter</th><th>type</th><th>Description</th></tr>
<tr><td><code>recipient</code></td><td>string</td><td>The recipient of the message</td></tr>
<tr><td><code>sender</code></td><td>string</td><td>The sender of the message</td></tr>
<tr><td><code>from</code></td><td>string</td><td>The sender of the message as reported by the from message header. Example: <bob@example.com>"</td></tr>
<tr><td><code>subject</code></td><td>string</td><td>The subject string</td></tr>
<tr><td>`Body-mime | string | The Full MIME envelope. You will need a MIME parsing library to process this data</td><td></td><td></td></tr>
</table>

Spam Filter

A spam filter is necessary when receiving email. Mailgun is powered by an army of SpamAssassin machines. Mailgun gives you three ways to configure spam filtering.

Click the Domains tab on the Control Panel and select from one of the following three options:

- Disabled (default)
- Delete Spam (spam is removed and you won't see it)
- Mark spam with MIME headers (You decide what to do with it)

If you choose to mark spam with MIME headers, Mailgun provides you with these four:

- **X-Mailgun-Sflag** - Inserted with the value `Yes` if the message was classified as spam.
- **X-Mailgun-Sscore** – A 'spamicity' score that you can use to calibrate your own filter. Inserted for every message checked for spam. The score ranges from low negative digits (very unlikely to be spam) to 20 and occasionally higher (very likely to be spam).
- **X-Mailgun-Dkim-Check-Result** - If DKIM is used to sign an inbound message, Mailgun will attempt DKIM validation, the results will be stored in this header. Possible values are: `Pass` or `Fail`
- **X-Mailgun-Spf -Mailgun** will perform an SPF validation, and results will be stored in this header. Possible values are: `Pass` , `Neutral` , `Fail` or `SoftFail` .

Viewing Stored Messages

To access the contents of the stored messages (including raw MIME) you'll need the email's storage URL. This can be found on a Domains Accepted/Delivered/Failed event.

The event can be found through the Events API or through the UI in the expanded log entry under the Send->Logs section.

Sample code

Run the following python script with the storage key as a parameter. The script will retrieve the message from Mailgun. In the script the message is saved to "message.eml", which can then be opened in Mozilla Thunderbird for analysis.

```
"""View a message using its Mailgun storage key."""
import os
import sys

import requests

if len(sys.argv) != 2:
  print "Usage: retrieve.py message_key"
  sys.exit(1)

api_key = YOUR_API_KEY

# output filename
filename = "message.eml"

# url for retrieval
domain = "mailgun.com"
key = sys.argv[1]
url = "https://api.mailgun.net/v3/domains/%s/messages/%s"
url = url % (domain, key)

headers = {"Accept": "message/rfc2822"}

# request to API
r = requests.get(url, auth=("api", api_key), headers=headers)

if r.status_code == 200:
  with open(filename, "w") as message:
    message.write(r.json()["body-mime"])
  os.system("thunderbird -file %s" % filename)
else:
  print "Oops! Something went wrong: %s" % r.content
```