User Manual

Introduction

This document is meant to be an overview of all of the capabilities of Mailgun and how you can best leverage those capabilities. It is organized around the three major features that Mailgun provides:

- Sending Messages
- Tracking Messages
- Receiving, Forwarding and Storing Messages

At the heart of Mailgun is the API. Most of the Mailgun service can be accessed through the RESTful HTTP API without the need to install any libraries. However, we have written Libraries for many popular languages. Be sure to check out the additional capabilities provided by using our libraries.

You can also access many Mailgun features through your Mailgun Control Panel using your browser and logging in at https://mailgun.com/cp.

In addition to the API, Mailgun supports the standard SMTP protocol. We have included some instructions on how to use Mailgun, via SMTP, at the end of the User Manual.

If you are anxious to get started right away, feel free to check out the quickstart or API Reference. There are also FAQ and Email Best Practices that you can reference.

Finally, always feel free to contact our Support Team.

Getting Started

We've tried to make the sign-up and on-boarding process as intuitive as possible. However, there are a few things to mention.

Pricing & Features Overview

Pricing

Pricing is a usage-based, monthly subscription. Usage is based on outbound messages and number dedicated IP addresses used. There is no charge for inbound messages. As your volume increases your price per message decreases according to the pricing calculator on our pricing page.

If you are a high volume sender or if you are interested in a custom contract, you can contact sales@mailgunhq.com for more details.

Features

All of Mailgun's features are available to both free and paid accounts.

There are some limitations if you have not given us your payment information:

- There is a limit of 10,000 emails per month.
- Data for Logs and the Events API are stored for 2 days.

If you have given us your payment information, there is no limit on number of messages sent and/or received and data retention for Logs and the Events API is at least 30 days.

Verifying Your Domain

Each new Mailgun account is automatically provisioned with a **sandbox domain** sandbox<uniq-alpha-numeric-string>@mailgun.org. This domain is to be used for **testing only**. It allows both sending and receiving messages; and also tracking can be enabled for it. But it only allows sending to a list of up to 5 authorized recipients. This limitation is also in effect for routes that are triggered by messages addressed to the sandbox domain and mailing lists created under that domain.

To be able to use Mailgun in production a custom domain(s) has to be created and verified with Mailgun.

Verifying your domain is easy. Start by adding a domain or subdomain you own in the Domains tab of the Mailgun control panel. Next add two **TXT** DNS records found in the **Domain Verification & DNS** section on the domain information page of the Mailgun control panel at your DNS provider:

- SPF: Sending server IP validation. Used by majority of email service providers. Learn about SPF.
- DKIM: Like SPF, but uses cryptographic methods for validation. Supported by many email service providers. This is the record that Mailgun references make sure that the domain actually belongs to you. Learn about DKIM

Once you've added the two **TXT** records and they've propagated, your domain will be verified. In the Mailgun control panel verified domains are marked by a green Verified badge next to their name.

If it has been awhile since you have configured the DNS records but the domain is still reported as Unverified, then try pressing the **Check DNS Records Now** button on the domain information page. If that does not help either, then please create a support ticket.

Other DNS records

- **CNAME** DNS record with value *mailgun.org*, should be added if you want Mailgun to track **clicks**, **opens**, and **unsubscribes**.
- **MX** DNS records are required if you want Mailgun to receive and route/store messages addressed to the domain recipients. You need to configure 2 **MX** records with values 10 mxa.mailgun.org and 10 mxb.mailgun.org. We recommend adding them even if you do not plan the domain to get inbound messages, because having **MX** DNS records configured may improve deliverability of messages sent from the domain. Learn about MX DNS records

Warning

Do not configure **MX** DNS records if you already have another provider handling inbound mail delivery for the domain.

DNS Records Summary

Type	Required	Purpose	Value
TXT	Yes	Domain verification (SPF)	v=spf1 include:mailgun.org ~all
TXT	Yes	Domain verification (DKIM)	*Find this record in Domain Verification & DNS section in the domain information page for a particular domain in the Mailgun control pannel*
CNAME		Enables tracking	mailgun.org
MX		Enables receiving	10 mxa.mailgun.org
MX		Enables receiving	10 mxb.mailgun.org

Common DNS Provider Documentation

Common providers are listed below. If yours is not listed, contact your DNS provider for assistance:

- GoDaddy: MX - CNAME - TXT
- NameCheap: All Records
- Network Solutions: MX - CNAME - TXT
- Rackspace Email & Apps: All Records
- Rackspace Cloud DNS: Developer Guide
- Amazon Route 53: Developer Guide

Sending Messages

There are two ways to send messages using Mailgun:

- HTTP API
- SMTP

Both methods work great and support the same feature set, so choose one based on your preferences and requirements.

Sending via API

When sending via HTTP API, Mailgun offers two options:

- You can send emails in MIME format, but this would require you to use a MIME building library for your programming language.
- You can submit the individual parts of your messages to Mailgun, such as text and html parts, attachments, and so on. This doesn't require any MIME knowledge on your part.

Note

Mailgun supports maximum messages size of 25MB.

See sending messages section in our API Reference for a full list of message sending options.

Examples: sending messages via HTTP

Sending mails using Mailgun API is extremely simple: as simple as performing an HTTP POST request to an API URL.

Sending a plain text message:

```
curl -s --user 'api:YOUR_API_KEY' \
    https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/messages \
    -F from='Excited User <mailgun@YOUR_DOMAIN_NAME>' \
    -F to=YOU@YOUR_DOMAIN_NAME \
    -F to=bar@example.com \
    -F subject='Hello' \
    -F text='Testing some Mailgun awesomness!'
```

Sample response:

```
{
  "message": "Queued. Thank you.",
  "id": "<20111114174239.25659.5817@samples.mailgun.org>"
}
```

Sending a message with HTML and text parts. This example also attaches two files to the message:

```
curl -s --user 'api:YOUR_API_KEY' \
    https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/messages \
    -F from='Excited User <YOU@YOUR_DOMAIN_NAME>' \
```

```
    -F to='foo@example.com' \
    -F cc='bar@example.com' \
    -F bcc='baz@example.com' \
    -F subject='Hello' \
    -F text='Testing some Mailgun awesomness!' \
    --form-string html='<html>HTML version of the body</html>' \
    -F attachment=@files/cartman.jpg \
    -F attachment=@files/cartman.png
```

Sending a MIME message which you pre-build yourself using a MIME library of your choice:

```
curl -s --user 'api:YOUR_API_KEY' \
    https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/messages.mime \
    -F to='bob@example.com' \
    -F message=@files/message.mime
```

An example of how to toggle tracking on a per-message basis. Note the o:tracking option. This will disable link rewriting for this message:

```
curl -s --user 'api:YOUR_API_KEY' \
    https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/messages \
    -F from='Sender Bob <sbob@YOUR_DOMAIN_NAME>' \
    -F to='alice@example.com' \
    -F subject='Hello' \
    -F text='Testing some Mailgun awesomness!' \
    -F o:tracking=False
```

An example of how to set message delivery time using the o:deliverytime option:

```
curl -s --user 'api:YOUR_API_KEY' \
    https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/messages \
    -F from='Sender Bob <sbob@YOUR_DOMAIN_NAME>' \
    -F to='alice@example.com' \
    -F subject='Hello' \
    -F text='Testing some Mailgun awesomness!' \
    -F o:deliverytime='Fri, 14 Oct 2011 23:10:10 -0000'
```

An example of how to tag a message with the o:tag option:

```
curl -s --user 'api:YOUR_API_KEY' \
    https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/messages \
    -F from='Sender Bob <sbob@YOUR_DOMAIN_NAME>' \
    -F to='alice@example.com' \
    -F subject='Hello' \
    -F text='Testing some Mailgun awesomness!' \
    -F o:tag='September newsletter' \
    -F o:tag='newsletters'
```

An example of how to send a message with custom connection settings:

```
curl -s --user 'api:YOUR_API_KEY' \
    https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/messages \
    -F from='Sender Bob <sbob@YOUR_DOMAIN_NAME>' \
```

```
    -F to='alice@example.com' \
    -F subject='Hello' \
    -F text='Testing some Mailgun awesomness!' \
    -F o:require-tls=True \
    -F o:skip-verification=False
```

Sending Inline Images

Mailgun assigns content-id to each image passed via inline API parameter, so it can be referenced in HTML part.

Example of sending inline image. Note how image is referenced in HTML part simply by the filename:

```
curl -s --user 'api:YOUR_API_KEY' \
    https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/messages \
    -F from='Excited User <YOU@YOUR_DOMAIN_NAME>' \
    -F to='alice@example.com' \
    -F subject='Hello' \
    -F text='Testing some Mailgun awesomness!' \
    --form-string html='<html>Inline image here: <img
src="cid:cartman.jpg"></html>' \
    -F inline=@files/cartman.jpg
```

Sending via SMTP

Mailgun supports sending via SMTP. Our servers listen on ports 25, 465 (SSL/TLS), 587 (STARTTLS), and 2525.

Note

Some ISPs are blocking or throttling SMTP port 25. We recommend using #587 instead.

Note

Google Compute Engine allows port 2525 for SMTP submission.

Warning

IP addresses for HTTP and SMTP API endpoints will change frequently and subjected to change without notice. Ensure there are no IP-based ACLs that would prevent communication to new IP addresses that may be added or removed at any time.

Use “plain text” SMTP authentication and the credentials from the domain details page in your Control Panel which can be found by clicking on a domain in the Domains Tab. For enhanced security, use TLS encryption.

Note

See SMTP to learn how to configure the most popular SMTP software and email clients to work with Mailgun

Passing Sending Options

When sending a message via SMTP you can pass additional sending options via custom MIME headers listed in the table below.

Header	Description
X-Mailgun-Tag	Tag string used for aggregating stats. See Tagging for more information. You can mark a message with several categories by setting multiple X-Mailgun-Tag headers.
X-Mailgun-Campaign-Id	Id of the campaign the message belongs to. See um-campaign-analytics for details. You can assign a message to several campaigns by setting multiple different X-Mailgun-Campaign-Id headers.
X-Mailgun-Dkim	Enables/disables DKIM signatures on per-message basis. Use yes or no.
X-Mailgun-Deliver-By	Desired time of delivery. See Scheduling Delivery and Date Format.
X-Mailgun-Drop-Message	Enables sending in test mode. Pass yes if needed. See Sending in Test Mode.
X-Mailgun-Track	Toggles tracking on a per-message basis, see Tracking Messages for details. Pass yes or no.
X-Mailgun-Track-Clicks	Toggles clicks tracking on a per-message basis. Has higher priority than domain-level setting. Pass yes, no or htmlonly.
X-Mailgun-Track-Opens	Toggles opens tracking on a per-message basis. Has higher priority than domain-level setting. Pass yes or no.
X-Mailgun-Require-TLS	Use this header to control TLS connection settings. See TLS Sending Connection Settings
X-Mailgun-Skip-Verification	Use this header to control TLS connection settings. See TLS Sending Connection Settings
X-Mailgun-Recipient-Variables	Use this header to substitute recipient variables referenced in a batched mail message. See Batch Sending
X-Mailgun-Variables	Use this header to attach a custom JSON data to the message. See Attaching Data to Messages for more information.

Message Queue

When you submit messages for delivery Mailgun places them in a message queue.

- You can submit a large amount of messages and Mailgun will automatically queue the delivery in compliance with the receiving domains' guidelines and maximum allowed sending rate optimized for each ESP (email service provider) such as Yahoo, GMail, etc.
- The Queue is dynamic so as you send more messages, your sending rates will increase, assuming you are sending quality traffic. (See Email Best Practices about warming up IP addresses.) Do not get discouraged if your messages take longer to be delivered at the beginning. As your reputation grows, your sending rate will grow too.

The queueing algorithms are one of the most important features of Mailgun. If you try to send bulk mailings all at once, most ISPs will block you, or worse, just drop your messages without telling you. In addition, it is important to gradually increase your sending rates according to many factors, including consistency of traffic, IP address sending history, and domain reputation.

Batch Sending

Mailgun supports the ability send to a group of recipients through a single API call or SMTP session. This is achieved by either:

- Using Batch Sending by specifying multiple recipient email addresses as to parameters and using Recipient Variables.
- Using Mailing Lists with Template Variables.

Warning

It is important when using Batch Sending to also use Recipient Variables. This tells Mailgun to send each recipient an individual email with only their email in the to field. If they are not used, all recipients' email addresses will show up in the to field for each recipient.

Recipient Variables

Recipient Variables are custom variables that you define, which you can then reference in the message body. They give you the ability to send a custom message to each recipient while still using a single API Call (or SMTP session).

To access a recipient variable within your email, simply reference %recipient.yourkey%. For example, consider the following JSON:

```
{
  "user1@example.com" : {"unique_id": "ABC123456789"},
  "user2@example.com" : {"unique_id": "ZXY987654321"}
}
```

To reference the above variables within your email, use %recipient.unique_id%.

Recipient Variables allow you to:

- Submit a message template;

- Include multiple recipients; and
- Include a set of key:value pairs with unique data for each recipient.

```
curl -s --user 'api:YOUR_API_KEY' \
    https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/messages \
    -F from='Excited User <YOU@YOUR_DOMAIN_NAME>' \
    -F to=alice@example.com \
    -F to=bob@example.com \
    -F recipient-variables='{"bob@example.com": {"first":"Bob", "id":1},
"alice@example.com": {"first":"Alice", "id": 2}}' \
    -F subject='Hey, %recipient.first%' \
    -F text='If you wish to unsubscribe, click
http://mailgun/unsubscribe/%recipient.id%'
```

Note

The maximum number of recipients allowed for Batch Sending is 1,000.

Note

Recipient variables should be set as a valid JSON-encoded dictionary, where key is a plain recipient address and value is a dictionary with variables.

In the example above, Alice and Bob both will get personalized subject lines “Hey, Alice” and “Hey, Bob” and unique unsubscribe links.

When sent via SMTP, recipient variables can be included by adding the following header to your email, “X-Mailgun-Recipient-Variables: {“my_message_id”: 123}”.

Example:

```
X-Mailgun-Recipient-Variables: {"bob@example.com": {"first":"Bob", "id":1},
"alice@example.com": {"first":"Alice", "id": 2}}
From: me@example.com
To: alice@example.com, bob@example.com
Date: 29 Mar 2016 00:23:35 -0700
Subject: Hello, %recipient.first%!
Message-Id: <20160329071939.35138.9413.6915422C@example.com>
Content-Type: text/plain; charset="us-ascii"
Content-Transfer-Encoding: quoted-printable

Hi, %recipient.first%,
=20
Please review your profile at example.com/orders/%recipient.id%.
=20
Thanks,
Example.com Team
```

Note

The value of the “X-Mailgun-Recipient-Variables” header should be valid JSON string, otherwise Mailgun won’t be able to parse it. If your “X-Mailgun-Recipient-Variables” header exceeds 998 characters, you should use folding to spread the variables over multiple lines.

They can also supplied through a special construct, called a variables container.

To contain variables you create the following MIME construct:

```
multipart/mailgun-variables
--application/json (base64 encoded)
--message/rfc822
----original-message
```

In this construct, JSON will be Base64 encoded and will be stored inside the part body, which will handle recipient variables containing special characters.

Example:

```
Content-Type: multipart/mailgun-variables;
boundary="8686cc907910484e9d21c54776cd791c"
Mime-Version: 1.0
From: bob@bob-mg
Date: Thu, 26 Jul 2012 15:43:07 +0000
Message-Id: <20120726154307.29852.44460@definebox.com>
Sender: bob=bob-mg@definebox.com

--8686cc907910484e9d21c54776cd791c
Mime-Version: 1.0
Content-Type: application/json
Content-Transfer-Encoding: base64

eyJkZXNjcmlwdGlvbiI6ICJrbGl6aGVudGFzIn0=

--8686cc907910484e9d21c54776cd791c
Content-Type: message/rfc822
Mime-Version: 1.0

Date: Thu, 26 Jul 2012 19:42:55 +0400
To: %recipient.description% <support@mailgunhq.com>
From: bob@bob-mg
Subject: (rackspace) Hello
 MSK 2012 support@mailgunhq.com %recipient.description%
Message-Id: <20120726154302.29322.40670@definebox.com>

support@mailgunhq.com %recipient.description%

--8686cc907910484e9d21c54776cd791c--
```

Mailing Lists

Mailing Lists provide a convenient way to send to multiple recipients by using an alias email address. Mailgun sends a copy of the message sent to the alias address to each subscribed

member of the Mailing List. You can create and maintain your subscriber lists using the API or Control Panel. In addition, you can use Template Variables to create a unique message for each member of the Mailing List.

Overview

To use Mailing Lists you create a Mailing List address, like devs@example.com and add member addresses to it. Each time you send a message to devs@example.com, a copy of it is delivered to each subscribed member.

Managing a list

You can create Mailing Lists using the Mailing List tab in the Control Panel or through the API. We support a couple of formats to make your life easier: you can upload a CSV file with members, use JSON or use form-like file upload.

Creating a mailing list through the API:

```
curl -s --user 'api:YOUR_API_KEY' \
    https://api.mailgun.net/v3/lists \
    -F address='LIST@YOUR_DOMAIN_NAME' \
    -F description='Mailgun developers list'
```

Adding a member through the API:

```
curl -s --user 'api:YOUR_API_KEY' \
    https://api.mailgun.net/v3/lists/LIST@YOUR_DOMAIN_NAME/members \
    -F subscribed=True \
    -F address='bar@example.com' \
    -F name='Bob Bar' \
    -F description='Developer' \
    -F vars='{"age": 26}'
```

Note

You can attach a JSON dictionary with the structured data to each member of the mailing list and reference that data in the message body using Template Variables (see vars parameter in the example above).

Note

There are two modes available when adding a new member: strict and upsert. Strict will raise an error in case the member already exists, upsert will update an existing member if it’s here or insert a new one. Learn how to toggle between the the modes and skip malformed addresses in the Mailing Lists API section.

Sending to a list

You can set the access level of Mailing Lists to:

- Only allow the administrator to post to the list (limited to an API call or authenticated SMTP session);
- Allow Mailing List members to post to the list; or
- Allow anybody to post to the list.

Campaigns

Mailing lists are integrated with um-campaign-analytics. Each message sent to a list with a Campaign ID will be tracked and reported. In this case, the mailing list will have detailed analytics for all recipients that can be retrieved via API or seen in the Campaign tab of the Control Panel.

Template Variables

There are some pre-defined variables you can use to personalize your message to each recipient. When adding members to a Mailing List you can also define your own variables in addition to these pre-defined variables by using the vars parameter.

Variable	Description
%recipient%	Full recipient spec, like “Bob <bob@example.com>” (for using as value for “To” MIME header).
%recipient_email%	Recipient’s email address, like bob@example.com.
%recipient_name%	Recipient’s full name, like “John Q. Public”.
%recipient_fname%	Recipient’s first name.
%recipient_lname%	Recipient’s last name.
%unsubscribe_url%	A generated URL which allows users to unsubscribe from messages.
%mailing_list_unsubscribe_url%	A generated URL which allows users to unsubscribe from mailing lists.
%unsubscribe_email%	An email address which can be used for automatic unsubscription by adding it to List-Unsubscribe MIME header.
%recipient.yourvar%	Accessing a custom datavalue. (see Attaching Data to Messages)

Unsubscribing

For managing unsubscribes in Mailing Lists, you can use %mailing_list_unsubscribe_url%. We will generate the unique link to unsubscribe from the mailing list. Once a recipient clicks on the unsubscribe link, we mark the recipient as “unsubscribed” from this list and they won’t get any further emails addressed to this list. Note, that you can still override the “unsubscribe” setting via

the API or the Control Panel (in case of user error or accidental unsubscribe, for example). You can also manually unsubscribe the customer without using any links via the API or in the Control Panel. Read more in the Mailing Lists API section.

Mailing Lists and Routes

Mailing Lists work independently from Routes. If there is a Mailing List or Route with the same address, the incoming message will hit the Route and Mailing List simultaneously. This can be pretty convenient for processing replies to the Mailing List and integrating into things like forums or commenting systems.

Scheduling Delivery

Mailgun also allows you to request a specific time for your message delivery by using the o:deliverytime parameter if sending via the API, or X-Mailgun-Deliver-By MIME header if sending via SMTP.

While messages are not guaranteed to arrive at exactly the requested time due to the dynamic nature of the queue, Mailgun will do its best.

Note

Messages can be scheduled for a maximum of 3 days in the future.

Scheduling Delivery API Example

Supply **RFC 2822#section-3.3** or Unix epoch time to schedule your message:

```
curl -s --user 'api:YOUR_API_KEY' \
    https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/messages \
    -F from='Sender Bob <sbob@YOUR_DOMAIN_NAME>' \
    -F to='alice@example.com' \
    -F subject='Hello' \
    -F text='Testing some Mailgun awesomness!' \
    -F o:deliverytime='Fri, 14 Oct 2011 23:10:10 -0000'
```

Sending in Test Mode

You can send messages in test mode by setting o:testmode parameter to true. When you do this, Mailgun will accept the message but will not send it. This is useful for testing purposes.

Note

You are charged for messages sent in test mode.

Tracking Messages

Once you start sending and receiving messages, it's important to track what's happening with them. We try to make tracking your messages as easy as possible through Events, Stats and Campaigns.

In addition, Mailgun permanently stores when a message can not be delivered due to a hard bounce (permanent failure) or when a recipient unsubscribes or complains of spam. In these cases, Mailgun will not attempt to deliver to these recipients in the future, in order to protect your sending reputation.

Mailgun provides a variety of methods to access data on your emails:

- View and search Events through the Logs tab in the Control Panel to see every event that has happened to every message. You can search by fields like recipient, subject line and even fields that don't show up in the Logs, like message-id. Data is stored for at least 30 days for paid accounts and at least 2 days for free accounts.
- Access data on Events programmatically through the Events API. Data is stored for at least 30 days for paid accounts and at least 2 days for free accounts.
- View, search and edit tables for Bounces, Unsubscribes and Spam Complaints in Suppression Lists or their respective APIs (Bounces API, Unsubscribes API, Complaints API). Data is stored indefinitely.
- Access statistics aggregated by tags in the Tracking tab of the Control Panel or the Stats API. Data is stored for at least 6 months.
- Create Campaigns and access detailed analytics on those Campaigns through the Control Panel or the Campaigns API. Data is stored for at least 6 months other than the delivered event which is stored for 2 weeks.
- Receive notifications of events through a Webhook each time an Event happens and store the data on your side.

Enable Tracking

Event tracking is automatically enabled except for Unsubscribes, Opens and Clicks.

You can enable Unsubscribes tracking for your domain via the "Domains" tab of the Control Panel. You can also manage unsubscribes per message by using unsubscribe variables (see Tracking Unsubscribes)

You can enable Opens & Clicks tracking on two levels: per sending domain and per message.

- You can enable Open & Click tracking on per domain basis under the "Domain Settings" subsection on the domain info page.
- Tracking can also be toggled by setting o:tracking, o:tracking-clicks and o:tracking-opens parameters when sending your message. This will override the domain-level setting.

Note

You will also have to point CNAME records to mailgun.org for Mailgun to rewrite links and track opens. In addition, there needs to be an html part of message for Mailgun to track opens (see Tracking Opens and Tracking Clicks for more detail).

Events

Mailgun keeps track of every event that happens to every message (both inbound and outbound) and stores this data for at least 30 days for paid accounts and 2 days for free accounts.

Below is the table of events that Mailgun tracks.

Event	Description
accepted	Mailgun accepted the request to send/forward the email and the message has been placed in queue.
rejected	Mailgun rejected the request to send/forward the email.
delivered	Mailgun sent the email and it was accepted by the recipient email server.
failed	Mailgun could not deliver the email to the recipient email server.
opened	The email recipient opened the email and enabled image viewing. Open tracking must be enabled in the Mailgun control panel, and the CNAME record must be pointing to mailgun.org.
clicked	The email recipient clicked on a link in the email. Click tracking must be enabled in the Mailgun control panel, and the CNAME record must be pointing to mailgun.org.
unsubscribed	The email recipient clicked on the unsubscribe link. Unsubscribe tracking must be enabled in the Mailgun control panel.
complained	The email recipient clicked on the spam complaint button within their email client. Feedback loops enable the notification to be received by Mailgun.
stored	Mailgun has stored an incoming message

You can access Events through a few interfaces:

- Webhooks (we POST data to your URL).
- The Events API (you GET data through the API).
- The Logs Tab of the Control Panel (GUI).

Events API

You can programmatically query and download events through the Events API.

```
curl -s --user 'api:YOUR_API_KEY' -G \
      https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/events \
      --data-urlencode begin='Fri, 3 May 2013 09:00:00 -0000' \
      --data-urlencode ascending=yes \
      --data-urlencode limit=25 \
```

```
      --data-urlencode pretty=yes \
      --data-urlencode recipient=joe@example.com
```

Sample response:


```
{
  "items": [
    {
      "tags": [],
      "timestamp": 1376325780.160809,
      "envelope": {
        "sender": "me@samples.mailgun.org",
        "transport": ""
      },
      "event": "accepted",
      "campaigns": [],
      "user-variables": {},
      "flags": {
        "is-authenticated": true,
        "is-test-mode": false
      },
      "message": {
        "headers": {
          "to": "user@example.com",
          "message-id": "20130812164300.28108.52546@samples.mailgun.org",
          "from": "Excited User <me@samples.mailgun.org>",
          "subject": "Hello"
        },
        "attachments": [],
        "recipients": [
          "user@example.com"
        ],
        "size": 69
      },
      "recipient": "user@example.com",
      "method": "http"
    }
  ],
  "paging": {
    "next":
        "https://api.mailgun.net/v3/samples.mailgun.org/events/W3siY...",
    "previous":
        "https://api.mailgun.net/v3/samples.mailgun.org/events/Lkawm..."
  }
}
```


Webhooks

Mailgun can make an HTTP POST to your URLs when events occur with your messages. If you would like Mailgun to POST event notifications, you need to provide a callback URL in the Webhooks tab of the Control Panel. Webhooks are at the domain level so you can provide a unique URL for each domain by using the domain drop down selector.

You can read more about the data that is posted in the appropriate section below (Tracking Opens, Tracking Clicks, Tracking Unsubscribes, Tracking Spam Complaints, Tracking Bounces, Tracking Failures, Tracking Deliveries). We recommend using http://bin.mailgun.net/ for creating temporary URLs to test and debug your webhooks.

For Webhook POSTs, Mailgun listens for the following codes from your server and reacts accordingly:

- If Mailgun receives a 200 (Success) code it will determine the webhook POST is successful and not retry.
- If Mailgun receives a 406 (Not Acceptable) code, Mailgun will determine the POST is rejected and not retry.
- For any other code, Mailgun will retry POSTing according to the schedule below for Webhooks other than the delivery notification.

If your application is unable to process the webhook request but you do not return a 406 error code, Mailgun will retry (other than for delivery notification) during 8 hours at the following intervals before stop trying: 10 minutes, 10 minutes, 15 minutes, 30 minutes, 1 hour, 2 hour and 4 hours.

The Webhooks API endpoint allows you to programmatically manipulate the webhook URLs defined for a specific domain. Head over to the Webhooks API endpoint documentation.

Securing Webhooks

To ensure the authenticity of event requests, Mailgun signs them and posts the signature along with other webhook parameters:

Parameter	Type	Description
timestamp	int	Number of seconds passed since January 1, 1970.
token	string	Randomly generated string with length 50.
signature	string	String with hexadecimal digits generate by HMAC algorithm.

To verify the webhook is originating from Mailgun you need to:

- Concatenate timestamp and token values.
- Encode the resulting string with the HMAC algorithm (using your API Key as a key and SHA256 digest mode).
- Compare the resulting hexdigest to the signature.
- Optionally, you can cache the token value locally and not honor any subsequent request with the same token. This will prevent replay attacks.
- Optionally, you can check if the timestamp is not too far from the current time.

Note

Due to potentially large size of posted data, Mailgun computes an authentication signature based on a limited set of HTTP headers.

Below is a Python code sample used to verify the signature:

```
import hashlib, hmac

def verify(api_key, token, timestamp, signature):
    hmac_digest = hmac.new(key=api_key,
                           msg='{}{}'.format(timestamp, token),
                           digestmod=hashlib.sha256).hexdigest()
    return hmac.compare_digest(unicode(signature), unicode(hmac_digest))
```

And here's a sample in Ruby:

```
require 'openssl'

def verify(api_key, token, timestamp, signature)
  digest = OpenSSL::Digest::SHA256.new
  data = [timestamp, token].join
  signature == OpenSSL::HMAC.hexdigest(digest, api_key, data)
end
```

And here's a sample in PHP:

Attaching Data to Messages

When sending, you can attach data to your messages by passing custom data to the API or SMTP endpoints. The data will be represented as a header within the email, X-Mailgun-Variables. The data is formatted in JSON and included in any webhook events related to the email containing the custom data. Several such headers may be included and their values will be combined.

Example:

```
X-Mailgun-Variables: {"first_name": "John", "last_name": "Smith"}
X-Mailgun-Variables: {"my_message_id": 123}
```

To add this header to your message:

API: Pass the following parameter, "v:my-custom-data" => "{"my_message_id": 123}".

SMTP: Add the following header to your email, "X-Mailgun-Variables: {"my_message_id": 123}".

You can also use values from your recipient variables to provide a custom variable per a recipient using templating. For example when sending via the API:

```
{'v:Recipient-Id': '%recipient.id%'}
```

Note

The value of the “X-Mailgun-Variables” header should be valid JSON string, otherwise Mailgun won’t be able to parse it. If your X-Mailgun-Variables header exceeds 998 characters, you should use folding to spread the variables over multiple lines.

Tagging

Sometimes it’s helpful to categorize your outgoing email traffic based on some criteria, perhaps separate signup emails from password recovery emails or from user comments. Mailgun lets you tag each outgoing message with a custom value. When you access stats on you messages, they will be aggregated by these tags.

Tagging Code Samples

Supply one or more o:tag parameters to tag the message.

```
curl -s --user 'api:YOUR_API_KEY' \
    https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/messages \
    -F from='Sender Bob <sbob@YOUR_DOMAIN_NAME>' \
    -F to='alice@example.com' \
    -F subject='Hello' \
    -F text='Testing some Mailgun awesomness!' \
    -F o:tag='September newsletter' \
    -F o:tag='newsletters'
```

Note

A single message may be marked with up to 3 tags.

Note

Tags are case insensitive and should be ascii only. Maximum tag length is 128 characters.

Tracking Opens

Mailgun can keep track of every time a recipient opens your messages. You can see when Opens happen in the Logs tab or see aggregate counters of opens in the Tracking tab of the Control Panel. In addition, you can be notified through a webhook or get the data programmatically through the Events API.

You can enable Open tracking by clicking on the checkbox in the Tracking tab of your Control Panel or using the o:tracking or o:tracking-opens parameters when sending a message. You will also have to add the appropriate CNAME records to your DNS as specified in the ‘Domain’ tab of your Control Panel.

Opens are tracked by including a transparent .png file, which will only work if there is an HTML component to the email (i.e., text only emails will not track opens). You should note that many email service providers disable images by default, so this data will only show up if the recipient clicks on display images button in his/her email.

Note

Return Path certification allows your images to be enabled by default at many ISPs. Please contact us if you would like to get your IP Address certified.

Opens Webhook

You can specify a webhook URL in the 'Webhooks' tab of your Control Panel. When a user opens one of your emails, your URL will be called with the following parameters.

Parameter Name	Description
event	Event name ("opened").
recipient	Recipient who opened.
domain	Domain that sent the original message.
ip	IP address the event originated from.
country	Two-letter country code (as specified by ISO3166) the event came from or 'unknown' if it couldn't be determined.
region	Two-letter or two-digit region code or 'unknown' if it couldn't be determined.
city	Name of the city the event came from or 'unknown' if it couldn't be determined.
user-agent	User agent string of the client triggered the event.
device-type	Device type the email was opened on. Can be 'desktop', 'mobile', 'tablet', 'other' or 'unknown'.
client-type	Type of software the email was opened in, e.g. 'browser', 'mobile browser', 'email client'.
client-name	Name of the client software, e.g. 'Thunderbird', 'Chrome', 'Firefox'.
client-os	OS family running the client software, e.g. 'Linux', 'Windows', 'OSX'.
campaign-id	The id of campaign triggering the event.
campaign-name	The name of campaign triggering the event.
tag	Message tag, if message was tagged. See Tagging
mailing-list	The address of mailing list the original message was sent to.
"custom variables"	Your own custom JSON object included in the header (see Attaching Data to Messages).
timestamp	Number of seconds passed since January 1, 1970 (see securing webhooks).

Parameter Name	Description
token	Randomly generated string with length 50 (see securing webhooks).
signature	String with hexadecimal digits generate by HMAC algorithm (see securing webhooks).

Tracking Clicks

Mailgun can keep track of every time a recipient clicks on links in your messages. You can see when clicks happen in the Logs tab or see aggregate counters of clicks in the Tracking tab of the Control Panel. In addition, you can be notified through a webhook or get the data programmatically through the Events API.

You can enable click tracking by clicking on the checkbox in the Tracking tab of your Control Panel or using the o:tracking or o:tracking-clicks parameters when sending a message. You will also have to add the appropriate CNAME records to your DNS as specified in the Domains tab of your Control Panel. If you enable Click tracking, links will be overwritten and pointed to our servers so we can track clicks. You can specify that you only want links rewritten in the HTML part of a message with the parameter o:tracking-clicks and passing htmlonly.

Clicks Webhook

You can specify a webhook URL in the 'Webhooks' tab of your Control Panel. Every time a user clicks on a link inside of your messages, your URL will be called with the following parameters:

Parameter Name	Description
event	Event name ("clicked").
recipient	Recipient who clicked.
domain	Domain that sent the original message.
ip	IP address the event originated from.
country	Two-letter country code (as specified by ISO3166) the event came from or 'unknown' if it couldn't be determined.
region	Two-letter or two-digit region code or 'unknown' if it couldn't be determined.
city	Name of the city the event came from or 'unknown' if it couldn't be determined.
user-agent	User agent string of the client triggered the event.
device-type	Device type the link was clicked on. Can be 'desktop', 'mobile', 'tablet', 'other' or 'unknown'.
client-type	Type of software the link was opened in, e.g. 'browser', 'mobile browser', 'email client'.

Parameter Name	Description
client-name	Name of the client software, e.g. ‘Thunderbird’, ‘Chrome’, ‘Firefox’.
client-os	OS family running the client software, e.g. ‘Linux’, ‘Windows’, ‘OSX’.
campaign-id	The id of campaign triggering the event.
campaign-name	The name of campaign triggering the event.
tag	Message tag, if it was tagged. See Tagging.
url	The URL that was clicked.
mailing-list	The address of mailing list the original message was sent to.
“custom variables”	Your own custom JSON object included in the header (see Attaching Data to Messages).
timestamp	Number of seconds passed since January 1, 1970 (see securing webhooks).
token	Randomly generated string with length 50 (see securing webhooks).
signature	String with hexadecimal digits generate by HMAC algorithm (see securing webhooks).

Tracking Unsubscribes

Mailgun can keep track of every time a recipient requests to be unsubscribed from your mailings. If you enable unsubscribe tracking, Mailgun will insert unsubscribe links and remove those recipients from your mailings automatically for you.

You can see when usubscribes happen in the Logs tab or see aggregate counters of unsubscribes in the Tracking tab of the Control Panel. In addition, you can be notified through a webhook or get the data programmatically through the Events API or the Bounces API.

Mailgun supports three types of unsubscribes: domain, tag or Mailing Lists levels.

- Domain level: Once recipient selects to unsubscribe from domain, he will not receive any more messages from this sending domain.
- Tag level: Sometimes you need to separate traffic by types, for example provide newsletter mailings, security updates mailings and so on. Recipients may want to unsubscribe from your newsletters but still receive security updates. For this purpose you can use tags: mark your messages by setting approriate X-Mailgun-Tag header and use special %tag_unsubscribe_url% variable (see below).
- Mailing Lists level: If a recipient unsubscribes from a Mailing List, they will still be a member of the Mailing List but will be flagged as unsubscribed and Mailgun will no longer send messages from that Mailing List to the unsubscribed recipient.

Auto-Handling

You can enable Mailgun's Unsubscribe functionality by turning it on in the settings area for your domain. We will automatically prevent future emails being sent to recipients that have unsubscribed. You can edit the unsubscribed address list from your Control Panel or through the API.

Note

Before enabling, you will need to configure the required DNS entries provided in your Control Panel.

Mailgun provides you with several unsubscribe variables:

Variable	Description
%unsubscribe_url%	link to unsubscribe recipient from all messages sent by given domain
%tag_unsubscribe_url%	link to unsubscribe from all tags provided in the message
%mailing_list_unsubscribe_url%	link to unsubscribe from future messages sent to a mailing list

If you include these variables in your emails, any recipient that clicks on the url will be automatically unsubscribed and those email addresses will be blocked from receiving future emails from that domain or message tag as appropriate.

Mailgun can automatically provide an unsubscribe footer in each email you send. You can customize your unsubscribe footer by editing the settings in the control panel.

To enable/disable unsubscribes programmaticaly per message you can do the following:

- Enable unsubscription feature for your domain.
- Remove text in the html and text footer templates so they won't be appended automatically.
- Insert a variable in the html and text bodies of your email when you need unsubscribe links.
- This variable will be replaced by the corresponding unsubscribe link.

In the "Suppressions" tab of the Control Panel or through the API you can also:

- View/get a list of unsubscribed addresses.
- Remove an unsubscribed address from the list.
- Add a new unsubscribed address.

Take a look at Unsubscribes section of the API reference to learn how to programmatically manage lists of unsubscribed users.

Unsubscribes Webhook

You can specify a webhook URL in the 'Webhooks' tab of your Control Panel. When a user unsubscribes, Mailgun will invoke the webhook with the following parameters:

Parameter Name	Description
event	Event name ("unsubscribed").
recipient	Recipient who unsubscribed.
domain	Domain that sent the original message.
ip	IP address the event originated from.
country	Two-letter country code (as specified by ISO3166) the event came from or 'unknown' if it couldn't be determined.
region	Two-letter or two-digit region code or 'unknown' if it couldn't be determined.
city	Name of the city the event came from or 'unknown' if it couldn't be determined.
user-agent	User agent string of the client triggered the event.
device-type	Device type the person unsubscribed on. Can be 'desktop', 'mobile', 'tablet', 'other' or 'unknown'.
client-type	Type of software the unsubscribe link was clicked in, e.g. 'browser', 'mobile browser', 'email client'.
client-name	Name of the client software, e.g. 'Thunderbird', 'Chrome', 'Firefox'.
client-os	OS family running the client software, e.g. 'Linux', 'Windows', 'OSX'.
campaign-id	The id of the campaign that recipient has unsubscribed from.
campaign-name	The name of campaign triggering the event.
tag	Message tag, if it was tagged. See Tagging.
mailing-list	The address of mailing list the original message was sent to.
"custom variables"	Your own custom JSON object included in the header (see Attaching Data to Messages).
timestamp	Number of seconds passed since January 1, 1970 (see securing webhooks).
token	Randomly generated string with length 50 (see securing webhooks).
signature	String with hexadecimal digits generate by HMAC algorithm (see securing webhooks).

Tracking Spam Complaints

Mailgun automatically keeps track of every time a recipient complains that a message is spam.

You can see when complaints happen in the Logs tab or see aggregate counters of complaints in the Tracking tab of the Control Panel. In addition, you can be notified through a webhook or get the data programmatically through the Events API or the Complaints API.

Email service providers (“ESPs”) are very sensitive to users clicking on spam complaint buttons and it’s important to monitor that activity to maintain a good sending reputation. While, not every ESP supports Feedback Loop (“FBL”) notifications, we make sure that you get data on all of the ones that do. We will remove recipients from future messages if a complaint has been filed by that recipient. This is necessary to maintain your reputation and not have your emails automatically sent to spam folders.

Spam Complaint tracking is always enabled.

Mailgun provides Spam complaints API to programmatically manage the lists of users who have complained.

Spam Complaints Webhook

You can specify a webhook URL in the ‘Webhooks’ tab in the control panel. When a user reports one of your emails as spam, Mailgun will invoke the webhook with the following parameters:

Parameter Name	Description
event	Event name (“complained”).
recipient	Recipient who clicked spam.
domain	Domain that sent the original message.
message-headers	String list of all MIME headers of the original message dumped to a JSON string (order of headers preserved).
campaign-id	The id of campaign triggering the event.
campaign-name	The name of campaign triggering the event.
tag	Message tag, if it was tagged. See Tagging.
mailing-list	The address of mailing list the original message was sent to.
“custom variables”	Your own custom JSON object included in the header (see Attaching Data to Messages).
timestamp	Number of seconds passed since January 1, 1970 (see securing webhooks).
token	Randomly generated string with length 50 (see securing webhooks).
signature	String with hexadecimal digits generate by HMAC algorithm (see securing webhooks).
attachment-x	attached file (‘x’ stands for number of the attachment). Attachments are included if the recipient ESP includes them in the bounce message. They are handled as file uploads, encoded as multipart/form-data.

Tracking Bounces

An email message is said to “bounce” if it is rejected by the recipient SMTP server.

With respect to failure persistence Mailgun classifies bounces into the following two groups:

- Hard bounces (permanent failure): Recipient is not found and the recipient email server specifies the recipient does not exist. Mailgun stops attempting delivery to invalid recipients after one Hard Bounce. These addresses are added to the table in the Bounces tab and Mailgun will not attempt delivery in the future.
- Soft bounces (temporary failure): Email is not delivered because the mailbox is full or for other reasons. These addresses are not added to the table in the Bounces tab.

With respect to when the recipient SMTP server rejected an incoming message Mailgun classifies bounces into the following two groups:

- Immediate bounce: An email message is rejected by the recipient SMTP server during the SMTP session.
- Delayed (asynchronous) bounce: The recipient SMTP server accepts an email message during the SMTP session. After some time it will then send a Non-Delivery Report email message to the message sender.

Note

In the case of a bounce Mailgun will retry to deliver the message only if the bounce was both Immediate and Soft. After several unsuccessful attempts Mailgun will quit retrying in order to maintain your sending reputation.

Warning

Mailgun can track delayed bounces but only if the domain, that the email message was sent from, has MX records pointing to Mailgun. Otherwise NDR email messages won’t reach Mailgun. Please refer to Verifying Your Domain for details on how to do that.

You can see when bounces happen in the Logs tab or see aggregate counters of bounces in the Tracking tab of the Control Panel. In addition, you can be notified through a webhook or get the data programmatically through the Events API or the Bounces API.

Mailgun provides Bounces API to programmatically manage the lists of hard bounces.

Bounce Event Webhook

You can specify a webhook URL in the ‘Webhooks’ tab of your Control Panel. If you do, every time a message experiences a hard bounce, your URL will be invoked with the following parameters:

Parameter Name	Description
event	Event name (“bounced”).
recipient	Recipient who could not be reached.
domain	Domain that sent the original message.
message-headers	String list of all MIME headers of the original message dumped to a JSON string (order of headers preserved).
code	SMTP bounce error code in form (X.X.X).
error	SMTP bounce error string.
notification	Detailed reason for bouncing (optional).
campaign-id	The id of campaign triggering the event.
campaign-name	The name of campaign triggering the event.
tag	Message tag, if it was tagged. See Tagging.
mailing-list	The address of mailing list the original message was sent to.
“custom variables”	Your own custom JSON object included in the header (see Attaching Data to Messages).
timestamp	Number of seconds passed since January 1, 1970 (see securing webhooks).
token	Randomly generated string with length 50 (see securing webhooks).
signature	String with hexadecimal digits generate by HMAC algorithm (see securing webhooks).
attachment-x	attached file (‘x’ stands for number of the attachment). Attachments are included if the recipient ESP includes them in the bounce message. They are handled as file uploads, encoded as multipart/form-data.

Tracking Failures

Mailgun tracks all delivery failures. Failures consist of both Hard Bounces (permanent failures) and Soft Bounces (temporary failures).

You can see when failures happen in the Logs tab. In addition, you can be notified through a webhook when a message is dropped (i.e., stop retries) or get the data programmatically through the Events API.

Drop Event Webhook

In the Webhooks tab, you can specify a URL to be notified every time a message is dropped. There are a few reasons why Mailgun needs to stop attempting to deliver messages and drop them. The most common reason is that Mailgun received a Hard bounce or repeatedly received Soft bounces and continuing attempting to deliver may hurt your reputation with the receiving ESP. Also, if the address is on one of the ‘do not send lists’ because that recipient had previously

bounced, unsubscribed, or complained of spam, we will not attempt delivery and drop the message. If one of these events occur we will POST the following parameters to your URL:

Parameter Name	Description
event	Event name (“dropped”).
recipient	Intended recipient.
domain	Domain that sent the original message.
message-headers	String list of all MIME headers of the original message dumped to a JSON string (order of headers preserved).
reason	Reason for failure. Can be one either “hardfail” or “old”. See below.
code	ESP response code, e.g. if the message was blocked as a spam (optional).
description	Detailed explanation of why the messages was dropped
“custom variables”	Your own custom JSON object included in the header (see Attaching Data to Messages).
timestamp	Number of seconds passed since January 1, 1970 (see securing webhooks).
token	Randomly generated string with length 50 (see securing webhooks).
signature	String with hexadecimal digits generate by HMAC algorithm (see securing webhooks).
attachment-x	attached file (‘x’ stands for number of the attachment). Attachments are included if the recipient ESP includes them in the bounce message. They are handled as file uploads, encoded as multipart/form-data.

- old indicates that Mailgun tried to deliver the message unsuccessfully for more than 8 hours.
- hardfail not delivering to an address that previously bounced, unsubscribed, or complained.

Tracking Deliveries

Mailgun tracks all successful deliveries of messages. A successful delivery occurs when the recipient email server responds that it has accepted the message.

You can see when deliveries happen in the Logs tab. In addition, you can be notified through a webhook when a message is delivered or get the data programmatically through the Events API.

Delivered Event Webhook

In the Webhooks tab, you can specify a URL to be notified every time a message is delivered. If the message is successfully delivered to the intended recipient, we will POST the following parameters to your URL:

Parameter Name	Description
event	Event name (“delivered”).
recipient	Intended recipient.
domain	Domain that sent the original message.
message-headers	String list of all MIME headers dumped to a JSON string (order of headers preserved).
Message-Id	String id of the original message delivered to the recipient.
“custom variables”	Your own custom JSON object included in the header of the original message (see Attaching Data to Messages).
timestamp	Number of seconds passed since January 1, 1970 (see securing webhooks).
token	Randomly generated string with length 50 (see securing webhooks).
signature	String with hexadecimal digits generate by HMAC algorithm (see securing webhooks).

Note

Unlike other event webhooks (due to frequency of delivered events), Delivered Event will only POST once, right after delivery, and won’t attempt again in case of failure to POST successfully.

Stats

Stats provide you with the summary of the events that occur with your messages and can be aggregated by tag, see Tagging above.

You can see your current statistics in the control panel, or download them using the API

Receiving, Forwarding and Storing Messages

Mailgun allows you to receive emails through Routes. Routes will accept emails and then perform an action which can include:

- Forwarding the email to a different email address.
- POSTing the data in the email to a URL.
- Storing the email temporarily for subsequent retrieval through a GET request.

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 forward it to your application via HTTP or to another email address or store the message temporarily (3 days) for subsequent retrieval.

You can define routes visually in the Control Panel, or programmatically using the Routes API.

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

Here's a more formal list of route properties:

Field	Description
Priority	Integer indicating the priority of route execution. Lower numbers have higher priority.
Filter	Filters available in routes - match_recipient() match_header() catchall() (see below 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)

Route Filters

Route filters are expressions that determine when an action is triggered. You can create a filter 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 expression 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. This 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)

Similar to match_recipient but instead of looking at a message recipient, it applies the pattern to an arbitrary MIME header of 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()

Matches if no preceeding 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 evaluates 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 separating them by a comma:

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

Note

If you forward messages to another email address, then you should disable click tracking, 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)

Stores the message temporarily (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 just want to retrieve them later to reduce the frequency of hits on your server.

You can specify a URL and we will notify you when the email arrives along with a URL where 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()

Simply stops the priority waterfall so the subsequent routes will not be evaluated. Without a stop() action executed, all lower priority Routes will also be evaluated.

Receiving Messages via HTTP through a forward() action

When you specify a URL of your application as a route destination through a forward() action, Mailgun will perform an HTTP POST request into it using one of two following formats:

- Fully parsed: Mailgun will parse the message, transcode it into UTF-8 encoding, process attachments, and attempt to separate quoted parts from the actual message. This is the preferred option.
- Raw MIME: message is posted as-is. In this case you are responsible for parsing MIME. To receive raw MIME messages, the destination URL must end with mime.

For Route POSTs, Mailgun listens for the following codes from your server and reacts accordingly:

- If Mailgun receives a 200 (Success) code it will determine the webhook POST is successful and not retry.
- If Mailgun receives a 406 (Not Acceptable) code, Mailgun will determine the POST is rejected and not retry.
- For any other code, Mailgun will retry POSTing according to the schedule below for Webhooks other than the delivery notification.

If your application is unable to process the webhook request but you do not return a 406 error code, Mailgun will retry (other than for delivery notification) during 8 hours at the following intervals before stop trying: 10 minutes, 10 minutes, 15 minutes, 30 minutes, 1 hour, 2 hour and 4 hours.

Below are two tables of HTTP parameters that you can expect to be posted into your application through a forward() action.

Note

In addition to these parameters Mailgun will post *all* MIME headers.

Note

Do not rely on the body-plain, stripped-text, and stripped-signature fields for HTML sanitization. These fields merely provide content from the text/plain portion of an incoming message. This content may contain unescaped HTML.

Parsed Messages Parameters

Parameter	Type	Description
recipient	string	recipient of the message as reported by MAIL TO during SMTP chat.
sender	string	sender of the message as reported by MAIL FROM during SMTP chat. Note: this value may differ from From MIME header.
from	string	sender of the message as reported by From message header, for example “Bob <bob@example.com>”.
subject	string	subject string.

Parameter	Type	Description
body-plain	string	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	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	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	HTML version of the message, without quoted parts.
attachment-count	int	how many attachments the message has.
attachment-x	string	attached file (‘x’ stands for number of the attachment). Attachments are handled as file uploads, encoded as multipart/form-data.
timestamp	int	number of seconds passed since January 1, 1970 (see securing webhooks).
token	string	randomly generated string with length 50 (see securing webhooks).
signature	string	string with hexadecimal digits generate by HMAC algorithm (see securing webhooks).
message-headers	string	list of all MIME headers dumped to a json string (order of headers 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 the message-headers parameter. It was added because not all web frameworks support multi-valued keys parameters. For example Ruby on Rails requires a special syntax to post params like that: you need to add *[]* to a key to collect it’s 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	recipient of the message.
sender	string	sender of the message as reported by SMTP MAIL FROM.
from	string	sender of the message as reported by From message header, for example “Bob <bob@example.com>”.

Parameter	Type	Description
subject	string	subject string.
body-mime	string	full MIME envelope. You will need a MIME parsing library to process this data.
timestamp	int	number of seconds passed since January 1, 1970 (see securing webhooks).
token	string	randomly generated string with length 50 (see securing webhooks).
signature	string	string with hexadecimal digits generate by HMAC algorithm(see securing webhooks).

Note

To receive raw MIME messages and perform your own parsing you must configure a route with a URL ending with “mime”, like http://myhost/post_mime.

Note

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 inspecting the posted data. No programming required.

Storing and Retrieving Messages

When storing an email through a store() action in a Route, you can chose 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.

If 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.

Parameter	Type	Description
domain	string	domain name this message was received for.
recipient	string	recipient of the message as reported by MAIL TO during SMTP chat.
sender	string	sender of the message as reported by MAIL FROM during SMTP chat. Note: this value may differ from From MIME header.
from	string	sender of the message as reported by From message header, for example “Bob Lee <blee@mailgun.net>”.
subject	string	subject string.
body-plain	string	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.

Parameter	Type	Description
stripped-text	string	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	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	HTML version of the message, without quoted parts.
attachments	string	contains a json list of metadata objects, one for each attachment, see below.
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	number of seconds passed since January 1, 1970 (see securing webhooks).
token	string	randomly generated string with length 50 (see securing webhooks).
signature	string	string with hexadecimal digits generate by HMAC algorithm (see securing webhooks).
message-headers	string	list of all MIME headers dumped to a json string (order of headers 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.

The attachments JSON contains the following items.

Parameter	Type	Description
size	integer	indicates the size of the attachment in bytes.
url	string	contains the url where the attachment can be found. This does not support DELETE.
name	string	the name of the attachment
content-type	string	the content type of the attachment

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

Parameter	Type	Description
recipient	string	recipient of the message.

Parameter	Type	Description
sender	string	sender of the message as reported by SMTP MAIL FROM.
from	string	sender of the message as reported by From message header, for example “Bob <bob@example.com>”.
subject	string	subject string.
body-mime	string	full MIME envelope. You will need a MIME parsing library to process this data.

API Routing Samples

You can define routes programmatically using our HTTP API like in these examples.

Create a route of the highest priority with multiple actions:

```
curl -s --user 'api:YOUR_API_KEY' \
    https://api.mailgun.net/v3/routes \
    -F priority=0 \
    -F description='Sample route' \
    -F expression='match_recipient(".*@YOUR_DOMAIN_NAME")' \
    -F action='forward("http://myhost.com/messages/")' \
    -F action='stop()'
```

Sample response:

```
{
  "message": "Route has been created",
  "route": {
      "description": "Sample route",
      "created_at": "Wed, 15 Feb 2012 13:03:31 GMT",
      "actions": [
          "forward(\"http://myhost.com/messages/\")",
          "stop()"
      ],
      "priority": 0,
      "expression": "match_recipient(\".*@samples.mailgun.org\")",
      "id": "4f3bad2335335426750048c6"
  }
}
```

Note

Higher priority routes are handled first. Smaller numbers indicate higher priority. Default is 0.

Listing routes:

```
curl -s --user 'api:YOUR_API_KEY' -G \
    https://api.mailgun.net/v3/routes \
    -d skip=1 \
    -d limit=1
```

Sample response:

```
{
  "total_count": 266,
  "items": [
      {
          "description": "Sample route",
          "created_at": "Wed, 15 Feb 2012 12:58:12 GMT",
          "actions": [
              "forward(\"http://myhost.com/messages/\")",
              "stop()"
          ],
          "priority": 0,
          "expression": "match_recipient(\".*@samples.mailgun.org\")",
          "id": "4f3babe4ba8a481c6400476a"
      }
  ]
}
```

Access the route by id:

```
curl -s --user 'api:YOUR_API_KEY' \
    https://api.mailgun.net/v3/routes/4f3bad2335335426750048c6
```

Sample response:

```
{
  "route": {
      "description": "Sample route",
      "created_at": "Wed, 15 Feb 2012 13:03:31 GMT",
      "actions": [
          "forward(\"http://myhost.com/messages/\")",
          "stop()"
      ],
      "priority": 0,
      "expression": "match_recipient(\".*@samples.mailgun.org\")",
      "id": "4f3bad2335335426750048c6"
  }
}
```

Credentials

Mailgun gives you the ability to programmatically create SMTP credentials which can be used to send mail. SMTP credentials can be used to relay email, through Mailgun, using the SMTP protocol.

SMTP Credentials API Examples

Listing all credentials:

```
curl -s --user 'api:YOUR_API_KEY' -G \
    https://api.mailgun.net/v3/domains/YOUR_DOMAIN_NAME/credentials
```

Sample response:

```
{
  "total_count": 2,
  "items": [
    {
      "size_bytes": 0,
      "created_at": "Tue, 27 Sep 2011 20:24:22 GMT",
      "mailbox": "user@samples.mailgun.org"
      "login": "user@samples.mailgun.org"
    },
    {
      "size_bytes": 0,
      "created_at": "Thu, 06 Oct 2011 10:22:36 GMT",
      "mailbox": "user@samples.mailgun.org"
      "login": "user@samples.mailgun.org"
    }
  ]
}
```

Creating a new SMTP credential:

```
curl -s --user 'api:YOUR_API_KEY' \
    https://api.mailgun.net/v3/domains/YOUR_DOMAIN_NAME/credentials \
    -F login='alice@YOUR_DOMAIN_NAME' \
    -F password='supasecret'
{
  "message": "Created 1 credentials pair(s)"
}
```

Updating the password for a given credential:

```
curl -s --user 'api:YOUR_API_KEY' -X PUT \
    https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/credentials/alice \
    -F password='abc123'
```

Sample response:

```
{
  "message": "Password changed"
}
```

Deleting a given credential:

```
curl -s --user 'api:YOUR_API_KEY' -X DELETE \
    https://api.mailgun.net/v3/domains/YOUR_DOMAIN_NAME/credentials/alice
```

Sample response:

```
{
  "message": "Credentials have been deleted",
  "spec": "alice@samples.mailgun.org"
}
```

Spam Filter

If you are receiving email, you need spam filtering. Mailgun spam filtering is powered by an army of SpamAssassin machines. Mailgun gives you three ways to configure spam filtering. You can select the appropriate option in the Control Panel when you click on a domain name in the 'Domains' tab.

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

If you chose option 3, there are four headers we provide for you: X-Mailgun-Sflag, X-Mailgun-Sscore, X-Mailgun-Dkim-Check-Result and X-Mailgun-Spf.

X-Mailgun-Sflag
: Inserted with the value 'Yes' if the message was classified as a spam.

X-Mailgun-Sscore
: A 'spamicity' score that you can use to calibrate your own filter. Inserted for every message checked for a spam. The score ranges from low negative digits (very unlikely to be spam) to 20 and occasionally higher (very likely to be spam).

  At the time of writing this, we are filtering spam at a score of around 5.0 but we are constantly calibrating this.

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'.

SMTP Protocol

In addition to our HTTP API, Mailgun servers supports the standard SMTP protocol... You can send using SMTP with or without TLS.

Please consult a standard library documentation for language of your choice to learn how to use the SMTP protocol. Below are some helpful links for a few popular languages:

- Ruby SMTP
- Python SMTP
- JavaMail API

SMTP Relay

You can also configure your own mailserver to relay mail via Mailgun as shown below. All of them require these three variables which you can look up in the Control Panel:

- Your SMTP username
- Your SMTP password
- SMTP host name mailserver (these instructions will use smtp.mailgun.org as an example)

You have an SMTP username and password for each domain you have at Mailgun. To send mail from a particular domain, just use the appropriate credentials.

Postfix Instructions

You have to configure a relay host with SASL authentication like shown below:

```
# /etc/postfix/main.cf:

mydestination = localhost.localdomain, localhost
relayhost = [smtp.mailgun.org]:587
smtp_sasl_auth_enable = yes
smtp_sasl_password_maps = static:postmaster@mydomain.com:password
smtp_sasl_security_options = noanonymous

# TLS support
smtp_tls_security_level = may
smtpd_tls_security_level = may
smtp_tls_note_starttls_offer = yes
```

When using TLS encryption, make sure Postfix knows where to locate the CA database for your Linux distribution:

```
smtpd_tls_key_file = /etc/ssl/private/smtpd.key
smtpd_tls_cert_file = /etc/ssl/certs/smtpd.crt
smtpd_tls_CApath = /etc/ssl/certs
```

Note

You can use SMTP Credentials, but not your Control Panel password.

Exim Instructions

For more information see Exim's documentation for authenticated outgoing SMTP. You need to configure "smarthost" for your Exim setup.

```
# In your exim.conf:
# In routes configuration:
mailgun:
        driver = manualroute
        domains = ! +local_domains
        transport = mailgun_transport
        route_list = * smtp.mailgun.org byname
```

```
# In transports configuration:
        mailgun_transport:
        driver=smtp
        hosts_try_auth = smtp.mailgun.org
```

Also make sure to configure login credentials (in your /etc/exim/passwd.client):

```
*.mailgun.org:username:password
```

Sendmail Instructions

Define the smarthost in your sendmail.mc before mailer definitions:

```
## Mailgun
define(`SMART_HOST', `smtp.mailgun.org')dnl
FEATURE(`authinfo', `hash /etc/mail/authinfo')dnl
# optional, see http://www.sendmail.org/m4/features.html before enabling:
# FEATURE(`accept_unresolvable_domains')dnl
# FEATURE(`accept_unqualified_senders')dnl
# execute: make -C /etc/mail
## Mailgun
```

Specify login credentials in your authinfo:

```
AuthInfo:smtp.mailgun.org "U:<LOGIN>" "P:<PASSWORD>" "M:PLAIN"
```

Don’t forget to run the following command and then restart sendmail:

```
make -C /etc/mail
```

Using Standard Email Clients

Standard email clients like Thunderbird or Outlook can also be used to send mail.

Settings for sending mail:

```
SMTP server: smtp.mailgun.org
```

Note

Use a full address like “user@mymailgundomain.com” as a login for SMTP. SSL or TLS are supported.

TLS Sending Connection Settings

For message delivery, Mailgun exposes two flags that will work at the domain level or message level (message level will override domain level) that allow you to control how messages are

delivered. See documentation for sending messages and domains for examples on how these fields can be updated.

require tls: If set to *True* this requires the message only be sent over a TLS connection. If a TLS connection can not be established, Mailgun will not deliver the message. If set to *False*, Mailgun will still try and upgrade the connection, but if Mailgun can not, the message will be delivered over a plaintext SMTP connection. The default is False.

skip verification: If set to *True*, the certificate and hostname will not be verified when trying to establish a TLS connection and Mailgun will accept any certificate during delivery. If set to *False*, Mailgun will verify the certificate and hostname. If either one can not be verified, a TLS connection will not be established. The default is *False*.

To help you better understand the configuration possibilities and potential issues, take a look at the following table. Take into account the type of threat you are concerned with when making your decision on how to configure sending settings. By default, *require-tls* and *skip-verification* are *false*.

require-tls	skip-verification	TLS	TLS Active Attack (MITM)	TLS Passive Attack (Capture)	Passive Plaintext Capture
false	false	Attempt	Not Possible	Not Possible	Possible via downgrade
false	true	Attempt	Possible	Not Possible	If STARTTLS not offered
true	false	Required	Not Possible	Not Possible	Not Possible
true	true	Required	Possible	Not Possible	Not Possible

Additionally the following fields are available in your logs under *delivery-status* to indicate how the message was delivered:

Field	Description
tls	Indicates if a TLS connection was used or not when delivering the message.
certificate-verified	Indicates if we verified the certificate or not when delivering the message.
mx-host	Tells you the MX server we connected to to deliver the message.