Mailgun REST API Documentation

Mailgun Inc

Mar 10, 2021

Contents

Learn how to send email from your app, SMTP vs API, verifying your domain, email reputation; we'll explain it all here.

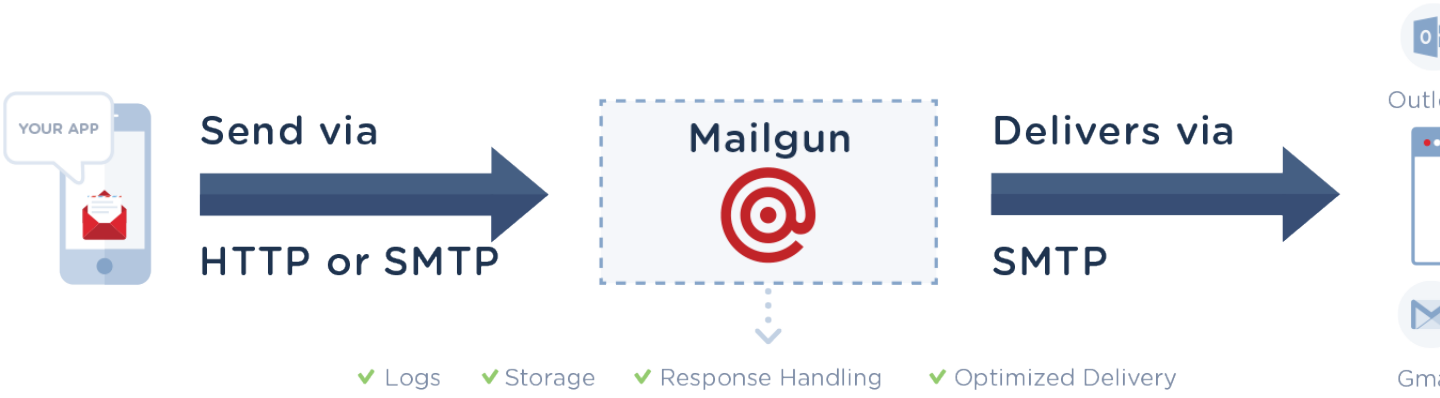
CHAPTER 1

Quickstart Guide

Learn how to send email from your app, SMTP vs API, verifying your domain, email reputation; we'll explain it all here.

Once you sign up for a Mailgun account, keep your API key handy and try our quickstart guides below.

1.1 How to start sending email



1.1.1 Send with SMTP or API

It's up to you, whatever you find easier is fine with us. Here's something to consider:

In short, SMTP is an open and established protocol with large ecosystem, while Mailgun API is better long term performance and maintenance wise.

Send via API

Run this:

```
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 awesomeness!'
```

```
import java.io.File;

import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;


public class MGSample {

    // ...

    public static JsonNode sendSimpleMessage() throws UnirestException {

        HttpResponse<JsonNode> request = Unirest.post("https://api.mailgun.net/v3/" +␣
↪YOUR_DOMAIN_NAME + "/messages")
            .basicAuth("api", API_KEY)
            .field("from", "Excited User <USER@YOURDOMAIN.COM>")
            .field("to", "artemis@example.com")
            .field("subject", "hello")
            .field("text", "testing")
            .asJson();

        return request.getBody();
    }
}
```

```
# Include the Autoloader (see "Libraries" for install instructions)
require 'vendor/autoload.php';
use Mailgun\Mailgun;

# Instantiate the client.
$mgClient = Mailgun::create('PRIVATE_API_KEY', 'https://API_HOSTNAME');
$domain = "YOUR_DOMAIN_NAME";
$params = array(
  'from'    => 'Excited User <YOU@YOUR_DOMAIN_NAME>',
  'to'      => 'bob@example.com',
  'subject' => 'Hello',
  'text'    => 'Testing some Mailgun awesomness!'
);

# Make the call to the client.
$mgClient->messages()->send($domain, $params);
```

```python
def send_simple_message():
    return requests.post(
        "https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/messages",
        auth=("api", "YOUR_API_KEY"),
        data={"from": "Excited User <mailgun@YOUR_DOMAIN_NAME>",
              "to": ["bar@example.com", "YOU@YOUR_DOMAIN_NAME"],
              "subject": "Hello",
              "text": "Testing some Mailgun awesomness!"})
```

```ruby
def send_simple_message
  RestClient.post "https://api:YOUR_API_KEY"\
  "@api.mailgun.net/v3/YOUR_DOMAIN_NAME/messages",
  :from => "Excited User <mailgun@YOUR_DOMAIN_NAME>",
  :to => "bar@example.com, YOU@YOUR_DOMAIN_NAME",
  :subject => "Hello",
  :text => "Testing some Mailgun awesomness!"
end
```

```csharp
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class SendSimpleMessageChunk
{

    public static void Main (string[] args)
    {
        Console.WriteLine (SendSimpleMessage ().Content.ToString ());
    }

    public static IRestResponse SendSimpleMessage ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.AddParameter ("domain", "YOUR_DOMAIN_NAME", ParameterType.UrlSegment);
        request.Resource = "{domain}/messages";
        request.AddParameter ("from", "Excited User <mailgun@YOUR_DOMAIN_NAME>");
        request.AddParameter ("to", "bar@example.com");
        request.AddParameter ("to", "YOU@YOUR_DOMAIN_NAME");
        request.AddParameter ("subject", "Hello");
        request.AddParameter ("text", "Testing some Mailgun awesomness!");
        request.Method = Method.POST;
        return client.Execute (request);
    }

}
```

```go
import (
    "context"
    "github.com/mailgun/mailgun-go/v3"
    "time"
```

(continues on next page)

(continued from previous page)

```
}

func SendSimpleMessage(domain, apiKey string) (string, error) {
    mg := mailgun.NewMailgun(domain, apiKey)
    m := mg.NewMessage(
        "Excited User <mailgun@YOUR_DOMAIN_NAME>",
        "Hello",
        "Testing some Mailgun awesomeness!",
        "YOU@YOUR_DOMAIN_NAME",
    )

    ctx, cancel := context.WithTimeout(context.Background(), time.Second*30)
    defer cancel()

    _, id, err := mg.Send(ctx, m)
    return id, err
}
```

```
var API_KEY = 'YOUR_API_KEY';
var DOMAIN = 'YOUR_DOMAIN_NAME';
var mailgun = require('mailgun-js')({apiKey: API_KEY, domain: DOMAIN});

const data = {
  from: 'Excited User <me@samples.mailgun.org>',
  to: 'foo@example.com, bar@example.com',
  subject: 'Hello',
  text: 'Testing some Mailgun awesomeness!'
};

mailgun.messages().send(data, (error, body) => {
  console.log(body);
});
```

NOTE: If you're sending from our EU infrastructure, be sure to substitute our EU endpoint in the above example: https://api.eu.mailgun.net/v3

What actually happened:

- Mailgun assembled a MIME message.
- Added the log entries to our full text search index.
- Delivered the email.

You can find your private API key on your dashboard.

Send via SMTP

Run this:

```
# Swaks is an smtp of CURL, install it first:
curl http://www.jetmore.org/john/code/swaks/files/swaks-20130209.0/swaks -o swaks
# Set the permissions for the script so you can run it
chmod +x swaks
# It's based on perl, so install perl
sudo apt-get -y install perl
# now send!
```

(continues on next page)

(continued from previous page)

```
./swaks --auth \
        --server smtp.mailgun.org \
        --au postmaster@YOUR_DOMAIN_NAME \
        --ap 3kh9umujora5 \
        --to bar@example.com \
        --h-Subject: "Hello" \
        --body 'Testing some Mailgun awesomness!'
```

```
import java.io.*;
import java.net.InetAddress;
import java.util.Properties;
import java.util.Date;
import javax.mail.*;
import javax.mail.internet.*;
import com.sun.mail.smtp.*;

public class MGSendSimpleSMTP {

    public static void main(String args[]) throws Exception {

        Properties props = System.getProperties();
        props.put("mail.smtps.host", "smtp.mailgun.org");
        props.put("mail.smtps.auth", "true");

        Session session = Session.getInstance(props, null);
        Message msg = new MimeMessage(session);
        msg.setFrom(new InternetAddress("YOU@YOUR_DOMAIN_NAME"));

        InternetAddress[] addrs = InternetAddress.parse("bar@example.com", false);
        msg.setRecipients(Message.RecipientType.TO, addrs)

        msg.setSubject("Hello");
        msg.setText("Testing some Mailgun awesomness");
        msg.setSentDate(new Date());

        SMTPTransport t =
            (SMTPTransport)session.getTransport("smtps");
        t.connect("smtp.mailgun.org", "postmaster@YOUR_DOMAIN_NAME", "YOUR_SMTP_
↪PASSWORD");
        t.sendMessage(msg, msg.getAllRecipients());

        System.out.println("Response: " + t.getLastServerResponse());

        t.close();
    }
}
```

```
// Using Awesome https://github.com/PHPMailer/PHPMailer
<?php
require 'PHPMailerAutoload.php';

$mail = new PHPMailer;

$mail->isSMTP();                                      // Set mailer to use SMTP
$mail->Host = 'smtp.mailgun.org';                     // Specify main and backup SMTP␣
↪servers
```

(continues on next page)

(continued from previous page)

```
$mail->SMTPAuth = true;                                 // Enable SMTP authentication
$mail->Username = 'postmaster@YOUR_DOMAIN_NAME';    // SMTP username
$mail->Password = 'secret';                             // SMTP password
$mail->SMTPSecure = 'tls';                              // Enable encryption, only 'tls
↪' is accepted

$mail->From = 'YOU@YOUR_DOMAIN_NAME';
$mail->FromName = 'Mailer';
$mail->addAddress('bar@example.com');                   // Add a recipient

$mail->WordWrap = 50;                                   // Set word wrap to 50␣
↪characters

$mail->Subject = 'Hello';
$mail->Body    = 'Testing some Mailgun awesomness';

if(!$mail->send()) {
    echo 'Message could not be sent.';
    echo 'Mailer Error: ' . $mail->ErrorInfo;
} else {
    echo 'Message has been sent';
}
```

```
import smtplib

from email.mime.text import MIMEText

msg = MIMEText('Testing some Mailgun awesomness')
msg['Subject'] = "Hello"
msg['From']    = "foo@YOUR_DOMAIN_NAME"
msg['To']      = "bar@example.com"

s = smtplib.SMTP('smtp.mailgun.org', 587)

s.login('postmaster@YOUR_DOMAIN_NAME', '3kh9umujora5')
s.sendmail(msg['From'], msg['To'], msg.as_string())
s.quit()
```

```
# install `mail` gem first: `gem install mail`

require 'mail'

Mail.defaults do
  delivery_method :smtp, {
    :port      => 587,
    :address   => "smtp.mailgun.org",
    :user_name => "",
    :password  => "",
  }
end

mail = Mail.deliver do
  to      'bar@example.com'
  from    'foo@YOUR_DOMAIN_NAME'
  subject 'Hello'
```

(continues on next page)

(continued from previous page)

```
  text_part do
    body 'Testing some Mailgun awesomness'
  end
end
```

```
using System;
using System.IO;
using MailKit;
using MailKit.Net.Smtp;
using MimeKit;
using RestSharp;
using RestSharp.Authenticators;

public class SmtpMessageChunk
{

    public static void Main (string[] args)
    {
        SendMessageSmtp ();
    }

    public static void SendMessageSmtp ()
    {
        // Compose a message
        MimeMessage mail = new MimeMessage ();
        mail.From.Add (new MailboxAddress ("Excited Admin", "foo@YOUR_DOMAIN_NAME"));
        mail.To.Add (new MailboxAddress ("Excited User", "bar@example.com"));
        mail.Subject = "Hello";
        mail.Body = new TextPart ("plain") {
            Text = @"Testing some Mailgun awesomesauce!",
        };

        // Send it!
        using (var client = new SmtpClient ()) {
            // XXX - Should this be a little different?
            client.ServerCertificateValidationCallback = (s, c, h, e) => true;

            client.Connect ("smtp.mailgun.org", 587, false);
            client.AuthenticationMechanisms.Remove ("XOAUTH2");
            client.Authenticate ("postmaster@YOUR_DOMAIN_NAME", "3kh9umujora5");

            client.Send (mail);
            client.Disconnect (true);
        }
    }


}
```

```
import (
    "github.com/jordan-wright/email"
)

func main() {
    e := email.NewEmail()
    e.From = "Your Name <foo@YOUR_DOMAIN_NAME>"
    e.To = []string{"bar@example.com"}
```

(continues on next page)

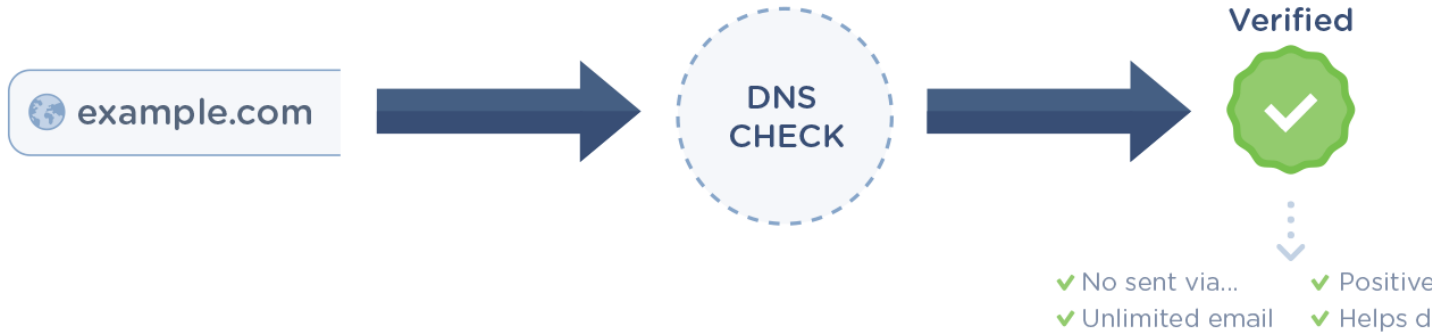
(continued from previous page)

```
    e.Subject = "Hello"
    e.Text = []byte("Testing some Mailgun awesomeness")
    err := e.Send("smtp.mailgun.org:587", smtp.PlainAuth("", "YOUR_USERNAME", "YOUR_
↪PASSWORD", "smtp.mailgun.org"))
    if err != nil {
        panic(err)
    }
}
```

You can find your SMTP credentials for each domain on your domains tab.

1.1.2 Verify Your Domain

Add a domain you own and verify it by setting up the DNS records we provide (known as SPF and DKIM) at your DNS provider.



you are an **authorized sender** for the domain.

How to verify your domain

1. Add your domain or subdomain.
2. Open your DNS provider and add the **two TXT DNS records** provided.
3. If you want Mailgun to **track clicks and opens** you can also add the **CNAME** record.
4. MX records should also be added, **unless you already have MX records** for your domain pointed at another email service provider (e.g. Gmail).

Once you've added the records and they've propagated, your domain will be verified.

Note: It can take 24-48 hours for DNS changes to propagate.

If you will be creating a lot of domains, Mailgun offers an API endpoint for adding/editing/removing domains from your account. See the *Domains* endpoint for more information.

Add Sending & Tracking DNS Records

- **SPF**: Sending server IP validation. Used by majority of inbound mail servers. SPF Information.
- **DKIM**: Like SPF, but uses cryptographic methods for validation. Supported by many inbound mail servers. DKIM Information
- **CNAME**: Used for tracking opens and clicks, when enabled. *Tracking Messages*

Type	Value	Purpose
TXT	"v=spf1 include:mailgun.org ~all"	SPF (Required)
TXT	*Find this record in your Control Panel, Domains Tab*	DKIM (Required)
CNAME	"mailgun.org"	Tracking (Optional)

Note: While the CNAME is listed as optional, it is required to enable Unsubscribe and Click tracking links.

Add Receiving MX Records

Mail server for handling inbound messages. MX Information

Type	Value	Purpose
MX	mxa.mailgun.org	Receiving (Optional)
MX	mxb.mailgun.org	Receiving (Optional)

Warning: Do not configure Receiving MX DNS records if you already have another provider handling inbound mail delivery (e.g. Gmail).

1.1.3 Common DNS Providers

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

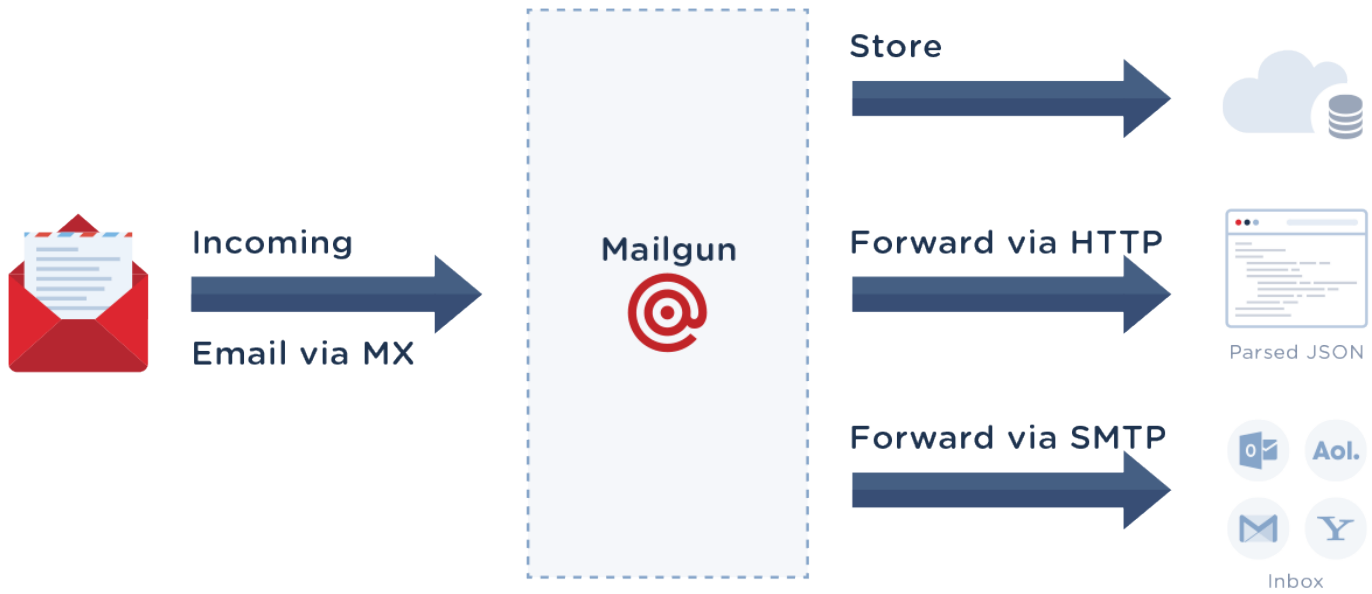
Amazon Route 53: Developer Guide

DigitalOcean: Mailgun and DigitalOcean Guide

You are all set!

Read more about *How to start receiving inbound email* and *How to start tracking email events*.

1.2 How to start receiving inbound email



1.2.1 Add Receiving MX Records

Your domain needs Mailgun MX records to handle inbound messages. Open up your DNS provider and add these. MX Information

Type	Value	Purpose
MX	mxa.mailgun.org	Receiving (Optional)
MX	mxb.mailgun.org	Receiving (Optional)

Warning: Do not configure Receiving MX DNS records if you already have another provider handling inbound mail delivery for your domain (e.g. Gmail). Instead we recommend using a subdomain on Mailgun (e.g. mg.yourdomain.com)

1.2.2 Inbound Routes and Parsing

You can define a list of **routes to handle incoming emails** and prioritize the sequence of their execution.

- Each **route consists of a filter expression and an action**.
- When a message is received, Mailgun evaluates the filter expression against it.
- If the expression is true, the action is executed.

Regular expressions can be used to match against message recipients or arbitrary headers such as subject.

Examples of filter expressions for routes

Expression	Description
match_recipient("bob@myapp.com")	Returns true if the incoming message is going to bob@myapp.com.
match_recipient(".*@myapp.com")	Returns true if the incoming message is going to any user at @myapp.com.
match_header("subject", "hello")	Returns true if the subject of the message contains word 'hello'.
catch_all()	Returns true if no other route matched, to implement catch-all behaviour.

Supported actions for routes

Action	Description
forward("http://myapp/post")	Parses the message and forwards it to a given URL.
forward("support@myapp.com")	Forwards the message to a given email address.
store(notify="http://myapp/post")	Stores the message temporarily to be retrieved later.
stop()	Stops and doesn't look at any other routes.

Routes can be defined and tested using the Mailgun API (in addition, to using the Control Panel).

```
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")' \
```

(continues on next page)

(continued from previous page)

```
    -F action='forward("http://myhost.com/messages/")' \
    -F action='stop()'
```

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode createRoute() throws UnirestException {

        HttpResponse <JsonNode> request = Unirest.post("https://api.mailgun.net/v3/
↪routes")
            .basicAuth("api", API_KEY)
            .field("priority", "0")
            .field("description", "sample route")
            .field("expression", "match_recipient('.*@YOUR_DOMAIN_NAME')")
            .field("action", "forward('http://myhost.com/messages/')")
            .field("action", "stop()")
            .asJson();

        return request.getBody();
    }
}
```

```
# Include the Autoloader (see "Libraries" for install instructions)
require 'vendor/autoload.php';
use Mailgun\Mailgun;

# Instantiate the client.
$mgClient = Mailgun::create('PRIVATE_API_KEY', 'https://API_HOSTNAME');

# Define your expression, actions, and description
$expression  = 'match_recipient(".*@mg.example.com")';
$actions     = array('forward("my_address@example.com")', 'stop()');
$description = 'Catch All and Forward';

# Issue the call to the client.
$result = $mgClient->routes()->create($expression, $actions, $description);
```

```
def create_route():
    return requests.post(
        "https://api.mailgun.net/v3/routes",
        auth=("api", "YOUR_API_KEY"),
        data={"priority": 0,
              "description": "Sample route",
              "expression": "match_recipient('.*@YOUR_DOMAIN_NAME')",
              "action": ["forward('http://myhost.com/messages/')", "stop()"]})
```

```
def create_route
  data = {}
  data[:priority] = 0
```

(continues on next page)

(continued from previous page)

```
  data[:description] = "Sample route"
  data[:expression] = "match_recipient('.*@YOUR_DOMAIN_NAME')"
  data[:action] = []
  data[:action] << "forward('http://myhost.com/messages/')"
  data[:action] << "stop()"
  RestClient.post "https://api:YOUR_API_KEY"\
  "@api.mailgun.net/v3/routes", data
end
```

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class CreateRouteChunk
{

    public static void Main (string[] args)
    {
        Console.WriteLine (CreateRoute ().Content.ToString ());
    }

    public static IRestResponse CreateRoute ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.Resource = "routes";
        request.AddParameter ("priority", 0);
        request.AddParameter ("description", "Sample route");
        request.AddParameter ("expression", "match_recipient('.*@YOUR_DOMAIN_NAME')");
        request.AddParameter ("action",
                              "forward('http://myhost.com/messages/')");
        request.AddParameter ("action", "stop()");
        request.Method = Method.POST;
        return client.Execute (request);
    }

}
```

```
import (
    "context"
    "github.com/mailgun/mailgun-go/v3"
    "time"
)

func CreateRoute(domain, apiKey string) (mailgun.Route, error) {
    mg := mailgun.NewMailgun(domain, apiKey)

    ctx, cancel := context.WithTimeout(context.Background(), time.Second*30)
    defer cancel()

    return mg.CreateRoute(ctx, mailgun.Route{
```

(continues on next page)

(continued from previous page)

```
        Priority:    1,
        Description: "Sample Route",
        Expression:  "match_recipient(\".*@YOUR_DOMAIN_NAME\")",
        Actions: []string{
            "forward(\"http://example.com/messages/\")",
            "stop()",
        },
    })
}
```

```
var DOMAIN = 'YOUR_DOMAIN_NAME';
var mailgun = require('mailgun-js')({ apiKey: "YOUR_API_KEY", domain: DOMAIN });

mailgun.post('/routes', {"priority": 0, "description": 'Sample route', "expression":
↪'match_recipient(".*@YOUR_DOMAIN_NAME")', "action": 'forward("http://myhost.com/
↪messages/")', "action": 'stop()'}, function (error, body) {
  console.log(body);
});
```

The example above defines a new route which will forward all messages coming to @samples.mailgun.org to http://myhost.com/messages and will stop evaluating any other routes.

Now let's look at how to build HTTP handlers for incoming messages, i.e. what needs to be done on your end to handle a message that Mailgun forwards to your URL.

Consider this Django code:

```
# Handler for HTTP POST to http://myhost.com/messages for the route defined above
def on_incoming_message(request):
    if request.method == 'POST':
        sender    = request.POST.get('sender')
        recipient = request.POST.get('recipient')
        subject   = request.POST.get('subject', '')

        body_plain = request.POST.get('body-plain', '')
        body_without_quotes = request.POST.get('stripped-text', '')
        # note: other MIME headers are also posted here...

        # attachments:
        for key in request.FILES:
            file = request.FILES[key]
            # do something with the file

    # Returned text is ignored but HTTP status code matters:
    # Mailgun wants to see 2xx, otherwise it will make another attempt in 5 minutes
    return HttpResponse('OK')
```

Mailgun routes are very powerful. For example, you can use regular expression captures and refer to captured values in your destination.

To learn more about Routes, check out the *Routes* section of the *User Manual*.

1.3 How to start tracking email events

Once you start sending and receiving messages, it's important to track what's happening with them. Mailgun provides a variety of methods to access data about your emails, which you can read more about in the *Tracking Messages*

section of the *User Manual*. Below is a brief summary of Events, the Events API and Events Webhooks.

1.3.1 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
```

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...
```

(continues on next page)

(continued from previous page)

```
    public static JsonNode getLogs() throws UnirestException {

        HttpResponse<JsonNode> request = Unirest.get("https://api.mailgun.net/v3/" +
↪YOUR_DOMAIN_NAME + "/events")
            .basicAuth("api", API_KEY)
            .queryString("begin", "Thurs, 18 May 2017 09:00:00 -0000")
            .queryString("ascending", "yes")
            .queryString("limit", 1)
            .asJson();

        return request.getBody();
    }
}
```

```
# Include the Autoloader (see "Libraries" for install instructions)
require 'vendor/autoload.php';
use Mailgun\Mailgun;

# Instantiate the client.
$mgClient    = Mailgun::create('PRIVATE_API_KEY', 'https://API_HOSTNAME');
$domain      = 'YOUR_DOMAIN_NAME';
$queryString = array(
    'begin'        => 'Wed, 1 Jan 2020 09:00:00 -0000',
    'ascending'    => 'yes',
    'limit'        =>  25,
    'pretty'       => 'yes',
    'recipient'    => 'bob@example.com'
);

# Issue the call to the client.
$result = $mgClient->events()->get($domain, $queryString);
```

```
def get_logs():
    return requests.get(
        "https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/events",
        auth=("api", "YOUR_API_KEY"),
        params={"begin"       : "Fri, 3 May 2013 09:00:00 -0000",
                "ascending"   : "yes",
                "limit"       :  25,
                "pretty"      : "yes",
                "recipient"   : "joe@example.com"})
```

```
def get_logs
  RestClient.get "https://api:YOUR_API_KEY"\
  "@api.mailgun.net/v3/YOUR_DOMAIN_NAME/events",
   :params => {
    :'begin'       => 'Fri, 3 May 2013 09:00:00 -0000',
    :'ascending'   => 'yes',
    :'limit'       =>  25,
    :'pretty'      => 'yes',
    :'recipient' => 'joe@example.com'
   }
end
```

```
using System;
```

(continues on next page)

(continued from previous page)

```
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class EventsDateTimeRecipientChunk
{

    public static void Main (string[] args)
    {
        Console.WriteLine (EventsDateTimeRecipient ().Content.ToString ());
    }

    public static IRestResponse EventsDateTimeRecipient ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.AddParameter ("domain", "YOUR_DOMAIN_NAME", ParameterType.UrlSegment);
        request.Resource = "{domain}/events";
        request.AddParameter ("begin", "Fri, 3 May 2013 09:00:00 -0000");
        request.AddParameter ("ascending", "yes");
        request.AddParameter ("limit", 25);
        request.AddParameter ("pretty", "yes");
        request.AddParameter ("recipient", "joe@example.com");
        return client.Execute (request);
    }

}
```

```
import (
    "context"
    "github.com/mailgun/mailgun-go/v3"
    "time"
)

func PrintEventLog(domain, apiKey string) error {
    mg := mailgun.NewMailgun(domain, apiKey)

    // Create an iterator
    it := mg.ListEvents(&mailgun.ListEventOptions{
        Begin: time.Now().Add(-50 * time.Minute),
        Limit: 100,
        Filter: map[string]string{
            "recipient": "joe@example.com",
        },
    })

    ctx, cancel := context.WithTimeout(context.Background(), time.Second*30)
    defer cancel()

    // Iterate through all the pages of events
    var page []mailgun.Event
    for it.Next(ctx, &page) {
```

(continues on next page)

(continued from previous page)

```
        for _, event := range page {
            fmt.Printf("%+v\n", event)
        }
    }

    // Did iteration end because of an error?
    if it.Err() != nil {
        return it.Err()
    }

    return nil
}
```

```
var DOMAIN = 'YOUR_DOMAIN_NAME';
var mailgun = require('mailgun-js')({ apiKey: "YOUR_API_KEY", domain: DOMAIN });

mailgun.get(`/${DOMAIN}/events`, {"begin": "Thurs, 06 July 2017 09:00:00 -0000",
↪"ascending": "yes", "limit": 1},  function (error, body) {
  console.log(body);
});
```

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"
    }
  ],
```

(continues on next page)

(continued from previous page)

```
  "paging": {
    "next":
        "https://api.mailgun.net/v3/samples.mailgun.org/events/W3siY..."
    "previous":
        "https://api.mailgun.net/v3/samples.mailgun.org/events/Lkawm..."
  }
}
```

Events Webhooks

Mailgun can also 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 respective 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 *Webhooks* section of the *User Manual*. We recommend using http://bin.mailgun.net for creating temporary URLs to test and debug your webhooks.

1.3.2 Other Goodies

In addition to sending, receiving and storing mail, Mailgun can also help developers with the following:

- Automatic "Unsubscribe me" functionality.
- Support for email campaigns and tracking their performance via tags.
- Bounce handling.
- Spam complaints handling.
- Spam filtering for incoming messages.
- Searchable email logs.

The list of what Mailgun can do for you is growing every day. Please take a look at our *User Manual* to learn more.

CHAPTER 2

User Manual

2.1 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 four major features that Mailgun provides:

- *Sending Messages*
- *Tracking Messages*
- *Receiving, Forwarding and Storing Messages*
- *Email Validation V3*
- *Email Validation V4*
- *Inbox Placement*

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://app.mailgun.com/app/dashboard.

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 Guide* or *API Reference*. There are also *FAQ* and *Email Best Practices* that you can reference.

Finally, always feel free to contact our Support Team.

2.2 Getting Started

2.2.1 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 the two **TXT** DNS records found in the **Domain Verification & DNS** section of the domain settings page of the Mailgun control panel to 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 of the settings page for a particular domain in the Mailgun control panel.*
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

2.2.2 Managing User Roles

How to Manage

Role-based access control sets all current users to Admin-level users by default. To assign different roles to your account's users, please visit the Account section of the control panel. There, you can choose the appropriate permissions level for each user. And when it's time to add new users to your account, you'll be able to easily select a role upon user creation.

Roles

Role	Description
Analyst	Analyst users are very limited. They have access to read most data, but can only modify their own settings.
Billing	Billing users are focused on billing actions. Most of their access will be read only, billing is the only non-admin users who have access to: • Account Upgrade • Editing Credit card on File • Setting/Clearing Custom Send Limits • Setting/Clearing Custom Validation Limits
Support	Support users are restricted in what they can edit. In addition to being able to read most data, they will be able to: • Edit suppressions • Edit mailing lists and members • Edit authorized recipients • Open and comment on support tickets
Developer	Developer users are highly trusted. This role can read and write almost all data. This includes everything support has, plus can: • Edit webhooks • Edit routes • Edit domain settings • View API Keys and SMTP passwords **Warning:** This role has access to read API Keys and SMTP credentials. This data is highly sensitive.
Admin	Admin users have read and write access to everything. Only admins on the account can: • Rotate and expire API Keys • Create and revoke SMTP credentials • Create and administer control panel users • Edit account details **Note:** The account owner will always be an admin user.

2.3 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.

2.3.1 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 awesomeness!'
```

```
import java.io.File;

import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;


public class MGSample {

    // ...

    public static JsonNode sendSimpleMessage() throws UnirestException {

        HttpResponse<JsonNode> request = Unirest.post("https://api.mailgun.net/v3/" +␣
↪YOUR_DOMAIN_NAME + "/messages"
```

(continues on next page)

(continued from previous page)

```
            .basicAuth("api", API_KEY)
            .field("from", "Excited User <USER@YOURDOMAIN.COM>")
            .field("to", "artemis@example.com")
            .field("subject", "hello")
            .field("text", "testing")
            .asJson();

        return request.getBody();
    }
}
```

```
# Include the Autoloader (see "Libraries" for install instructions)
require 'vendor/autoload.php';
use Mailgun\Mailgun;

# Instantiate the client.
$mgClient = Mailgun::create('PRIVATE_API_KEY', 'https://API_HOSTNAME');
$domain = "YOUR_DOMAIN_NAME";
$params = array(
  'from'    => 'Excited User <YOU@YOUR_DOMAIN_NAME>',
  'to'      => 'bob@example.com',
  'subject' => 'Hello',
  'text'    => 'Testing some Mailgun awesomness!'
);

# Make the call to the client.
$mgClient->messages()->send($domain, $params);
```

```
def send_simple_message():
    return requests.post(
        "https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/messages",
        auth=("api", "YOUR_API_KEY"),
        data={"from": "Excited User <mailgun@YOUR_DOMAIN_NAME>",
              "to": ["bar@example.com", "YOU@YOUR_DOMAIN_NAME"],
              "subject": "Hello",
              "text": "Testing some Mailgun awesomness!"})
```

```
def send_simple_message
  RestClient.post "https://api:YOUR_API_KEY"\
  "@api.mailgun.net/v3/YOUR_DOMAIN_NAME/messages",
  :from => "Excited User <mailgun@YOUR_DOMAIN_NAME>",
  :to => "bar@example.com, YOU@YOUR_DOMAIN_NAME",
  :subject => "Hello",
  :text => "Testing some Mailgun awesomness!"
end
```

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class SendSimpleMessageChunk
{

    public static void Main (string[] args)
```

(continues on next page)

(continued from previous page)

```
    {
        Console.WriteLine (SendSimpleMessage ().Content.ToString ());
    }

    public static IRestResponse SendSimpleMessage ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.AddParameter ("domain", "YOUR_DOMAIN_NAME", ParameterType.UrlSegment);
        request.Resource = "{domain}/messages";
        request.AddParameter ("from", "Excited User <mailgun@YOUR_DOMAIN_NAME>");
        request.AddParameter ("to", "bar@example.com");
        request.AddParameter ("to", "YOU@YOUR_DOMAIN_NAME");
        request.AddParameter ("subject", "Hello");
        request.AddParameter ("text", "Testing some Mailgun awesomness!");
        request.Method = Method.POST;
        return client.Execute (request);
    }

}
```

```
import (
    "context"
    "github.com/mailgun/mailgun-go/v3"
    "time"
)

func SendSimpleMessage(domain, apiKey string) (string, error) {
    mg := mailgun.NewMailgun(domain, apiKey)
    m := mg.NewMessage(
        "Excited User <mailgun@YOUR_DOMAIN_NAME>",
        "Hello",
        "Testing some Mailgun awesomeness!",
        "YOU@YOUR_DOMAIN_NAME",
    )

    ctx, cancel := context.WithTimeout(context.Background(), time.Second*30)
    defer cancel()

    _, id, err := mg.Send(ctx, m)
    return id, err
}
```

```
var API_KEY = 'YOUR_API_KEY';
var DOMAIN = 'YOUR_DOMAIN_NAME';
var mailgun = require('mailgun-js')({apiKey: API_KEY, domain: DOMAIN});

const data = {
  from: 'Excited User <me@samples.mailgun.org>',
  to: 'foo@example.com, bar@example.com',
  subject: 'Hello',
  text: 'Testing some Mailgun awesomeness!'
```

(continues on next page)

(continued from previous page)

```
});

mailgun.messages().send(data, (error, body) => {
  console.log(body);
});
```

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
```

```
import java.io.File;

import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode sendComplexMessage() throws UnirestException {

        HttpResponse<JsonNode> request = Unirest.post("https://api.mailgun.net/v3/" +␣
↪YOUR_DOMAIN_NAME + "/messages")
                .basicAuth("api", API_KEY)
                .field("from", "Excited User <USER@YOURDOMAIN.COM>")
                .field("to", "alice@example.com")
                .field("cc", "bob@example.com")
                .field("bcc", "joe@example.com")
                .field("subject", "Hello")
                .field("text", "Testing out some Mailgun awesomeness!")
                .field("html", "<html>HTML version </html>")
                .field("attachment", new File("/temp/folder/test.txt"))
                .asJson();

        return request.getBody();
    }
}
```

```php
# Include the Autoloader (see "Libraries" for install instructions)
require 'vendor/autoload.php';
use Mailgun\Mailgun;

# Instantiate the client.
$mgClient = Mailgun::create('PRIVATE_API_KEY', 'https://API_HOSTNAME');
$domain = "YOUR_DOMAIN_NAME";
$params = array(
      'from'    => 'Excited User <YOU@YOUR_DOMAIN_NAME>',
      'to'      => 'bob@example.com',
      'cc'      => 'alice@example.com',
      'bcc'     => 'john@example.com',
      'subject' => 'Hello',
      'text'    => 'Testing some Mailgun awesomness!',
      'html'    => '<html>HTML version of the body</html>',
      'attachment' => array(
              array(
                  'filePath' => 'test.txt',
                  'filename' => 'test_file.txt'
              )
          )
    );

# Make the call to the client.
$result = $mgClient->messages()->send($domain, $params);
```

```python
def send_complex_message():
    return requests.post(
        "https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/messages",
        auth=("api", "YOUR_API_KEY"),
        files=[("attachment", ("test.jpg", open("files/test.jpg","rb").read())),
               ("attachment", ("test.txt", open("files/test.txt","rb").read()))],
        data={"from": "Excited User <YOU@YOUR_DOMAIN_NAME>",
              "to": "foo@example.com",
              "cc": "baz@example.com",
              "bcc": "bar@example.com",
              "subject": "Hello",
              "text": "Testing some Mailgun awesomness!",
              "html": "<html>HTML version of the body</html>"})
```

```ruby
def send_complex_message
  data = {}
  data[:from] = "Excited User <YOU@YOUR_DOMAIN_NAME>"
  data[:to] = "foo@example.com"
  data[:cc] = "baz@example.com"
  data[:bcc] = "bar@example.com"
  data[:subject] = "Hello"
  data[:text] = "Testing some Mailgun awesomness!"
  data[:html] = "<html>HTML version of the body</html>"
  data[:attachment] = []
  data[:attachment] << File.new(File.join("files", "test.jpg"))
  data[:attachment] << File.new(File.join("files", "test.txt"))
  RestClient.post "https://api:YOUR_API_KEY"\
  "@api.mailgun.net/v3/YOUR_DOMAIN_NAME/messages", data
end
```

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class SendComplexMessageChunk
{

    public static void Main (string[] args)
    {
        Console.WriteLine (SendComplexMessage ().Content.ToString ());
    }

    public static IRestResponse SendComplexMessage ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.AddParameter ("domain", "YOUR_DOMAIN_NAME", ParameterType.UrlSegment);
        request.Resource = "{domain}/messages";
        request.AddParameter ("from", "Excited User <YOU@YOUR_DOMAIN_NAME>");
        request.AddParameter ("to", "foo@example.com");
        request.AddParameter ("cc", "baz@example.com");
        request.AddParameter ("bcc", "bar@example.com");
        request.AddParameter ("subject", "Hello");
        request.AddParameter ("text", "Testing some Mailgun awesomness!");
        request.AddParameter ("html",
                              "<html>HTML version of the body</html>");
        request.AddFile ("attachment", Path.Combine ("files", "test.jpg"));
        request.AddFile ("attachment", Path.Combine ("files", "test.txt"));
        request.Method = Method.POST;
        return client.Execute (request);
    }

}
```

```
import (
    "context"
    "github.com/mailgun/mailgun-go/v3"
    "time"
)

func SendComplexMessage(domain, apiKey string) (string, error) {
    mg := mailgun.NewMailgun(domain, apiKey)
    m := mg.NewMessage(
        "Excited User <YOU@YOUR_DOMAIN_NAME>",
        "Hello",
        "Testing some Mailgun awesomeness!",
        "foo@example.com",
    )
    m.AddCC("baz@example.com")
    m.AddBCC("bar@example.com")
    m.SetHtml("<html>HTML version of the body</html>")
    m.AddAttachment("files/test.jpg")
```

(continues on next page)

(continued from previous page)

```
    m.AddAttachment("files/test.txt")

    ctx, cancel := context.WithTimeout(context.Background(), time.Second*30)
    defer cancel()

    _, id, err := mg.Send(ctx, m)
    return id, err
}
```

```
const path = require('path');
var DOMAIN = 'YOUR_DOMAIN_NAME';
var api_key = 'YOUR_API_KEY';
var mailgun = require('mailgun-js')({ apiKey: api_key, domain: DOMAIN });

var filepath = path.join(__dirname, 'sample.jpg');

var data = {
  from: 'Excited User <me@samples.mailgun.org>',
  to: 'foo@example.com, baz@example.com, bar@example.com',
  cc: 'baz@example.com',
  bcc: 'bar@example.com',
  subject: 'Complex',
  text: 'Testing some Mailgun awesomness!',
  html: "<html>HTML version of the body</html>",
  attachment: filepath
};

mailgun.messages().send(data, function (error, body) {
  console.log(body);
});
```

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
```

```
import java.io.File;

import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode sendMIMEMessage() throws UnirestException {

        HttpResponse<JsonNode> request = Unirest.post("https://api.mailgun.net/v3/" +␣
↪YOUR_DOMAIN_NAME + "/messages.mime")
            .basicAuth("api", API_KEY)
            .header("content-type", "multipart/form-data;")
            .field("from", "Excited User <USER@YOURDOMAIN.COM>")
```

(continues on next page)

(continued from previous page)

```
            .field("to", "Megan@example.com")
            .field("subject", "Bah-weep-graaaaagnah wheep nini bong.")
            .field("message", new File("/temp/folder/file.mime"))
            .asJson();

        return request.getBody();
    }
}
```

```
# Include the Autoloader (see "Libraries" for install instructions)
require 'vendor/autoload.php';
use Mailgun\Mailgun;

# Instantiate the client.
$mgClient = Mailgun::create('PRIVATE_API_KEY', 'https://API_HOSTNAME');
$domain = "YOUR_DOMAIN_NAME";

$recipients = array(
    'bob@example.com',
    'alice@example.com',
    'john@example.com;
);
$params = array(
    'from' => 'Excited User <YOU@YOUR_DOMAIN_NAME>'
);

$mime_string = '<Pass fully formed MIME string here>'

# Make the call to the client.
$result = $mgClient->messages()->sendMime($domain, $recipients, $mime_string,
↪$params);
```

```
def send_mime_message():
    return requests.post(
        "https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/messages.mime",
        auth=("api", "YOUR_API_KEY"),
        data={"to": "bar@example.com"},
        files={"message": open("files/message.mime")})
```

```
def send_mime_message
  RestClient.post "https://api:YOUR_API_KEY"\
  "@api.mailgun.net/v3/YOUR_DOMAIN_NAME/messages.mime",
  :to => "bar@example.com",
  :message => File.new(File.join("files", "message.mime"))
end
```

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class SendMimeMessageChunk
{

    public static void Main (string[] args)
```

(continues on next page)

(continued from previous page)

```
    {
        Console.WriteLine (SendMimeMessage ().Content.ToString ());
    }

    public static IRestResponse SendMimeMessage ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.AddParameter ("domain", "YOUR_DOMAIN_NAME", ParameterType.UrlSegment);
        request.Resource = "{domain}/messages.mime";
        request.AddParameter ("to", "bar@example.com");
        request.AddFile ("message", Path.Combine ("files", "message.mime"));
        request.Method = Method.POST;
        return client.Execute (request);
    }

}
```

```
import (
    "context"
    "github.com/mailgun/mailgun-go/v3"
    "os"
    "time"
)

func SendMimeMessage(domain, apiKey string) (string, error) {
    mg := mailgun.NewMailgun(domain, apiKey)
    mimeMsgReader, err := os.Open("files/message.mime")
    if err != nil {
        return "", err
    }

    m := mg.NewMIMEMessage(mimeMsgReader, "bar@example.com")

    ctx, cancel := context.WithTimeout(context.Background(), time.Second*30)
    defer cancel()

    _, id, err := mg.Send(ctx, m)
    return id, err
}
```

```
var DOMAIN = 'YOUR_DOMAIN_NAME';
var mailgun = require('mailgun-js')({ apiKey: "YOUR_API_KEY", domain: DOMAIN });
var MailComposer = require('nodemailer/lib/mail-composer');

var mailOptions = {
  from: 'YOU@YOUR_DOMAIN_NAME',
  to: 'bob@example.com',
  subject: 'Hello',
  text: 'Testing some Mailgun awesomeness!'
};
```

(continues on next page)

(continued from previous page)

```
var mail = new MailComposer(mailOptions);

mail.compile().build(function(mailBuildError, message) {

    var dataToSend = {
        to: 'bob@example.com',
        message: message.toString('ascii')
    };

    mailgun.messages().sendMime(dataToSend, function (sendError, body) {
        if (sendError) {
            console.log(sendError);
            return;
        }
    });
});
```

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
```

```
import java.io.File;

import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode sendMessageNoTracking() throws UnirestException {

        HttpResponse<JsonNode> request = Unirest.post("https://api.mailgun.net/v3/" +␣
↪YOUR_DOMAIN_NAME + "/messages")
            .basicAuth("api", API_KEY)
            .field("from", "Excited User <YOU@YOUR_DOMAIN_NAME>")
            .field("to", "alice@example.com")
            .field("subject", "Hello")
            .field("text", "Testing some Mailgun awesomeness")
            .field("o:tracking", "False")
            .asJson();

        return request.getBody();
    }
}
```

```php
# Include the Autoloader (see "Libraries" for install instructions)
require 'vendor/autoload.php';
use Mailgun\Mailgun;

# Instantiate the client.
$mgClient = Mailgun::create('PRIVATE_API_KEY', 'https://API_HOSTNAME');
$domain = "YOUR_DOMAIN_NAME";

$params = array(
    'from'       => 'Excited User <YOU@YOUR_DOMAIN_NAME>',
    'to'         => 'foo@example.com',
    'subject'    => 'Hello',
    'text'       => 'Testing some Mailgun awesomness!',
    'o:tracking' => false
);

# Make the call to the client.
$result = $mgClient->messages()->send($domain, $params);
```

```python
def send_message_no_tracking():
    return requests.post(
        "https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/messages",
        auth=("api", "YOUR_API_KEY"),
        data={"from": "Excited User <YOU@YOUR_DOMAIN_NAME>",
              "to": ["bar@example.com", "baz@example.com"],
              "subject": "Hello",
              "text": "Testing some Mailgun awesomness!",
              "o:tracking": False})
```

```ruby
def send_message_no_tracking
  RestClient.post "https://api:YOUR_API_KEY"\
  "@api.mailgun.net/v3/YOUR_DOMAIN_NAME/messages",
  :from => "Excited User <YOU@YOUR_DOMAIN_NAME>",
  :to => "bar@example.com, baz@example.com",
  :subject => "Hello",
  :text => "Testing some Mailgun awesomness!",
  "o:tracking" => false
end
```

```csharp
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class SendMessageNoTrackingChunk
{

    public static void Main (string[] args)
    {
        Console.WriteLine (SendMessageNoTracking ().Content.ToString ());
    }

    public static IRestResponse SendMessageNoTracking ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
```

(continues on next page)

(continued from previous page)

```
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.AddParameter ("domain", "YOUR_DOMAIN_NAME", ParameterType.UrlSegment);
        request.Resource = "{domain}/messages";
        request.AddParameter ("from", "Excited User <YOU@YOUR_DOMAIN_NAME>");
        request.AddParameter ("to", "bar@example.com");
        request.AddParameter ("to", "baz@example.com");
        request.AddParameter ("subject", "Hello");
        request.AddParameter ("text", "Testing some Mailgun awesomness!");
        request.AddParameter ("o:tracking", false);
        request.Method = Method.POST;
        return client.Execute (request);
    }

}
```

```
import (
    "context"
    "github.com/mailgun/mailgun-go/v3"
    "time"
)

func SendMessageNoTracking(domain, apiKey string) (string, error) {
    mg := mailgun.NewMailgun(domain, apiKey)
    m := mg.NewMessage(
        "Excited User <YOU@YOUR_DOMAIN_NAME>",
        "Hello",
        "Testing some Mailgun awesomeness!",
        "foo@example.com",
    )
    m.SetTracking(false)

    ctx, cancel := context.WithTimeout(context.Background(), time.Second*30)
    defer cancel()

    _, id, err := mg.Send(ctx, m)
    return id, err
}
```

```
var mailgun = require("mailgun-js");
var api_key = 'YOUR_API_KEY';
var DOMAIN = 'YOUR_DOMAIN_NAME';
var mailgun = require('mailgun-js')({apiKey: api_key, domain: DOMAIN});

var data = {
  from: 'Excited User <me@samples.mailgun.org>',
  to: 'alice@example.com',
  subject: 'Hello',
  text: 'Testing some Mailgun awesomeness!',
  "o:tracking": 'False'
};

mailgun.messages().send(data, function (error, body) {
  console.log(body);
```

(continues on next page)

(continued from previous page)

```
});
```

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'
```

```
import java.io.File;

import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode sendScheduledMessage() throws UnirestException {

        HttpResponse<JsonNode> request = Unirest.post("https://api.mailgun.net/v3/" +␣
↪YOUR_DOMAIN_NAME + "/messages")
            .basicAuth("api", API_KEY)
            .field("from", "Excited User <USER@YOURDOMAIN.COM>")
            .field("to", "bruce@example")
            .field("subject", "Bah-weep-graaaaagnah wheep nini bong.")
            .field("text", "Testing some MailGun awesomeness")
            .field("o:deliverytime", "Sat, 20 May 2017 2:50:00 -0000")
            .asJson();

        return request.getBody();
    }
}
```

```
# Include the Autoloader (see "Libraries" for install instructions)
require 'vendor/autoload.php';
use Mailgun\Mailgun;

# Instantiate the client.
$mgClient = Mailgun::create('PRIVATE_API_KEY', 'https://API_HOSTNAME');
$domain = "YOUR_DOMAIN_NAME";
$params = array(
    'from'            => 'Excited User <YOU@YOUR_DOMAIN_NAME>',
    'to'              => 'bob@example.com',
    'subject'         => 'Hello',
    'text'            => 'Testing some Mailgun awesomness!',
    'o:deliverytime' => 'Wed, 01 Jan 2020 09:00:00 -0000'
);

# Make the call to the client.
$result = $mgClient->messages()->send($domain, $params);
```

```
def send_scheduled_message():
    return requests.post(
        "https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/messages",
        auth=("api", "YOUR_API_KEY"),
        data={"from": "Excited User <YOU@YOUR_DOMAIN_NAME>",
              "to": "bar@example.com",
              "subject": "Hello",
              "text": "Testing some Mailgun awesomness!",
              "o:deliverytime": "Fri, 25 Oct 2011 23:10:10 -0000"})
```

```
def send_scheduled_message
  RestClient.post "https://api:YOUR_API_KEY"\
  "@api.mailgun.net/v3/YOUR_DOMAIN_NAME/messages",
  :from => "Excited User <YOU@YOUR_DOMAIN_NAME>",
  :to => "bar@example.com",
  :subject => "Hello",
  :text => "Testing some Mailgun awesomeness!",
  "o:deliverytime" => "Fri, 25 Oct 2011 23:10:10 -0000"
end
```

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class SendScheduledMessageChunk
{

    public static void Main (string[] args)
    {
        Console.WriteLine (SendScheduledMessage ().Content.ToString ());
    }

    public static IRestResponse SendScheduledMessage ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.AddParameter ("domain", "YOUR_DOMAIN_NAME", ParameterType.UrlSegment);
        request.Resource = "{domain}/messages";
        request.AddParameter ("from", "Excited User <YOU@YOUR_DOMAIN_NAME>");
        request.AddParameter ("to", "bar@example.com");
        request.AddParameter ("subject", "Hello");
        request.AddParameter ("text", "Testing some Mailgun awesomness!");
        request.AddParameter ("o:deliverytime",
                              "Fri, 14 Oct 2011 23:10:10 -0000");
        request.Method = Method.POST;
        return client.Execute (request);
    }

}
```

```
import (
```

(continues on next page)

(continued from previous page)

```
    "context"
    "github.com/mailgun/mailgun-go/v3"
    "time"
)

func SendScheduledMessage(domain, apiKey string) (string, error) {
    mg := mailgun.NewMailgun(domain, apiKey)
    m := mg.NewMessage(
        "Excited User <YOU@YOUR_DOMAIN_NAME>",
        "Hello",
        "Testing some Mailgun awesomeness!",
        "bar@example.com",
    )
    m.SetDeliveryTime(time.Now().Add(5 * time.Minute))

    ctx, cancel := context.WithTimeout(context.Background(), time.Second*30)
    defer cancel()

    _, id, err := mg.Send(ctx, m)
    return id, err
}
```

```
const API_KEY = 'YOUR_API_KEY';
const DOMAIN = 'YOUR_DOMAIN_NAME';
const mailgun = require('mailgun-js')({apiKey: API_KEY, domain: DOMAIN});

const data = {
  from: 'Excited User <me@samples.mailgun.org>',
  to: 'bar@example.com',
  subject: 'Scheduled Message',
  text: 'Testing some Mailgun awesomeness!',
  "o:deliverytime": 'Fri, 6 Jul 2017 18:10:10 -0000'
};

mailgun.messages().send(data, (error, body) => {
  console.log(body);
});
```

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'
```

```
import java.io.File;

import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;
```

(continues on next page)

(continued from previous page)

```java
public class MGSample {

    // ...

    public static JsonNode sendTaggedMessage() throws UnirestException {

        HttpResponse<JsonNode> request = Unirest.post("https://api.mailgun.net/v3/" +
↪YOUR_DOMAIN_NAME + "/messages")
            .basicAuth("api", API_KEY)
            .field("from", "Excited User <YOU@YOUR_DOMAIN_NAME>")
            .field("to", "alice@example")
            .field("subject", "Hello.")
            .field("text", "Testing some Mailgun awesomeness")
            .field("o:tag", "newsletters")
            .field("o:tag", "September newsletter")
            .asJson();

        return request.getBody();
    }
}
```

```php
# Include the Autoloader (see "Libraries" for install instructions)
require 'vendor/autoload.php';
use Mailgun\Mailgun;

# Instantiate the client.
$mgClient = Mailgun::create('PRIVATE_API_KEY', 'https://API_HOSTNAME');
$domain = "YOUR_DOMAIN_NAME";
$params = array(
    'from'    => 'Excited User <YOU@YOUR_DOMAIN_NAME>',
    'to'      => 'bob@example.com',
    'subject' => 'Hello',
    'text'    => 'Testing some Mailgun awesomness!',
    'o:tag'   => array('Tag1', 'Tag2', 'Tag3')
);

# Make the call to the client.
$result = $mgClient->messages()->send($domain, $params);
```

```python
def send_tagged_message():
    return requests.post(
        "https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/messages",
        auth=("api", "YOUR_API_KEY"),
        data={"from": "Excited User <YOU@YOUR_DOMAIN_NAME>",
              "to": "bar@example.com",
              "subject": "Hello"
              "text": "Testing some Mailgun awesomness!",
              "o:tag": ["September newsletter", "newsletters"]})
```

```ruby
def send_tagged_message
  data = {}
  data[:from] = "Excited User <YOU@YOUR_DOMAIN_NAME>"
  data[:to] = "bar@example.com"
  data[:subject] = "Hello"
  data[:text] = "Testing some Mailgun awesomness!"
```

(continues on next page)

(continued from previous page)

```
  data["o:tag"] = []
  data["o:tag"] << "September newsletter"
  data["o:tag"] << "newsletters"
  RestClient.post "https://api:YOUR_API_KEY"\
  "@api.mailgun.net/v3/YOUR_DOMAIN_NAME/messages", data
end
```

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class SendTaggedMessageChunk
{

    public static void Main (string[] args)
    {
        Console.WriteLine (SendTaggedMessage ().Content.ToString ());
    }

    public static IRestResponse SendTaggedMessage ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.AddParameter ("domain", "YOUR_DOMAIN_NAME", ParameterType.UrlSegment);
        request.Resource = "{domain}/messages";
        request.AddParameter ("from", "Excited User <YOU@YOUR_DOMAIN_NAME>");
        request.AddParameter ("to", "bar@example.com");
        request.AddParameter ("subject", "Hello");
        request.AddParameter ("text", "Testing some Mailgun awesomness!");
        request.AddParameter ("o:tag", "September newsletter");
        request.AddParameter ("o:tag", "newsletters");
        request.Method = Method.POST;
        return client.Execute (request);
    }

}
```

```
import (
    "context"
    "github.com/mailgun/mailgun-go/v3"
    "time"
)

func SendTaggedMessage(domain, apiKey string) (string, error) {
    mg := mailgun.NewMailgun(domain, apiKey)
    m := mg.NewMessage(
        "Excited User <YOU@YOUR_DOMAIN_NAME>",
        "Hello",
        "Testing some Mailgun awesomeness!",
        "bar@example.com",
    )
```

(continues on next page)

(continued from previous page)

```
    err := m.AddTag("FooTag", "BarTag", "BlortTag")
    if err != nil {
        return "", err
    }

    ctx, cancel := context.WithTimeout(context.Background(), time.Second*30)
    defer cancel()

    _, id, err := mg.Send(ctx, m)
    return id, err
}
```

```
const API_KEY = 'YOUR_API_KEY';
const DOMAIN = 'YOUR_DOMAIN_NAME';
const mailgun = require('mailgun-js')({apiKey: API_KEY, domain: DOMAIN});

const data = {
  from: 'Excited User <me@samples.mailgun.org>',
  to: 'alice@example',
  subject: 'Tagged',
  text: 'Testing some Mailgun awesomeness!',
  "o:tag" : ['newsletters', 'September newsletter']
};

mailgun.messages().send(data, (error, body) => {
  console.log(body);
});
```

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
```

```
import java.io.File;

import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode sendConnection() throws UnirestException {

        HttpResponse<JsonNode> request = Unirest.post("https://api.mailgun.net/v3/" +␣
↪YOUR_DOMAIN_NAME + "/messages")
            .basicAuth("api", API_KEY)
```

(continues on next page)

(continued from previous page)

```
            .field("from", "Excited User <YOU@YOUR_DOMAIN_NAME>")
            .field("to", "alice@example.com")
            .field("to", "bob@example.com")
            .field("subject", "Hello")
            .field("text", "Testing out some Mailgun awesomeness!")
            .field("o:require-tls", "true")
            .field("o:skip-verification", "false")
            .asJson();

        return request.getBody();
    }
}
```

```
# Include the Autoloader (see "Libraries" for install instructions)
require 'vendor/autoload.php';
use Mailgun\Mailgun;

# Instantiate the client.
$mgClient = Mailgun::create('PRIVATE_API_KEY', 'https://API_HOSTNAME');
$domain = "YOUR_DOMAIN_NAME";
$params = array(
      'from'                => 'Excited User <YOU@YOUR_DOMAIN_NAME>',
      'to'                  => 'bob@example.com',
      'subject'             => 'Hello',
      'text'                => 'Testing some Mailgun awesomness!',
      'html'                => '<html>HTML version of the body</html>',
      'o:require-tls'       => true,
      'o:skip-verification' => false
    );

# Make the call to the client.
$result = $mgClient->messages()->send($domain, $params);
```

```
def send_require_tls():
    return requests.post(
        "https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/messages",
        auth=("api", "YOUR_API_KEY"),
        data={"from": "Excited User <YOU@YOUR_DOMAIN_NAME>",
              "to": ["bar@example.com", "baz@example.com"],
              "subject": "Hello",
              "text": "Testing some Mailgun awesomness!",
              "o:require-tls": True,
              "o:skip-verification": False})
```

```
def send_require_tls
  RestClient.post "https://api:YOUR_API_KEY"\
  "@api.mailgun.net/v3/YOUR_DOMAIN_NAME/messages",
  :from => "Excited User <YOU@YOUR_DOMAIN_NAME>",
  :to => "bar@example.com, baz@example.com",
  :subject => "Hello",
  :text => "Testing some Mailgun awesomness!",
  "o:require-tls" => true,
  "o:skip-verification" => false
end
```

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class SendConnectionChunk
{

    public static void Main (string[] args)
    {
        Console.WriteLine (SendWithTLS ().Content.ToString ());
    }

    public static IRestResponse SendWithTLS ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.AddParameter ("domain", "YOUR_DOMAIN_NAME", ParameterType.UrlSegment);
        request.Resource = "{domain}/messages";
        request.AddParameter ("from", "Excited User <YOU@YOUR_DOMAIN_NAME>");
        request.AddParameter ("to", "bar@example.com");
        request.AddParameter ("to", "baz@example.com");
        request.AddParameter ("subject", "Hello");
        request.AddParameter ("text", "Testing some Mailgun awesomness!");
        request.AddParameter ("o:require-tls", true);
        request.AddParameter ("o:skip-verification", false);
        request.Method = Method.POST;
        return client.Execute (request);
    }

}
```

```
import (
    "context"
    "github.com/mailgun/mailgun-go/v3"
    "time"
)

func SendWithConnectionOptions(domain, apiKey string) (string, error) {
    mg := mailgun.NewMailgun(domain, apiKey)
    m := mg.NewMessage(
        "Excited User <YOU@YOUR_DOMAIN_NAME>",
        "Hello",
        "Testing some Mailgun awesomeness!",
        "foo@example.com",
    )

    m.SetRequireTLS(true)
    m.SetSkipVerification(true)

    ctx, cancel := context.WithTimeout(context.Background(), time.Second*30)
    defer cancel()
```

(continues on next page)

(continued from previous page)

```
    _, id, err := mg.Send(ctx, m)
    return id, err
}
```

```
var mailgun = require("mailgun-js");
var api_key = 'YOUR_API_KEY';
var DOMAIN = 'YOUR_DOMAIN_NAME';
var mailgun = require('mailgun-js')({apiKey: api_key, domain: DOMAIN});

var data = {
  from: 'Excited User <me@samples.mailgun.org>',
  to: 'bar@example.com, baz@example.com',
  subject: 'Hello',
  text: 'Testing some Mailgun awesomeness!',
  "o:require-tls": 'True',
  "o:skip-verification": 'False'
};

mailgun.messages().send(data, function (error, body) {
  console.log(body);
});
```

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
```

```
import java.io.File;

import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode sendInlineImage() throws UnirestException {

        HttpResponse<JsonNode> request = Unirest.post("https://api.mailgun.net/v3/" +␣
↪YOUR_DOMAIN_NAME + "/messages")
            .basicAuth("api", API_KEY)
            .field("from", "Excited User <YOU@YOUR_DOMAIN_NAME>")
            .field("to", "alice@example.com")
```

(continues on next page)

(continued from previous page)

```
            .field("to", "bob@example.com")
            .field("cc", "joe@example.com")
            .field("subject", "Hello")
            .field("text", "Testing out some Mailgun awesomeness!")
            .field("html", "<html>Inline image here: <img src=\"cid:test.jpg\"></html>
↪")
            .field("inline", new File("/path/to/test.jpg"))
            .asJson();

        return request.getBody();
    }
}
```

```
# Include the Autoloader (see "Libraries" for install instructions)
require 'vendor/autoload.php';
use Mailgun\Mailgun;

# Instantiate the client.
$mgClient = Mailgun::create('PRIVATE_API_KEY', 'https://API_HOSTNAME');
$domain = "YOUR_DOMAIN_NAME";
$params = array(
      'from'    => 'Excited User <YOU@YOUR_DOMAIN_NAME>',
      'to'      => 'bob@example.com',
      'subject' => 'Hello',
      'text'    => 'Testing some Mailgun awesomness!',
      'html'    => '<html>Inline image: <img src="cid:test.jpg"></html>',
      'inline' => array(
          array('filePath' => '/path/to/test.jpg')
      )
);

 # Make the call to the client.
 $result = $mgClient->messages()->send($domain, $params);
```

```
def send_inline_image():
    return requests.post(
        "https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/messages",
        auth=("api", "YOUR_API_KEY"),
        files=[("inline", open("files/test.jpg"))],
        data={"from": "Excited User <YOU@YOUR_DOMAIN_NAME>",
              "to": "bar@example.com",
              "subject": "Hello",
              "text": "Testing some Mailgun awesomness!",
              "html": '<html>Inline image here: <img src="cid:test.jpg"></html>'})
```

```
def send_inline_image
  data = {}
  data[:from] = "Excited User <YOU@YOUR_DOMAIN_NAME>"
  data[:to] = "bar@example.com"
  data[:subject] = "Hello"
  data[:text] = "Testing some Mailgun awesomness!"
  data[:html] = '<html>Inline image here: <img src="cid:test.jpg"></html>'
  data[:inline] = File.new(File.join("files", "test.jpg"))
  RestClient.post "https://api:YOUR_API_KEY"\
  "@api.mailgun.net/v3/YOUR_DOMAIN_NAME/messages", data
end
```

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class SendInlineImageChunk
{

    public static void Main (string[] args)
    {
        Console.WriteLine (SendInlineImage ().Content.ToString ());
    }

    public static IRestResponse SendInlineImage ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.AddParameter ("domain", "YOUR_DOMAIN_NAME", ParameterType.UrlSegment);
        request.Resource = "{domain}/messages";
        request.AddParameter ("from", "Excited User <YOU@YOUR_DOMAIN_NAME>");
        request.AddParameter ("to", "baz@example.com");
        request.AddParameter ("subject", "Hello");
        request.AddParameter ("text", "Testing some Mailgun awesomness!");
        request.AddParameter ("html",
                              "<html>Inline image here: <img src=\"cid:test.jpg\"></
↪html>");
        request.AddFile ("inline", "files/test.jpg");
        request.Method = Method.POST;
        return client.Execute (request);
    }

}
```

```
import (
    "context"
    "github.com/mailgun/mailgun-go/v3"
    "time"
)

func SendInlineImage(domain, apiKey string) (string, error) {
    mg := mailgun.NewMailgun(domain, apiKey)
    m := mg.NewMessage(
        "Excited User <YOU@YOUR_DOMAIN_NAME>",
        "Hello",
        "Testing some Mailgun awesomeness!",
        "foo@example.com",
    )
    m.AddCC("baz@example.com")
    m.AddBCC("bar@example.com")
    m.SetHtml("<html>HTML version of the body</html>")
    m.AddInline("files/test.jpg")
    m.AddInline("files/test.txt")
```

(continues on next page)

(continued from previous page)

```
    ctx, cancel := context.WithTimeout(context.Background(), time.Second*30)
    defer cancel()

    _, id, err := mg.Send(ctx, m)
    return id, err
}
```

```
const path = require('path');
var DOMAIN = 'YOUR_DOMAIN_NAME';
var mailgun = require('mailgun-js')({ apiKey: "YOUR_API_KEY", domain: DOMAIN });

var filepath = path.join(__dirname, 'test.jpg');

var data = {
  from: 'Excited User <me@samples.mailgun.org>',
  to: 'foo@example.com, baz@example.com, bar@example.com',
  cc: 'baz@example.com',
  bcc: 'bar@example.com',
  subject: 'Hello',
  text: 'Testing some Mailgun awesomness!',
  html: '<html>Inline image here:<img src="cid:test.jpg"></html>',
  inline: filepath
};

mailgun.messages().send(data, function (error, body) {
  console.log(body);
});
```

2.3.2 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-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.
X-Mailgun-Delivery-Time-Optimize-Period	Toggles STO on a per-message basis. String should be set to the number of hours in [0- 9]+h format. See Sending a message with STO for details.
X-Mailgun-Time-Zone-Localize	Toggles TZO on a per-message basis. String should be set to preferred delivery time in HH:mm or hh:mmaa format, where HH:mm is used for 24 hour format without AM/PM and hh:mmaa is used for 12 hour format with AM/PM. See Sending a message with TZO for details.

2.3.3 Sending a message with STO

Mailgun's Send Time Optimization (STO) feature uses machine learning to analyze engagement data (opens and clicks) for a recipient to determine when a user is most engaged with their messages. If we have enough engagement data to make a determination of when a user is most engaged, Mailgun will hold onto the message and deliver it during that optimal period. The idea here is that if we can deliver a message to a user when they are most engaged, the message will be towards the top and is more likely to be engaged with.

You can send a message using STO via API by passing in the parameter `o:deliverytime-optimize-period` or via SMTP using the MIME header `X-Mailgun-Delivery-Time-Optimize-Period`. The value should be a string in the `[0-9]+h` format. This value defines the window in which Mailgun will run the optimization algorithm against the data we have and deliver the message. We recommend using a minimum value of `24h` for best results, and the max value is `72h`.

Please note that STO is only available on certain plans. See www.mailgun.com/pricing for more info.

2.3.4 Sending a message with TZO

Mailgun's Timezone Optimization feature allows senders to schedule messages to be delivered in a recipient's local timezone. Similar to how our message scheduling works, with TZO you pass your desired delivery time, and Mailgun will convert that to a user's local timezone, if we have data for that recipient. If we do not have data for that recipient, the message will be delivered immediately.

Timezones are estimated based on a user's IP address. Mailgun collects the IP address on click events, and uses a geolocation service to translate the IP address into a timezone for the user. We store that timezone in a database (hashed, of course), so that when TZO is used on a message, Mailgun will look up the timezone for that user, and schedule the message for the delivery time in that user's local timezone.

You can send a message using TZO via API by passing in the parameter `o:time-zone-localize` or via SMTP using the MIME header `X-Mailgun-Time-Zone-Localize`. The value (String) should be set to preferred delivery time in HH:mm or hh:mmaa format, where HH:mm is used for 24 hour format without AM/PM and hh:mmaa is used for 12 hour format with AM/PM.

Please note that TZO is only available on certain plans. See www.mailgun.com/pricing for more info.

2.3.5 Sending an AMP message

Google's Accelerated Mobile Pages (AMP) for Email allows senders to include a subset of AMP components within an email message. This lets recipients receive content rich, dynamic emails, that result in a more interactive experience with email messages. For recipients, this means the ability to view real time inventory, submit updates or replies in a doc, respond to surveys, etc., all from directly within the email message. For marketers, this could mean improved conversion rates since users can interact with you directly from the email without visiting a website!

AMP Requirements

While AMP is a really exciting email tool, it takes a bit of setup before you can successfully send an AMP email message to your recipients.

Registration

In order to send AMP emails to mailboxes that support it (Gmail for now), you'll need to register your sending domain with Google.

Content

Your AMP email content must comply with Google's requirements. First, you need to ensure that you're following Google's Bulk Senders Guidelines.

Next, It's important to follow the Amp for Email specification when building your AMP messages, specifically the required markup, AMP components, and CSS requirements. One of the gotchas you may run into, for example, is the <img> tag is replaced with <amp-img>. As you go along, you can use Gmail's AMP for Email Playground to test whether your message content will pass the validation process.

HTTPS URLs

All URLs must use HTTPS, including tracking and unsubscribe URLs. If you're using Mailgun for your open/click tracking and unsubscribe URLs, you'll need to follow these steps to enable HTTPS on your Mailgun tracking URLs.

Sending an AMP email on Mailgun

Mailgun has made it easy to send an AMP email using our API by providing the optional `amp-html` parameter along with your AMP content. Mailgun will take care of building the proper `text/x-amp-html` MIME portion. As long

as you're following the AMP requirements set by Google, you should be well on your way to sending your AMP messages.

Testing your email messages

If you're waiting on Google for your domain to be registered, you can still start building AMP emails and testing them. Visit your Gmail settings page (GSuite users will need their admins to enable the Dynamic Email option), and then under the *Dynamic Email* section, check the box to *Enable dynamic email*. After that, click on *Developer settings* and enter your sending address in the field in order to whitelist your sending address, then click OK. As long as you're following the proper requirements for sending AMP messages, you should be able to successfully receive an AMP email from your sending address to your Gmail account.

AMP Best Practices

MIME part organization

Some email clients will only render the last MIME part, so you should ensure your MIME parts are in the following order:

- `text/plain`
- `text/x-amp-html`
- `text/html`

If you send a message via our API using the `amp-html`, Mailgun will take care of the proper ordering.

Text / HTML fallback

Just like when you send an HTML email, you should provide a Text version as a fallback in case the recipient blocks HTML content. The same applies when sending AMP emails. If you send an AMP email to a recipient that doesn't support it or has blocked the content, they can still view the message in another format.

Replying to and forwarding AMP messages

It's important to note that email clients will strip out the `text/x-amp-html` MIME in your messages when you reply to or forward the message. This is another reason why you should ensure you have text and HTML versions as a fallback when you send your emails.

2.3.6 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.

2.3.7 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%'
```

```
import java.io.File;

import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...
```

(continues on next page)

(continued from previous page)

```
    public static JsonNode sendTemplateMessage() throws UnirestException {

        HttpResponse<JsonNode> request = Unirest.post("https://api.mailgun.net/v3/" +␣
↪YOUR_DOMAIN_NAME + "/messages")
            .basicAuth("api", API_KEY)
            .field("from", "Excited User <YOU@YOUR_DOMAIN_NAME>")
            .field("to", "alice@example.com")
            .field("to", "bob@example.com")
            .field("Subject", "Hello, %recipient.first%!")
            .field("text", "If you wish to unsubscribe, click <https://mailgun.com/
↪unsubscribe/%recipient.id%>")
            .field("recipient-variables", "{\"bob@example.com\": {\"first\":\"Bob\", \
↪"id\":1}, \"alice@example.com\": {\"first\":\"Alice\", \"id\": 2}}")
            .asJson();

        return request.getBody();
    }
}
```

```
# Include the Autoloader (see "Libraries" for install instructions)
require 'vendor/autoload.php';
use Mailgun\Mailgun;

# Instantiate the client.
$mgClient = Mailgun::create('PRIVATE_API_KEY', 'https://API_HOSTNAME');
$domain = "YOUR_DOMAIN_NAME";
$params =  array(
    'from'    => 'Excited User <YOU@YOUR_DOMAIN_NAME>',
    'to'      => array('bob@example.com, alice@example.com'),
    'subject' => 'Hey %recipient.first%',
    'text'    => 'If you wish to unsubscribe, click http://example.com/unsubscribe/
↪%recipient.id%',
    'recipient-variables' => '{"bob@example.com": {"first":"Bob", "id":1},
                               "alice@example.com": {"first":"Alice", "id": 2}}'
);

# Make the call to the client.
$result = $mgClient->messages()->send($domain, $params);
```

```
def send_template_message():
    return requests.post(
        "https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/messages",
        auth=("api", "YOUR_API_KEY"),
        data={"from": "Excited User <YOU@YOUR_DOMAIN_NAME>",
              "to": ["alice@example.com, bob@example.com"],
              "subject": "Hey, %recipient.first%",
              "text": "If you wish to unsubscribe, click http://mailgun/unsubscribe/
↪%recipient.id%'",
              "recipient-variables": ('{"bob@example.com": {"first":"Bob", "id":1}, '
                                      '"alice@example.com": {"first":"Alice", "id": 2}
↪}')})
```

```
def send_template_message
  RestClient.post "https://api:YOUR_API_KEY"\
```

(continues on next page)

(continued from previous page)

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

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class SendTemplateMessageChunk
{

    public static void Main (string[] args)
    {
        Console.WriteLine (SendTemplateMessage ().Content.ToString ());
    }

    public static IRestResponse SendTemplateMessage ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.AddParameter ("domain", "YOUR_DOMAIN_NAME", ParameterType.UrlSegment);
        request.Resource = "{domain}/messages";
        request.AddParameter ("from", "Excited User <YOU@YOUR_DOMAIN_NAME>");
        request.AddParameter ("to", "alice@example.com");
        request.AddParameter ("to", "bob@example.com");
        request.AddParameter ("subject", "Hey, %recipient.first%");
        request.AddParameter ("text",
                              "If you wish to unsubscribe, click http://mailgun/
↪unsubscribe/%recipient.id%'");
        request.AddParameter ("recipient-variables",
                              "{\"bob@example.com\": {\"first\":\"Bob\", \"id\":1}, \
↪"alice@example.com\": {\"first\":\"Alice\", \"id\": 2}}");
        request.Method = Method.POST;
        return client.Execute (request);
    }

}
```

```
import (
    "context"
    "github.com/mailgun/mailgun-go/v3"
    "time"
)

func SendTemplateMessage(domain, apiKey string) (string, error) {
```

(continues on next page)

(continued from previous page)

```
    mg := mailgun.NewMailgun(domain, apiKey)
    m := mg.NewMessage(
        "Excited User <YOU@YOUR_DOMAIN_NAME>",
        "Hey %recipient.first%",
        "If you wish to unsubscribe, click http://mailgun/unsubscribe/%recipient.id%",
    ) // IMPORTANT: No To:-field recipients!

    m.AddRecipientAndVariables("bob@example.com", map[string]interface{}{
        "first": "bob",
        "id":    1,
    })

    m.AddRecipientAndVariables("alice@example.com", map[string]interface{}{
        "first": "alice",
        "id":    2,
    })

    ctx, cancel := context.WithTimeout(context.Background(), time.Second*30)
    defer cancel()

    _, id, err := mg.Send(ctx, m)
    return id, err
}
```

```
var DOMAIN = 'YOUR_DOMAIN_NAME';
var mailgun = require('mailgun-js')({ apiKey: "YOUR_API_KEY", domain: DOMAIN });

var data = {
  from: 'Excited User <me@samples.mailgun.org>',
  to: 'alice@example.com, bob@example.com',
  subject: 'Hey %recipient.first%',
  text: 'If you wish to unsubscribe, click http://mailgun/unsubscribe/%recipient.id%',
      'recipient-variables': '{"alice@example.com": {"first":"Alice", "id":1},
↪"bob@example.com":{"first":"Bob", "id":2}}'
};

mailgun.messages().send(data, function (error, body) {
  console.log(body);
});
```

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

Example:

```
X-Mailgun-Recipient-Variables: {"bob@example.com": {"first":"Bob", "id":1},
↪"alice@example.com": {"first":"Alice", "id": 2}}
From: me@example.com
To: %recipient%
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
```

(continues on next page)

(continued from previous page)

```
Subject: (rackspace) Hello
 MSK 2012 support@mailgunhq.com %recipient.description%
Message-Id: <20120726154302.29322.40670@definebox.com>

support@mailgunhq.com %recipient.description%

--8686cc907910484e9d21c54776cd791c--
```

2.3.8 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 Lists` 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'
```

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode createMailingList() throws UnirestException {

        HttpResponse <JsonNode> request = Unirest.post("https://api.mailgun.net/v3/
↪lists")
            .basicAuth("api", API_KEY)
            .field("address", "LIST@YOUR_DOMAIN_NAME")
            .field("description", "LIST_DESCRIPTION")
            .asJson();

        return request.getBody();
    }
}
```

```php
# Include the Autoloader (see "Libraries" for install instructions)
require 'vendor/autoload.php';
use Mailgun\Mailgun;

# Instantiate the client.
$mgClient = Mailgun::create('PRIVATE_API_KEY', 'https://API_HOSTNAME');
$mailing_list     = 'LIST@YOUR_DOMAIN_NAME';
$list_name        = 'Mailgun Subscribers';
$list_description = 'News and service updates';
$access_level     = 'readonly';

# Issue the call to the client.
$result = $mgClient->mailingList()->create($mailing_list, $list_name, $list_
↪description, $access_level);
```

```python
def create_mailing_list():
    return requests.post(
        "https://api.mailgun.net/v3/lists",
        auth=('api', 'YOUR_API_KEY'),
        data={'address': 'LIST@YOUR_DOMAIN_NAME',
              'description': "Mailgun developers list"})
```

```ruby
def create_mailing_list
  RestClient.post("https://api:YOUR_API_KEY" \
                  "@api.mailgun.net/v3/lists",
                  :address => 'LIST@YOUR_DOMAIN_NAME',
                  :description => "Mailgun developers list")
end
```

```csharp
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class CreateMailingListChunk
{

    public static void Main (string[] args)
    {
        Console.WriteLine (CreateMailingList ().Content.ToString ());
    }

    public static IRestResponse CreateMailingList ()
    {

        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.Resource = "lists";
        request.AddParameter ("address", "LIST@YOUR_DOMAIN_NAME");
        request.AddParameter ("description", "Mailgun developers list");
        request.Method = Method.POST;
        return client.Execute (request);
```

(continues on next page)

(continued from previous page)

```
    }

}
```

```
import (
    "context"
    "github.com/mailgun/mailgun-go/v3"
    "time"
)

func CreateMailingList(domain, apiKey string) (mailgun.MailingList, error) {
    mg := mailgun.NewMailgun(domain, apiKey)

    ctx, cancel := context.WithTimeout(context.Background(), time.Second*30)
    defer cancel()

    return mg.CreateMailingList(ctx, mailgun.MailingList{
        Address:     "list@example.com",
        Name:        "dev",
        Description: "Mailgun developers list.",
        AccessLevel: mailgun.AccessLevelMembers,
    })
}
```

```
var DOMAIN = 'YOUR_DOMAIN_NAME';
var mailgun = require('mailgun-js')({ apiKey: "YOUR_API_KEY", domain: DOMAIN });

mailgun.post('/lists', {"address": `list_name@${DOMAIN}`, "description": "list_
↪description"}, function (error, body) {
  console.log(body);
});
```

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}'
```

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode addListMember() throws UnirestException {

        HttpResponse <JsonNode> request = Unirest.post("https://api.mailgun.net/v3/
↪lists/{list}@{domain}/members"
```

(continues on next page)

(continued from previous page)

```
            .basicAuth("api", API_KEY)
            .field("subscribed", true)
            .field("address", "bob@example.com")
            .field("name", "Bob Bar")
            .field("description", "developer")
            .field("vars", "{\"age\": 26}")
            .asJson();

        return request.getBody();
    }
}
```

```
# Include the Autoloader (see "Libraries" for install instructions)
require 'vendor/autoload.php';
use Mailgun\Mailgun;

# Instantiate the client.
$mgClient = Mailgun::create('PRIVATE_API_KEY', 'https://API_HOSTNAME');
$mailing_list = 'LIST@YOUR_DOMAIN_NAME';
$address ='bob@example.com';
$name = 'Bob';
$vars = array("id" => "123456");

# Issue the call to the client.
$result = $mgClient->mailingList()->member()->create(
                                                $mailing_list,
                                                $address,
                                                $name,
                                                $vars);
```

```
def add_list_member():
    return requests.post(
        "https://api.mailgun.net/v3/lists/LIST@YOUR_DOMAIN_NAME/members",
        auth=('api', 'YOUR_API_KEY'),
        data={'subscribed': True,
              'address': 'bar@example.com',
              'name': 'Bob Bar',
              'description': 'Developer',
              'vars': '{"age": 26}'})
```

```
def add_list_member
  RestClient.post("https://api:YOUR_API_KEY" \
                  "@api.mailgun.net/v3/lists/LIST@YOUR_DOMAIN_NAME/members",
                  :subscribed => true,
                  :address => 'bar@example.com',
                  :name => 'Bob Bar',
                  :description => 'Developer',
                  :vars => '{"age": 26}')
end
```

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;
```

(continues on next page)

(continued from previous page)

```
public class AddListMemberChunk
{

    public static void Main (string[] args)
    {
        Console.WriteLine (AddListMember ().Content.ToString ());
    }

    public static IRestResponse AddListMember ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.Resource = "lists/{list}/members";
        request.AddParameter ("list", "LIST@YOUR_DOMAIN_NAME",
                              ParameterType.UrlSegment);
        request.AddParameter ("address", "bar@example.com");
        request.AddParameter ("subscribed", true);
        request.AddParameter ("name", "Bob Bar");
        request.AddParameter ("description", "Developer");
        request.AddParameter ("vars", "{\"age\": 26}");
        request.Method = Method.POST;
        return client.Execute (request);
    }

}
```

```
import (
    "context"
    "github.com/mailgun/mailgun-go/v3"
    "time"
)

func AddListMember(domain, apiKey string) error {
    mg := mailgun.NewMailgun(domain, apiKey)

    memberJoe := mailgun.Member{
        Address:    "joe@example.com",
        Name:       "Joe Example",
        Subscribed: mailgun.Subscribed,
    }

    ctx, cancel := context.WithTimeout(context.Background(), time.Second*30)
    defer cancel()

    return mg.CreateMember(ctx, true, "mailingList@example.com", memberJoe)
}
```

```
var DOMAIN = 'YOUR_DOMAIN_NAME';
var mailgun = require('mailgun-js')({ apiKey: "YOUR_API_KEY", domain: DOMAIN });

var list = mailgun.lists(`mylist@${DOMAIN}`);
```

(continues on next page)

(continued from previous page)

```
var bob = {
  subscribed: true,
  address: 'bob@example.com',
  name: 'Bob Barr',
  vars: {age: 34}
};

list.members().create(bob, function (error, data) {
  console.log(data);
});
```

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.

Replying to a list

You can set the preferred method for where a reply to the list should go:

- `list` Replies to the list go to the list address. This is the default setting for any new list created, except for read-only lists, where replies can only go to the sender. Reply-all will still go to the list.
- `sender` Replies to the list go to the sender (FROM) address. This is the default and only option for read-only lists.

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.

2.3.9 Templates

Mailgun allows you to store predefined templates via *Template API* and use them to send messages via *sending api* just providing template name. Don't be confused with *Template Variables* as **Templating** works independently.

Mailgun's templates uses the very popular template engine handlebars (Only v3.0 is supported right now). To provide values for substitution you have to use *Attaching Data to Messages*. Let's see how to send a message using the template feature:

First of all a template has to be stored:

```
curl -s --user 'api:YOUR_API_KEY' -X POST \
  https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/templates \
  -F template='<div class="entry"> <h1>{{title}}</h1> <div class="body"> {{body}} </
↪div> </div>' \
  -F name = 'template.test'
  -F description='Sample template'
```

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...
```

(continues on next page)

(continued from previous page)

```
    public static JsonNode createTemplate() throws UnirestException {

        HttpResponse <JsonNode> request = Unirest.post("https://api.mailgun.net/v3/"␣
↪+ YOUR_DOMAIN_NAME + "/templates")
            .basicAuth("api", API_KEY)
            .field("template", "<div class=\"entry\"> <h1>{{title}}</h1> <div class=\
↪"body\"> {{body}} </div> </div>")
            .field("name", "Test template")
            .field("description", "Sample template")
            .asJson();

        return request.getBody();
    }
}
```

```
# Currently, the PHP SDK does not support the Templates endpoint.
# Consider using the following php curl function.
function add_template() {
  $params = array(
    'template'    => '<div class="entry"> <h1>{{title}}</h1> <div class="body"> {
↪{body}} </div> </div>',
    'name'        => 'Test template',
    'description' => 'sample_template'
  );

  $ch = curl_init();

  curl_setopt($ch, CURLOPT_HTTPAUTH, CURLAUTH_BASIC);
  curl_setopt($ch, CURLOPT_USERPWD, 'api:PRIVATE_API_KEY');
  curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);

  curl_setopt($ch, CURLOPT_CUSTOMREQUEST, 'POST');
  curl_setopt($ch, CURLOPT_URL, 'https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/templates
↪');
  curl_setopt($ch, CURLOPT_POSTFIELDS, $params);

  $result = curl_exec($ch);
  curl_close($ch);

  return $result;
}
```

```
def add_template():
    return requests.post(
        "https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/templates",
        auth=("api", "YOUR_API_KEY"),
        data={'template': '<div class="entry"> <h1>{{title}}</h1> <div class="body"> {
↪{body}} </div> </div>',
              'name': 'Test template',
              'description': 'Sample template'})
```

```
def add_template
  RestClient.post "https://api:YOUR_API_KEY"\
  "@api.mailgun.net/v3/YOUR_DOMAIN_NAME/templates",
```

(continues on next page)

(continued from previous page)

```
  :template => '<div class="entry"> <h1>{{title}}</h1> <div class="body"> {{body}} </
↪div> </div>',
  :name => 'Test template',
  :description => 'Sample template'
end
```

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class CreateTemplatesChunk
{

    public static void Main (string[] args)
    {
        Console.WriteLine (CreateTemplate ().Content.ToString ());
    }

    public static IRestResponse CreateTemplate ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.Resource = "{domain}/templates";
        request.AddParameter ("domain", "YOUR_DOMAIN_NAME", ParameterType.UrlSegment);
        request.AddParameter ("template", "<div class=\"entry\"> <h1>{{title}}</h1>
↪<div class=\"body\"> {{body}} </div> </div>");
        request.AddParameter ("description", "Sample template");
        request.AddParameter ("name", "Test template");
        request.Method = Method.POST;
        return client.Execute (request);
    }

}
```

```
import (
    "context"
    "github.com/mailgun/mailgun-go/v3"
    "time"
)

func SendMessageWithTemplate(domain, apiKey string) error {
    mg := mailgun.NewMailgun(domain, apiKey)
    var err error

    ctx, cancel := context.WithTimeout(context.Background(), time.Second*30)
    defer cancel()

    // Create a new template
    err = mg.CreateTemplate(ctx, &mailgun.Template{
        Name: "my-template",
        Version: mailgun.TemplateVersion{
```

(continues on next page)

(continued from previous page)

```
            Template: `'<div class="entry"> <h1>{{.title}}</h1> <div class="body"> {{.
↪body}} </div> </div>'`,
            Engine:   mailgun.TemplateEngineGo,
            Tag:      "v1",
        },
    })
    if err != nil {
        return err
    }

    // Give time for template to show up in the system.
    time.Sleep(time.Second * 1)

    // Create a new message with template
    m := mg.NewMessage("Excited User <excited@example.com>", "Template example", "")
    m.SetTemplate("my-template")

    // Add recipients
    m.AddRecipient("bob@example.com")
    m.AddRecipient("alice@example.com")

    // Add the variables to be used by the template
    m.AddVariable("title", "Hello Templates")
    m.AddVariable("body", "Body of the message")

    _, id, err := mg.Send(ctx, m)
    return err
}
```

```
var DOMAIN = 'YOUR_DOMAIN_NAME';
var mailgun = require('mailgun-js')({ apiKey: "YOUR_API_KEY", domain: DOMAIN });

mailgun.post(`/${DOMAIN}/templates`,
             {"template" : "<div class=\"entry\"> <h1>{{title}}</h1> <div class=\
↪"body\"> {{body}} </div> </div>",
              "name": "template.test"
              "description": "Sample template"},
                            function (error, body) {
                                    console.log(body);
                            }
             );
```

Response returns stored template information:

```
{
  "template": {
    "createdAt": "Wed, 29 Aug 2018 23:31:13 UTC",
    "description": "Sample template",
    "name": "template.test",
  },
  "message": "template has been stored"
}
```

Just using the template name you can send a message:

```
curl -s --user 'api:YOUR_API_KEY' \
```

(continues on next page)

(continued from previous page)

```
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 template='template.test' \
-F h:X-Mailgun-Variables='{"title": "API documentation", "body": "Sending messages␣
↪with templates"}'
```

```
import java.io.File;

import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode sendMessageByTemplateId() throws UnirestException {

        HttpResponse<JsonNode> request = Unirest.post("https://api.mailgun.net/v3/" +␣
↪YOUR_DOMAIN_NAME + "/messages")
            .basicAuth("api", API_KEY)
            .field("from", "Excited User <YOU@YOUR_DOMAIN_NAME>")
            .field("to", "alice@example.com")
            .field("subject", "Hello")
            .field("template", "template.test")
            .field("o:tracking", "False")
            .field("h:X-Mailgun-Variables", "{\"title\": \"API Documentation\", \"body\
↪": \"Sending messages with templates\"}")
            .asJson();

        return request.getBody();
    }
}
```

```
# Include the Autoloader (see "Libraries" for install instructions)
require 'vendor/autoload.php';
use Mailgun\Mailgun;

# Instantiate the client.
$mgClient = Mailgun::create('PRIVATE_API_KEY', 'https://API_HOSTNAME');
$domain = "YOUR_DOMAIN_NAME";
$params = array(
    'from'                  => 'Excited User <YOU@YOUR_DOMAIN_NAME>',
    'to'                    => 'bob@example.com',
    'subject'               => 'Hello',
    'template'              => 'template.test',
    'h:X-Mailgun-Variables' => '{"title": "API Documentation", "body": "Sending␣
↪messages with templates"}'
    );

# Make the call to the client.
$result = $mgClient->messages()->send($domain, $params);
```

```python
def send_message_by_template_id():
    return requests.post(
        "https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/messages",
        auth=("api", "YOUR_API_KEY"),
        data={"from": "Excited User <YOU@YOUR_DOMAIN_NAME>",
              "to": ["bar@example.com", "baz@example.com"],
              "subject": "Hello",
              "template": "template.test",
              "h:X-Mailgun-Variables": json.dumps({"title": "API documentation", "body
↪": "Sending messages with templates"}))
```

```ruby
def send_message_by_template_id
  RestClient.post "https://api:YOUR_API_KEY"\
  "@api.mailgun.net/v3/YOUR_DOMAIN_NAME/messages",
  :from => "Excited User <YOU@YOUR_DOMAIN_NAME>",
  :to => "bar@example.com, baz@example.com",
  :subject => "Hello",
  :template => "template.test",
  :"h:X-Mailgun-Variables" => '{"title": "API Documentation", "body": "Sending␣
↪messages with template"}'
end
```

```c#
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class SendMessageByTemplateIdChunk
{

    public static void Main (string[] args)
    {
        Console.WriteLine (SendMessageByTemplateId ().Content.ToString ());
    }

    public static IRestResponse SendMessageByTemplateId ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.AddParameter ("domain", "YOUR_DOMAIN_NAME", ParameterType.UrlSegment);
        request.Resource = "{domain}/messages";
        request.AddParameter ("from", "Excited User <YOU@YOUR_DOMAIN_NAME>");
        request.AddParameter ("to", "bar@example.com");
        request.AddParameter ("to", "baz@example.com");
        request.AddParameter ("subject", "Hello");
        request.AddParameter ("template", "template.test");
        request.AddParameter ("h:X-Mailgun-Variables", "{\"title\": \"API␣
↪Documentation\", \"body\": \"Sending messages with templates\"}");
        request.Method = Method.POST;
        return client.Execute (request);
    }

}
```

```
import (
  "context"
  "encoding/json"
  "github.com/mailgun/mailgun-go"
  "time"
)


func SendMessageWithTemplate() (id string, err error) {
  mg := mailgun.NewMailgun("YOUR_DOMAIN_NAME", "YOUR_API_KEY")
  ctx, cancel := context.WithTimeout(context.Background(), time.Second * 30)
  defer cancel()

  m := mg.NewMessage("Excited User <YOU@YOUR_DOMAIN_NAME>", "???", "")
  m.SetTemplate("template.test")
  if err := m.AddRecipient("bar@example.com"); err != nil {
    return "", err
  }

  vars, err := json.Marshal(map[string]string{
      "title": "API Documentation",
      "body":  "Sending messages with templates",
  })
  if err != nil {
    return "", err
  }
  m.AddHeader("X-Mailgun-Variables", vars)

  _, id, err := mg.Send(ctx, m)
  return
}
```

```
var mailgun = require("mailgun-js");
var api_key = 'YOUR_API_KEY';
var DOMAIN = 'YOUR_DOMAIN_NAME';
var mailgun = require('mailgun-js')({apiKey: api_key, domain: DOMAIN});

var data = {
  from: 'Excited User <me@samples.mailgun.org>',
  to: 'alice@example.com',
  subject: 'Hello',
  template: 'template.test',
  h:X-Mailgun-Variables: '{"title": "API Documentation", "body": "Sending messages␣
↪with templates"}'
};

mailgun.messages().send(data, function (error, body) {
  console.log(body);
});
```

Another way to provide variables is to use a form parameter prefixed with `v:` how it's shown *here*.

```
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' \
```

(continues on next page)

(continued from previous page)

```
-F template='template.test' \
-F v:title='API documentation' \
-F v:body='Sending messages with templates'
```

```
import java.io.File;

import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode sendMessageByTemplateId() throws UnirestException {

        HttpResponse<JsonNode> request = Unirest.post("https://api.mailgun.net/v3/" +␣
↪YOUR_DOMAIN_NAME + "/messages")
            .basicAuth("api", API_KEY)
            .field("from", "Excited User <YOU@YOUR_DOMAIN_NAME>")
            .field("to", "alice@example.com")
            .field("subject", "Hello")
            .field("template", "template.test")
            .field("o:tracking", "False")
            .field("v:title", "API Documentation")
            .field("v:body", "Sending messages with templates")
            .asJson();

        return request.getBody();
    }
}
```

```
# Include the Autoloader (see "Libraries" for install instructions)
require 'vendor/autoload.php';
use Mailgun\Mailgun;

# Instantiate the client.
$mgClient = new Mailgun('YOUR_API_KEY');
$domain = "YOUR_DOMAIN_NAME";
$params = array(
    'from'     => 'Excited User <YOU@YOUR_DOMAIN_NAME>',
    'to'       => 'foo@example.com',
    'subject'  => 'Hello',
    'template' => 'template.test',
    'v:title'  => 'API Documentation',
    'v:body'   => 'Sending messages with templates'
);
# Make the call to the client.
$result = $mgClient->messages()->send($domain, $params);
```

```
def send_message_by_template_id():
    return requests.post(
        "https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/messages",
        auth=("api", "YOUR_API_KEY"),
```

(continues on next page)

(continued from previous page)

```
        data={"from": "Excited User <YOU@YOUR_DOMAIN_NAME>",
              "to": ["bar@example.com", "baz@example.com"],
              "subject": "Hello",
              "template": "template.test",
              "v:title": "API Documentation",
              "v:body": "Sending messages with templates"})
```

```
def send_message_by_template_id
  RestClient.post "https://api:YOUR_API_KEY"\
  "@api.mailgun.net/v3/YOUR_DOMAIN_NAME/messages",
  :from => "Excited User <YOU@YOUR_DOMAIN_NAME>",
  :to => "bar@example.com, baz@example.com",
  :subject => "Hello",
  :template => "template.test",
  :"v:title" => "API Documentation",
  :"v:body" => "Sending messages with templates"
end
```

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class SendMessageByTemplateIdChunk
{

    public static void Main (string[] args)
    {
        Console.WriteLine (SendMessageByTemplateId ().Content.ToString ());
    }

    public static IRestResponse SendMessageByTemplateId ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.AddParameter ("domain", "YOUR_DOMAIN_NAME", ParameterType.UrlSegment);
        request.Resource = "{domain}/messages";
        request.AddParameter ("from", "Excited User <YOU@YOUR_DOMAIN_NAME>");
        request.AddParameter ("to", "bar@example.com");
        request.AddParameter ("to", "baz@example.com");
        request.AddParameter ("subject", "Hello");
        request.AddParameter ("template", "template.test");
        request.AddParameter ("v:title", "API Documentation");
        request.AddParameter ("v:body", "Sending messages with templates");
        request.Method = Method.POST;
        return client.Execute (request);
    }

}
```

```
// Not implemented yet
```

```
var mailgun = require("mailgun-js");
var api_key = 'YOUR_API_KEY';
var DOMAIN = 'YOUR_DOMAIN_NAME';
var mailgun = require('mailgun-js')({apiKey: api_key, domain: DOMAIN});

var data = {
  from: 'Excited User <me@samples.mailgun.org>',
  to: 'alice@example.com',
  subject: 'Hello',
  template: 'template.test',
  v:title: 'API Documentation',
  v:body: 'Sending messages with templates'
};

mailgun.messages().send(data, function (error, body) {
  console.log(body);
});
```

The second way is not recomended as it's limited to simple key value data. If you have arrays, dictionaries in values or complex json data you have to supply variables via `X-Mailgun-Variables` header.

Handlebars

Speaking of Handlebars, one of the cool things you can do with Handelbars is use their block helpers, which are easy ways to implement dynamic content in your template. Our implementation of Handlebars supports the following helpers: if, unless, each, with, lookup, log and equal. Let's explore what each of these do and some quick examples:

The `if` *block helper*

The `if` block helper will allow you to conditionally render a block in your template. For example, if you wanted to use a template that would dynamically change language body, you would include the following in your HTML:

```
{{#if english}}
<p>This text is in the English language.</p>
{{else if spanish}}
<p>Este texto está en idioma español.</p>
{{else if french}}
<p>Ce texte est en langue française.</p>
{{/if}}
```

In order to send the spanish version, for example, you would pass the `h:X-Mailgun-Variables` parameter with the following JSON data: {"spanish" : "true"}

The `unless` *block helper*

The `unless` helper is essentially the inverse of the `if` helper. The block will only be rendered if the expression returns a false value. Include the following in your HTML:

```
{{#unless paid}}
<h3 class="warning">WARNING: Your account is past due and will be suspended shortly.␣
↪Please contact our billing department for assistance</h3>
{{/unless}}
```

An example JSON payload would look like this: {"paid" : "false"}

The `each` *block helper*

Using the `each` helper, you can iterate through a list. Include the following in your HTML:

```
{{#each user.services}}
<li>You scheduled {{this.service}} on {{this.date}}</li>
{{/each}}
```

Your JSON data could look something like this:

```
{
 "user":
  {
    "services":
    [
        {
           "date":"07/30/2019"
           "service":"deliverability consultation"
        },
        {
           "date":"08/05/2019",
           "service":"sales consultation"
        }
    ]
  }
}
```

The email would end up looking like this:

- You scheduled deliverability consultation on 07/30/2019
- You scheduled sales consultation on 08/05/2019

2.3.10 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'
```

```
import java.io.File;

import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
```

(continues on next page)

(continued from previous page)

```
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode sendScheduledMessage() throws UnirestException {

        HttpResponse<JsonNode> request = Unirest.post("https://api.mailgun.net/v3/" +
↪YOUR_DOMAIN_NAME + "/messages")
            .basicAuth("api", API_KEY)
            .field("from", "Excited User <USER@YOURDOMAIN.COM>")
            .field("to", "bruce@example")
            .field("subject", "Bah-weep-graaaaagnah wheep nini bong.")
            .field("text", "Testing some MailGun awesomeness")
            .field("o:deliverytime", "Sat, 20 May 2017 2:50:00 -0000")
            .asJson();

        return request.getBody();
    }
}
```

```
# Include the Autoloader (see "Libraries" for install instructions)
require 'vendor/autoload.php';
use Mailgun\Mailgun;

# Instantiate the client.
$mgClient = Mailgun::create('PRIVATE_API_KEY', 'https://API_HOSTNAME');
$domain = "YOUR_DOMAIN_NAME";
$params = array(
    'from'           => 'Excited User <YOU@YOUR_DOMAIN_NAME>',
    'to'             => 'bob@example.com',
    'subject'        => 'Hello',
    'text'           => 'Testing some Mailgun awesomness!',
    'o:deliverytime' => 'Wed, 01 Jan 2020 09:00:00 -0000'
);

# Make the call to the client.
$result = $mgClient->messages()->send($domain, $params);
```

```
def send_scheduled_message():
    return requests.post(
        "https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/messages",
        auth=("api", "YOUR_API_KEY"),
        data={"from": "Excited User <YOU@YOUR_DOMAIN_NAME>",
              "to": "bar@example.com",
              "subject": "Hello",
              "text": "Testing some Mailgun awesomness!",
              "o:deliverytime": "Fri, 25 Oct 2011 23:10:10 -0000"})
```

```
def send_scheduled_message
  RestClient.post "https://api:YOUR_API_KEY"\
  "@api.mailgun.net/v3/YOUR_DOMAIN_NAME/messages",
  :from => "Excited User <YOU@YOUR_DOMAIN_NAME>",
```

(continues on next page)

(continued from previous page)

```
  :to => "bar@example.com",
  :subject => "Hello",
  :text => "Testing some Mailgun awesomeness!",
  "o:deliverytime" => "Fri, 25 Oct 2011 23:10:10 -0000"
end
```

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class SendScheduledMessageChunk
{

    public static void Main (string[] args)
    {
        Console.WriteLine (SendScheduledMessage ().Content.ToString ());
    }

    public static IRestResponse SendScheduledMessage ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.AddParameter ("domain", "YOUR_DOMAIN_NAME", ParameterType.UrlSegment);
        request.Resource = "{domain}/messages";
        request.AddParameter ("from", "Excited User <YOU@YOUR_DOMAIN_NAME>");
        request.AddParameter ("to", "bar@example.com");
        request.AddParameter ("subject", "Hello");
        request.AddParameter ("text", "Testing some Mailgun awesomness!");
        request.AddParameter ("o:deliverytime",
                              "Fri, 14 Oct 2011 23:10:10 -0000");
        request.Method = Method.POST;
        return client.Execute (request);
    }

}
```

```
import (
    "context"
    "github.com/mailgun/mailgun-go/v3"
    "time"
)

func SendScheduledMessage(domain, apiKey string) (string, error) {
    mg := mailgun.NewMailgun(domain, apiKey)
    m := mg.NewMessage(
        "Excited User <YOU@YOUR_DOMAIN_NAME>",
        "Hello",
        "Testing some Mailgun awesomeness!",
        "bar@example.com",
    )
    m.SetDeliveryTime(time.Now().Add(5 * time.Minute))
```

(continues on next page)

(continued from previous page)

```
    ctx, cancel := context.WithTimeout(context.Background(), time.Second*30)
    defer cancel()

    _, id, err := mg.Send(ctx, m)
    return id, err
}
```

```
const API_KEY = 'YOUR_API_KEY';
const DOMAIN = 'YOUR_DOMAIN_NAME';
const mailgun = require('mailgun-js')({apiKey: API_KEY, domain: DOMAIN});

const data = {
  from: 'Excited User <me@samples.mailgun.org>',
  to: 'bar@example.com',
  subject: 'Scheduled Message',
  text: 'Testing some Mailgun awesomeness!',
  "o:deliverytime": 'Fri, 6 Jul 2017 18:10:10 -0000'
};

mailgun.messages().send(data, (error, body) => {
  console.log(body);
});
```

2.3.11 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.

2.4 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 *Tagging*.

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 the `Suppressions` tab of the Control Panel or their respective APIs (*Bounces API*, *Unsubscribes API*, *Complaints API*). Data is stored indefinitely.

- Access statistics aggregated by tags in the `Analytics` tab of the Control Panel or the *Stats API*. Data is stored for at least 6 months.
- 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 a per domain basis under the **Tracking Settings** section of a particular domain's settings 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).

2.4.1 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
list_member_uploaded	This event occurs after successfully adding a member to a mailing list.
list_member_upload_error	This event occurs if an error occurs adding a member to a mailing list.
list_uploaded	This event occurs after successfully uploading a large list of members to a mailing list.

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
```

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode getLogs() throws UnirestException {

        HttpResponse<JsonNode> request = Unirest.get("https://api.mailgun.net/v3/" +␣
↪YOUR_DOMAIN_NAME + "/events")
            .basicAuth("api", API_KEY)
            .queryString("begin", "Thurs, 18 May 2017 09:00:00 -0000")
            .queryString("ascending", "yes")
            .queryString("limit", 1)
            .asJson();

        return request.getBody();
    }
}
```

```
# Include the Autoloader (see "Libraries" for install instructions)
require 'vendor/autoload.php';
use Mailgun\Mailgun;

# Instantiate the client.
$mgClient    = Mailgun::create('PRIVATE_API_KEY', 'https://API_HOSTNAME');
$domain      = 'YOUR_DOMAIN_NAME';
$queryString = array(
    'begin'        => 'Wed, 1 Jan 2020 09:00:00 -0000',
    'ascending'    => 'yes',
    'limit'        =>  25,
    'pretty'       => 'yes',
    'recipient'    => 'bob@example.com'
);

# Issue the call to the client.
$result = $mgClient->events()->get($domain, $queryString);
```

```
def get_logs():
    return requests.get(
        "https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/events",
        auth=("api", "YOUR_API_KEY"),
```

(continues on next page)

(continued from previous page)

```
        params={"begin"       : "Fri, 3 May 2013 09:00:00 -0000",
                "ascending"   : "yes",
                "limit"       :  25,
                "pretty"      : "yes",
                "recipient"   : "joe@example.com"})
```

```
def get_logs
  RestClient.get "https://api:YOUR_API_KEY"\
  "@api.mailgun.net/v3/YOUR_DOMAIN_NAME/events",
   :params => {
    :'begin'       => 'Fri, 3 May 2013 09:00:00 -0000',
    :'ascending'   => 'yes',
    :'limit'       =>  25,
    :'pretty'      => 'yes',
    :'recipient' => 'joe@example.com'
   }
end
```

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class EventsDateTimeRecipientChunk
{

    public static void Main (string[] args)
    {
        Console.WriteLine (EventsDateTimeRecipient ().Content.ToString ());
    }

    public static IRestResponse EventsDateTimeRecipient ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.AddParameter ("domain", "YOUR_DOMAIN_NAME", ParameterType.UrlSegment);
        request.Resource = "{domain}/events";
        request.AddParameter ("begin", "Fri, 3 May 2013 09:00:00 -0000");
        request.AddParameter ("ascending", "yes");
        request.AddParameter ("limit", 25);
        request.AddParameter ("pretty", "yes");
        request.AddParameter ("recipient", "joe@example.com");
        return client.Execute (request);
    }

}
```

```
import (
    "context"
    "github.com/mailgun/mailgun-go/v3"
    "time"
```

(continues on next page)

(continued from previous page)

```
}

func PrintEventLog(domain, apiKey string) error {
    mg := mailgun.NewMailgun(domain, apiKey)

    // Create an iterator
    it := mg.ListEvents(&mailgun.ListEventOptions{
        Begin: time.Now().Add(-50 * time.Minute),
        Limit: 100,
        Filter: map[string]string{
            "recipient": "joe@example.com",
        },
    })

    ctx, cancel := context.WithTimeout(context.Background(), time.Second*30)
    defer cancel()

    // Iterate through all the pages of events
    var page []mailgun.Event
    for it.Next(ctx, &page) {
        for _, event := range page {
            fmt.Printf("%+v\n", event)
        }
    }

    // Did iteration end because of an error?
    if it.Err() != nil {
        return it.Err()
    }

    return nil
}
```

```
var DOMAIN = 'YOUR_DOMAIN_NAME';
var mailgun = require('mailgun-js')({ apiKey: "YOUR_API_KEY", domain: DOMAIN });

mailgun.get(`/${DOMAIN}/events`, {"begin": "Thurs, 06 July 2017 09:00:00 -0000",
↪"ascending": "yes", "limit": 1},  function (error, body) {
  console.log(body);
});
```

Sample response:

```
{
  "items": [
    {
      "tags": [],
      "timestamp": 1376325780.160809,
      "envelope": {
        "sender": "me@samples.mailgun.org",
        "transport": ""
      },
      "event": "accepted",
      "campaigns": [],
      "user-variables": {},
      "flags": {
```

(continues on next page)

(continued from previous page)


```
        "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..."
  }
}
```


2.4.2 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 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.

Payload

When something has happened to your email, your URL will be called with application/json payload and with the following data:

```
{
  "signature":
  {
    "timestamp": "1529006854",
    "token": "a8ce0edb2dd8301dee6c2405235584e45aa91d1e9f979f3de0",
    "signature": "d2271d12299f6592d9d44cd9d250f0704e4674c30d79d07c47a66f95ce71cf55"
  }
  "event-data":
  {
    "event": "opened",
    "timestamp": 1529006854.329574,
    "id": "DACSsAdVSeGpLid7TN03WA",
    // ...
  }
}
```

The 'signature' parameters are described in *securing webhooks* and the 'event-data' parameters are the same as described in *Event Structure*

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 Webhook Signing 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 (version 3.x.x) code sample used to verify the signature:

```
import hashlib, hmac

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

And here's a sample in Ruby:

```ruby
require 'openssl'

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

And here's a sample in PHP:

```php
function verify($signingKey, $token, $timestamp, $signature)
{
    // check if the timestamp is fresh
    if (\abs(\time() - $timestamp) > 15) {
        return false;
    }

    // returns true if signature is valid
    return \hash_equals(\hash_hmac('sha256', $timestamp . $token, $signingKey),
↪$signature);
}
```

And here's a sample in Go

```go
import (
    "github.com/mailgun/mailgun-go/v3"
)

func VerifyWebhookSignature(domain, signingKey, timestamp, token, signature string)␣
↪(bool, error) {
    mg := mailgun.NewMailgun(domain, signingKey)

    return mg.VerifyWebhookSignature(mailgun.Signature{
        TimeStamp: timestamp,
        Token:     token,
        Signature: signature,
    })
}
```

And here's a sample in Node.js

```nodejs
const crypto = require('crypto')

const verify = ({ signingKey, timestamp, token, signature }) => {
    const encodedToken = crypto
        .createHmac('sha256', signingKey)
        .update(timestamp.concat(token))
        .digest('hex')

    return (encodedToken === signature)
}
```

2.4.3 Attaching Data to Messages

When sending you can attach data to your messages for later retrieval, you can for instance attach campaign identifiers or recipient identifiers to messages to help relate webhook payloads or events retrieved from mailgun back to marketing campaigns or individual recipients in your system.

There are two methods of attaching data to emails. If you are sending email via SMTP, you can attach data by providing a `X-Mailgun-Variables` header. The header data must be in JSON map format. For example:

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

Multiple `X-Mailgun-Variables` headers may be provided and their map values will be combined.

Note: The value of the "X-Mailgun-Variables" header must 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.

If you are sending email via the HTTP API, you can attach data by providing a form parameter prefixed with `v:`. For example:

```
v:first_name=John
v:last_name=Smith
v:my_message_id=123
```

The data provided will be included the recipients email via a header called `X-Mailgun-Variables`. Additionally the data will also be available via webhook payloads and events returned from the events API. The data will be attached to these payloads via the `user-variables` field as a JSON map. For Example:

```
{
    "event": "delivered",
    "user-variables": {
        "first_name": "John",
        "last_name:" "Smith",
        "my_message_id": "123"
    }
}
```

When sending batches of emails, you can use values from recipient variables to provide a custom variable per recipient using templating. For example given a variable of `v:recipient-id=%recipient.id%` and a recipient variable of `{"user1@example.com" : { "id": 123 }}` events and webhooks associated with the recipient `user1@example.com` will contain a `user-variable` field with the content of `{ "recipient-id": "123" }`

2.4.4 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 your messages, they will be aggregated by these tags.

The application of tags is more useful in tandem with our tracking features. Tags are unique to each send and can be used to collect data on different message distributions being sent out (e.g. how many recipients opened messages of a given tag or clicks on linked URLs in messages of a tag). This provides the ability to review the overall performance of tags as well as gives the ability to compare one tag against another. For example, two messages of similar content can be assigned different tags for analysis of which message type had better engagement, more recipient activity, etc.

Note: By default, each account is allowed a maximum of 4,000 tags. Any tags created after the 4,000 tag limit are dropped. If more tags are needed, please contact our support team by creating a support ticket here.

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'
```

```
import java.io.File;

import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode sendTaggedMessage() throws UnirestException {

        HttpResponse<JsonNode> request = Unirest.post("https://api.mailgun.net/v3/" +␣
↪YOUR_DOMAIN_NAME + "/messages")
            .basicAuth("api", API_KEY)
            .field("from", "Excited User <YOU@YOUR_DOMAIN_NAME>")
            .field("to", "alice@example")
            .field("subject", "Hello.")
            .field("text", "Testing some Mailgun awesomeness")
            .field("o:tag", "newsletters")
            .field("o:tag", "September newsletter")
            .asJson();

        return request.getBody();
    }
}
```

```
# Include the Autoloader (see "Libraries" for install instructions)
require 'vendor/autoload.php';
use Mailgun\Mailgun;

# Instantiate the client.
$mgClient = Mailgun::create('PRIVATE_API_KEY', 'https://API_HOSTNAME');
$domain = "YOUR_DOMAIN_NAME";
$params = array(
    'from'    => 'Excited User <YOU@YOUR_DOMAIN_NAME>',
    'to'      => 'bob@example.com',
    'subject' => 'Hello',
    'text'    => 'Testing some Mailgun awesomness!',
```

(continues on next page)

(continued from previous page)

```
    'o:tag'    => array('Tag1', 'Tag2', 'Tag3')
);

# Make the call to the client.
$result = $mgClient->messages()->send($domain, $params);
```

```
def send_tagged_message():
    return requests.post(
        "https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/messages",
        auth=("api", "YOUR_API_KEY"),
        data={"from": "Excited User <YOU@YOUR_DOMAIN_NAME>",
              "to": "bar@example.com",
              "subject": "Hello",
              "text": "Testing some Mailgun awesomness!",
              "o:tag": ["September newsletter", "newsletters"]})
```

```
def send_tagged_message
  data = {}
  data[:from] = "Excited User <YOU@YOUR_DOMAIN_NAME>"
  data[:to] = "bar@example.com"
  data[:subject] = "Hello"
  data[:text] = "Testing some Mailgun awesomness!"
  data["o:tag"] = []
  data["o:tag"] << "September newsletter"
  data["o:tag"] << "newsletters"
  RestClient.post "https://api:YOUR_API_KEY"\
  "@api.mailgun.net/v3/YOUR_DOMAIN_NAME/messages", data
end
```

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class SendTaggedMessageChunk
{

    public static void Main (string[] args)
    {
        Console.WriteLine (SendTaggedMessage ().Content.ToString ());
    }

    public static IRestResponse SendTaggedMessage ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.AddParameter ("domain", "YOUR_DOMAIN_NAME", ParameterType.UrlSegment);
        request.Resource = "{domain}/messages";
        request.AddParameter ("from", "Excited User <YOU@YOUR_DOMAIN_NAME>");
        request.AddParameter ("to", "bar@example.com");
        request.AddParameter ("subject", "Hello");
```

(continues on next page)

(continued from previous page)

```
        request.AddParameter("text", "Testing some Mailgun awesomness!");
        request.AddParameter("o:tag", "September newsletter");
        request.AddParameter("o:tag", "newsletters");
        request.Method = Method.POST;
        return client.Execute(request);
    }

}
```

```
import (
    "context"
    "github.com/mailgun/mailgun-go/v3"
    "time"
)

func SendTaggedMessage(domain, apiKey string) (string, error) {
    mg := mailgun.NewMailgun(domain, apiKey)
    m := mg.NewMessage(
        "Excited User <YOU@YOUR_DOMAIN_NAME>",
        "Hello",
        "Testing some Mailgun awesomeness!",
        "bar@example.com",
    )

    err := m.AddTag("FooTag", "BarTag", "BlortTag")
    if err != nil {
        return "", err
    }

    ctx, cancel := context.WithTimeout(context.Background(), time.Second*30)
    defer cancel()

    _, id, err := mg.Send(ctx, m)
    return id, err
}
```

```
const API_KEY = 'YOUR_API_KEY';
const DOMAIN = 'YOUR_DOMAIN_NAME';
const mailgun = require('mailgun-js')({apiKey: API_KEY, domain: DOMAIN});

const data = {
  from: 'Excited User <me@samples.mailgun.org>',
  to: 'alice@example',
  subject: 'Tagged',
  text: 'Testing some Mailgun awesomeness!',
  "o:tag" : ['newsletters', 'September newsletter']
};

mailgun.messages().send(data, (error, body) => {
  console.log(body);
});
```

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.

2.4.5 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 counters of opens aggregated by tags in the `Analytics` 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 in the **Tracking Settings** section of your domain's settings page in the `Domains` tab of your Control Panel or by 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 Verification & DNS** section, which is also located in your domain's settings page in the `Domains` 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.

Opens Webhook

You can specify webhook URLs programmatically using the *Webhooks* API. When a user opens one of your emails, your `opened` URLs will be called with the following *webhooks payload*.

2.4.6 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 counters of clicks aggregated by tags in the `Analytics` 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 in the **Tracking Settings** section of your domain's settings page in the `Domains` tab of your Control Panel or by 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 **Domain Verification & DNS** section, which is also located in your domain's settings page 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 webhook URLs programmatically using the *Webhooks* API. Every time a user clicks on a link inside of your messages, your `clicked` URLs will be called with the following *webhooks payload*.

2.4.7 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 unsubscribes happen in the `Logs` tab or see counters of unsubscribes aggregated by tags in the `Analytics` 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 programmatically 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 webhook URLs programmatically using the *Webhooks* API. When a user unsubscribes, Mailgun will invoke `unsubscribed` webhook with the following *webhooks payload*.

2.4.8 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 counters of complaints, aggregated by tags, in the `Analytics` 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 Complains Webhook

You can specify webhook URLs programmatically using the *Webhooks* API. When a user reports one of your emails as spam, Mailgun will invoke `complained` webhook with the following *webhooks payload*.

2.4.9 Tracking Failures

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

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 "Bounces" table in the `Suppressions` tab of your Control Panel 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 "Bounces" table in the `Suppressions` 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 counters of bounces, aggregated by tags, in the `Analytics` 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.

Permanent Failure Webhook

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 *webhooks payload* to your `permanent_fail` URLs. You can specify webhook URLs programmatically using the *Webhooks* API.

Temporary Failure Webhook

If Mailgun got a Soft bounce (temporary failure) we will POST the following *webhooks payload* to your `temporary_fail` URLs. You can specify webhooks URLs programmatically using the *Webhooks* API.

2.4.10 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

You can specify a webhook URL programmatically using the *Webhooks* API to be notified every time a message is delivered. If the message is successfully delivered to the intended recipient, we will POST the following *webhooks payload* to your `delivered` URLs

2.4.11 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*

2.5 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.

2.5.1 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)

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 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.
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:

```
def mailgun_posted_params
  message_headers = JSON.parse(params["message-headers"])
  message_headers.each do |header|
    key, value = header
    puts "header key: #{key}, header value: #{value}"
  end
end
```

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>".
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.
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.
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()'
```

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode createRoute() throws UnirestException {

        HttpResponse <JsonNode> request = Unirest.post("https://api.mailgun.net/v3/
↪routes")
            .basicAuth("api", API_KEY)
            .field("priority", "0")
            .field("description", "sample route")
            .field("expression", "match_recipient('.*@YOUR_DOMAIN_NAME')")
            .field("action", "forward('http://myhost.com/messages/')")
            .field("action", "stop()")
            .asJson();

        return request.getBody();
    }
}
```

```
# Include the Autoloader (see "Libraries" for install instructions)
require 'vendor/autoload.php';
use Mailgun\Mailgun;

# Instantiate the client.
```

(continues on next page)

(continued from previous page)

```
$mgClient = Mailgun::create('PRIVATE_API_KEY', 'https://API_HOSTNAME');

# Define your expression, actions, and description
$expression  = 'match_recipient(".*@mg.example.com")';
$actions     = array('forward("my_address@example.com")', 'stop()');
$description = 'Catch All and Forward';

# Issue the call to the client.
$result = $mgClient->routes()->create($expression, $actions, $description);
```

```
def create_route():
    return requests.post(
        "https://api.mailgun.net/v3/routes",
        auth=("api", "YOUR_API_KEY"),
        data={"priority": 0,
              "description": "Sample route",
              "expression": "match_recipient('.*@YOUR_DOMAIN_NAME')",
              "action": ["forward('http://myhost.com/messages/')", "stop()"]})
```

```
def create_route
  data = {}
  data[:priority] = 0
  data[:description] = "Sample route"
  data[:expression] = "match_recipient('.*@YOUR_DOMAIN_NAME')"
  data[:action] = []
  data[:action] << "forward('http://myhost.com/messages/')"
  data[:action] << "stop()"
  RestClient.post "https://api:YOUR_API_KEY"\
  "@api.mailgun.net/v3/routes", data
end
```

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class CreateRouteChunk
{

    public static void Main (string[] args)
    {
        Console.WriteLine (CreateRoute ().Content.ToString ());
    }

    public static IRestResponse CreateRoute ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.Resource = "routes";
        request.AddParameter ("priority", 0);
        request.AddParameter ("description", "Sample route");
```

(continues on next page)

(continued from previous page)

```
        request.AddParameter("expression", "match_recipient('.*@YOUR_DOMAIN_NAME')");
        request.AddParameter("action",
                             "forward('http://myhost.com/messages/')");
        request.AddParameter("action", "stop()");
        request.Method = Method.POST;
        return client.Execute(request);
    }

}
```

```
import (
    "context"
    "github.com/mailgun/mailgun-go/v3"
    "time"
)

func CreateRoute(domain, apiKey string) (mailgun.Route, error) {
    mg := mailgun.NewMailgun(domain, apiKey)

    ctx, cancel := context.WithTimeout(context.Background(), time.Second*30)
    defer cancel()

    return mg.CreateRoute(ctx, mailgun.Route{
        Priority:    1,
        Description: "Sample Route",
        Expression:  "match_recipient(\".*@YOUR_DOMAIN_NAME\")",
        Actions: []string{
            "forward(\"http://example.com/messages/\")",
            "stop()",
        },
    })
}
```

```
var DOMAIN = 'YOUR_DOMAIN_NAME';
var mailgun = require('mailgun-js')({ apiKey: "YOUR_API_KEY", domain: DOMAIN });

mailgun.post('/routes', {"priority": 0, "description": 'Sample route', "expression":
↪'match_recipient(".*@YOUR_DOMAIN_NAME")', "action": 'forward("http://myhost.com/
↪messages/")', "action": 'stop()'}, function (error, body) {
  console.log(body);
});
```

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"
```

(continues on next page)

(continued from previous page)

```
  }
}
```

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
```

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode getRoutes() throws UnirestException {

        HttpResponse <JsonNode> request = Unirest.get("https://api.mailgun.net/v3/
↪routes")
            .basicAuth("api", API_KEY)
            .queryString("skip", "0")
            .queryString("limit","5")
            .asJson();

        return request.getBody();
    }
}
```

```
# Include the Autoloader (see "Libraries" for install instructions)
require 'vendor/autoload.php';
use Mailgun\Mailgun;

# Instantiate the client.
$mgClient = Mailgun::create('PRIVATE_API_KEY', 'https://API_HOSTNAME');

# Issue the call to the client.
$result = $mgClient->routes()->index();
```

```
def get_routes():
    return requests.get(
        "https://api.mailgun.net/v3/routes",
        auth=("api", "YOUR_API_KEY"),
        params={"skip": 1,
                "limit": 1})
```

```
def get_routes
  RestClient.get "https://api:YOUR_API_KEY"\
```

(continues on next page)

(continued from previous page)

```
  "@api.mailgun.net/v3/routes", :params => {
    :skip => 1,
    :limit => 1
  }
end
```

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class GetRoutesChunk
{

    public static void Main (string[] args)
    {
        Console.WriteLine (GetRoutes ().Content.ToString ());
    }

    public static IRestResponse GetRoutes ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.Resource = "routes";
        request.AddParameter ("skip", 1);
        request.AddParameter ("limit", 1);
        return client.Execute (request);
    }

}
```

```
import (
    "context"
    "github.com/mailgun/mailgun-go/v3"
    "time"
)

func ListRoutes(domain, apiKey string) ([]mailgun.Route, error) {
    mg := mailgun.NewMailgun(domain, apiKey)
    it := mg.ListRoutes(nil)

    ctx, cancel := context.WithTimeout(context.Background(), time.Second*30)
    defer cancel()

    var page, result []mailgun.Route
    for it.Next(ctx, &page) {
        result = append(result, page...)
    }

    if it.Err() != nil {
        return nil, it.Err()
    }
```

(continues on next page)

(continued from previous page)

```
    return result, nil
}
```

```
var DOMAIN = 'YOUR_DOMAIN_NAME';
var mailgun = require('mailgun-js')({ apiKey: "YOUR_API_KEY", domain: DOMAIN });

mailgun.get('/routes', {"skip": 0, "limit": 5}, function (error, body) {
  console.log(body);
});
```

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
```

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode getSingleRoute() throws UnirestException {

        HttpResponse <JsonNode> request = Unirest.get("https://api.mailgun.net/v3/
↪routes/YOUR_ROUTE_ID")
            .basicAuth("api", API_KEY)
            .asJson();

        return request.getBody();
    }
}
```

```
# Include the Autoloader (see "Libraries" for install instructions)
```

(continues on next page)

(continued from previous page)

```
require 'vendor/autoload.php';
use Mailgun\Mailgun;

# Instantiate the client.
$mgClient = Mailgun::create('PRIVATE_API_KEY', 'https://API_HOSTNAME');
$route_id = '5d9fde0fd8b861ec16cf2549'

# Issue the call to the client.
$result = $mgClient->routes()->show($route_id);
```

```
def get_route():
    return requests.get(
        "https://api.mailgun.net/v3/routes/4e97c1b2ba8a48567f007fb6",
        auth=("api", "YOUR_API_KEY"))
```

```
def get_route
  RestClient.
    get("https://api:YOUR_API_KEY"\
        "@api.mailgun.net/v3/routes/"\
        "4e97c1b2ba8a48567f007fb6"){|response, request, result| response }
end
```

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class GetRouteChunk
{

    public static void Main (string[] args)
    {
        Console.WriteLine (GetRoute ().Content.ToString ());
    }

    public static IRestResponse GetRoute ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.Resource = "routes/{id}";
        request.AddUrlSegment ("id", "4e97c1b2ba8a48567f007fb6");
        return client.Execute (request);
    }


}
```

```
import (
    "context"
    "github.com/mailgun/mailgun-go/v3"
    "time"
)
```

(continues on next page)

(continued from previous page)

```
func GetRoute(domain, apiKey string) (mailgun.Route, error) {
    mg := mailgun.NewMailgun(domain, apiKey)

    ctx, cancel := context.WithTimeout(context.Background(), time.Second*30)
    defer cancel()

    return mg.GetRoute(ctx, "route_id")
}
```

```
var DOMAIN = 'YOUR_DOMAIN_NAME';
var mailgun = require('mailgun-js')({ apiKey: "YOUR_API_KEY", domain: DOMAIN });

mailgun.get('/routes/your_route_id', function (error, body) {
  console.log(body);
});
```

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"
  }
}
```

2.5.2 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
```

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...
```

(continues on next page)

(continued from previous page)

```
    public static JsonNode getCredentials() throws UnirestException {

        HttpResponse<JsonNode> request = Unirest.get("https://api.mailgun.net/v3/
↪domains/" + YOUR_DOMAIN_NAME + "/credentials")
            .basicAuth("api", API_KEY)
            .asJson();

        return request.getBody();
    }
}
```

```
# Include the Autoloader (see "Libraries" for install instructions)
require 'vendor/autoload.php';
use Mailgun\Mailgun;

# Instantiate the client.
$mgClient = Mailgun::create('PRIVATE_API_KEY', 'https://API_HOSTNAME');
$domain = "YOUR_DOMAIN_NAME";

# Issue the call to the client.
$result = $mgClient->domains()->credentials($domain);
```

```
def get_credentials():
    return requests.get(
        "https://api.mailgun.net/v3/domains/YOUR_DOMAIN_NAME/credentials",
        auth=("api", "YOUR_API_KEY"))
```

```
def get_credentials
  RestClient.get "https://api:YOUR_API_KEY"\
  "@api.mailgun.net/v3/domains/YOUR_DOMAIN_NAME/credentials"
end
```

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class GetCredentialsChunk
{

    public static void Main (string[] args)
    {
        Console.WriteLine (GetCredentials ().Content.ToString ());
    }

    public static IRestResponse GetCredentials ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.AddParameter ("domain", "YOUR_DOMAIN_NAME", ParameterType.UrlSegment);
```

(continues on next page)

(continued from previous page)

```
        request.Resource = "domains/{domain}/credentials";
        return client.Execute(request);
    }


}
```

```
import (
    "context"
    "github.com/mailgun/mailgun-go/v3"
    "time"
)

func ListCredentials(domain, apiKey string) ([]mailgun.Credential, error) {
    mg := mailgun.NewMailgun(domain, apiKey)
    it := mg.ListCredentials(nil)

    ctx, cancel := context.WithTimeout(context.Background(), time.Second*30)
    defer cancel()

    var page, result []mailgun.Credential
    for it.Next(ctx, &page) {
        result = append(result, page...)
    }

    if it.Err() != nil {
        return nil, it.Err()
    }
    return result, nil
}
```

```
var DOMAIN = 'YOUR_DOMAIN_NAME';
var mailgun = require('mailgun-js')({ apiKey: "YOUR_API_KEY", domain: DOMAIN });

mailgun.get(`/domains/${DOMAIN}/credentials`, function (error, body) {
  console.log(body);
});
```

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'
```

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode createCredentials() throws UnirestException {

        HttpResponse<JsonNode> jsonResponse = Unirest.post("https://api.mailgun.net/
↪v3/domains/" + YOUR_DOMAIN_NAME +"/credentials")
            .basicAuth("api", API_KEY)
            .field("login", "alice@YOUR_DOMAIN_NAME.com")
            .field("password", "supersecretpassword")
            .asJson();

        return jsonResponse.getBody();
    }
}
```

```
# Include the Autoloader (see "Libraries" for install instructions)
require 'vendor/autoload.php';
use Mailgun\Mailgun;

# Instantiate the client.
$mgClient = Mailgun::create('PRIVATE_API_KEY', 'https://API_HOSTNAME');
$domain   = 'YOUR_DOMAIN_NAME';
$smtpUser = 'bob';
$smtpPass = 'new_password';

# Issue the call to the client.
$result = $mgClient->domains()->createCredential($domain, $smtpUser, $smtpPass);
```

```
def create_credentials():
    return requests.post(
        "https://api.mailgun.net/v3/domains/YOUR_DOMAIN_NAME/credentials",
        auth=("api", "YOUR_API_KEY"),
        data={"login": "alice@YOUR_DOMAIN_NAME",
              "password": "secret"})
```

```
def create_credentials
  RestClient.post "https://api:YOUR_API_KEY"\
  "@api.mailgun.net/v3/domains/YOUR_DOMAIN_NAME/credentials",
  :login => "alice@YOUR_DOMAIN_NAME",
  :password => "secret"
end
```

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class CreateCredentialsChunk
{

    public static void Main (string[] args)
    {
        Console.WriteLine (CreateCredentials ().Content.ToString ());
    }

    public static IRestResponse CreateCredentials ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.AddParameter ("domain", "YOUR_DOMAIN_NAME", ParameterType.UrlSegment);
        request.Resource = "domains/{domain}/credentials";
        request.AddParameter ("login", "alice@YOUR_DOMAIN_NAME");
        request.AddParameter ("password", "secret");
        request.Method = Method.POST;
        return client.Execute (request);
    }

}
```

```
import (
    "context"
    "github.com/mailgun/mailgun-go/v3"
    "time"
)

func CreateCredential(domain, apiKey string) error {
    mg := mailgun.NewMailgun(domain, apiKey)

    ctx, cancel := context.WithTimeout(context.Background(), time.Second*30)
    defer cancel()

    return mg.CreateCredential(ctx, "alice@example.com", "secret")
}
```

```
var DOMAIN = 'YOUR_DOMAIN_NAME';
var mailgun = require('mailgun-js')({ apiKey: "YOUR_API_KEY", domain: DOMAIN });

mailgun.post(`/domains/${DOMAIN}/credentials`, {"login": "alice@YOUR_DOMAIN_NAME",
↪"password": "secret"}, function (error, body) {
  console.log(body);
});
```

```
{
  "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/domains/YOUR_DOMAIN_NAME/credentials/alice \
   -F password='abc123'
```

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode updatePassword() throws UnirestException {

        HttpResponse<JsonNode> jsonResponse = Unirest.put("https://api.mailgun.net/v3/
↪domains/YOUR_DOMAIN_NAME/credentials/alice")
            .basicAuth("api", API_KEY)
            .field("password", "supersecret")
            .asJson();

        return jsonResponse.getBody();
    }
}
```

```
# Include the Autoloader (see "Libraries" for install instructions)
require 'vendor/autoload.php';
use Mailgun\Mailgun;

# Instantiate the client.
$mgClient = Mailgun::create('PRIVATE_API_KEY', 'https://API_HOSTNAME');
$domain   = 'YOUR_DOMAIN_NAME';
$smtpUser = 'bob';
$smtpPass = 'new_password';

# Issue the call to the client.
$result = $mgClient->domains()->updateCredential($domain, $smtpUser, $smtpPass);
```

```
def change_credential_password():
    return requests.put(
        "https://api.mailgun.net/v3/domains/YOUR_DOMAIN_NAME/credentials/alice",
        auth=("api", "YOUR_API_KEY"),
        data={"password": "supersecret"})
```

```
def change_credential_password
  RestClient.put "https://api:YOUR_API_KEY"\
  "@api.mailgun.net/v3/domains/YOUR_DOMAIN_NAME/credentials/alice",
  :password => "supersecret"
end
```

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;
```

(continues on next page)

(continued from previous page)

```
public class ChangePwdCredentialsChunk
{

    public static void Main (string[] args)
    {
        Console.WriteLine (ChangeCredentialPassword ().Content.ToString ());
    }

    public static IRestResponse ChangeCredentialPassword ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.AddParameter ("domain", "YOUR_DOMAIN_NAME", ParameterType.UrlSegment);
        request.Resource = "domains/{domain}/credentials/{username}";
        request.AddUrlSegment ("username", "alice");
        request.AddParameter ("password", "supersecret");
        request.Method = Method.PUT;
        return client.Execute (request);
    }

}
```

```
import (
    "context"
    "github.com/mailgun/mailgun-go/v3"
    "time"
)

func ChangePassword(domain, apiKey string) error {
    mg := mailgun.NewMailgun(domain, apiKey)

    ctx, cancel := context.WithTimeout(context.Background(), time.Second*30)
    defer cancel()

    return mg.ChangeCredentialPassword(ctx, "alice", "super_secret")
}
```

```
var DOMAIN = 'YOUR_DOMAIN_NAME';
var mailgun = require('mailgun-js')({ apiKey: "YOUR_API_KEY", domain: DOMAIN });

mailgun.put(`/domains/${DOMAIN}/credentials/alice`, {"password" : "supersecret"},␣
↪function (error, body) {
  console.log(body);
});
```

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
```

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode deleteCredentials() throws UnirestException {

        HttpResponse<JsonNode> request = Unirest.delete("https://api.mailgun.net/v3/
↪domains/"+ YOUR_DOMAIN_NAME +"/credentials/user")
            .basicAuth("api", API_KEY)
            .asJson();

        return request.getBody();
    }
}
```

```
# Include the Autoloader (see "Libraries" for install instructions)
require 'vendor/autoload.php';
use Mailgun\Mailgun;

# Instantiate the client.
$mgClient = Mailgun::create('PRIVATE_API_KEY', 'https://API_HOSTNAME');
$domain   = 'YOUR_DOMAIN_NAME';
$smtpUser = 'bob';

# Issue the call to the client.
$result = $mgClient->domains()->deleteCredential($domain, $smtpUser);
```

```
def delete_credentials():
    return requests.delete(
        "https://api.mailgun.net/v3/domains/YOUR_DOMAIN_NAME/credentials/alice",
        auth=("api", "YOUR_API_KEY"))
```

```
def delete_credentials
  RestClient.delete "https://api:YOUR_API_KEY"\
  "@api.mailgun.net/v3/domains/YOUR_DOMAIN_NAME/credentials/alice"
end
```

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class DeleteCredentialsChunk
{

    public static void Main (string[] args)
    {
```

(continues on next page)

(continued from previous page)

```
        Console.WriteLine (DeleteCredentials ().Content.ToString ());
    }

    public static IRestResponse DeleteCredentials ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.AddParameter ("domain", "YOUR_DOMAIN_NAME", ParameterType.UrlSegment);
        request.Resource = "domains/{domain}/credentials/{login}";
        request.AddUrlSegment ("login", "alice");
        request.Method = Method.DELETE;
        return client.Execute (request);
    }

}
```

```
import (
    "context"
    "github.com/mailgun/mailgun-go/v3"
    "time"
)

func DeleteCredential(domain, apiKey string) error {
    mg := mailgun.NewMailgun(domain, apiKey)

    ctx, cancel := context.WithTimeout(context.Background(), time.Second*30)
    defer cancel()

    return mg.DeleteCredential(ctx, "alice")
}
```

```
var DOMAIN = 'YOUR_DOMAIN_NAME';
var mailgun = require('mailgun-js')({ apiKey: "YOUR_API_KEY", domain: DOMAIN });

mailgun.put(`/domains/${DOMAIN}/credentials/alice`, function (error, body) {
  console.log(body);
});
```

Sample response:

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

2.5.3 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'.

2.6 Email Validation V3

Note: The V3 Validations API has been deprecated in favor of V4

Mailgun's email validation service is intended for validating email addresses submitted through forms like newsletters, online registrations, and shopping carts.

Maintaining a list of valid and deliverable email addresses is important in order to reduce the ratio of bounce back emails and prevent negative impacts to your reputation as a sender.

Validate a single email address.

```
curl --user 'api:PRIVATE_API_KEY' -G \
    https://api.mailgun.net/v4/address/validate \
    --data-urlencode address='foo@mailgun.net'
```

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode validateEmail() throws UnirestException {

        HttpResponse<JsonNode> request = Unirest.get("https://api.mailgun.net/v4/
↪address/validate")
            .basicAuth("api", PRIVATE_API_KEY)
            .queryString("address", "foo@mailgun.com")
            .asJson();

        return request.getBody();
```

(continues on next page)

(continued from previous page)

```
    }
}
```

```
# Currently, the PHP SDK does not support the v4 Validations endpoint.
# Consider using the following php curl function.
function get_validate() {
  $params = array(
      "address" => "bob@example.com"
  );
  $ch = curl_init();

  curl_setopt($ch, CURLOPT_HTTPAUTH, CURLAUTH_BASIC);
  curl_setopt($ch, CURLOPT_USERPWD, 'api:PRIVATE_API_KEY');
  curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);

  curl_setopt($ch, CURLOPT_CUSTOMREQUEST, 'GET');
  curl_setopt($ch, CURLOPT_POSTFIELDS, $params);
  curl_setopt($ch, CURLOPT_URL, 'https://api.mailgun.net/v4/address/validate');
  $result = curl_exec($ch);
  curl_close($ch);

  return $result;
}
```

```
def get_validate():
    return requests.get(
        "https://api.mailgun.net/v4/address/validate",
        auth=("api", "PRIVATE_API_KEY"),
        params={"address": "foo@mailgun.net"})
```

```
def get_validate
  RestClient.get "https://api:PRIVATE_API_KEY"\
  "@api.mailgun.net/v4/address/validate",
  {params: {address: "foo@mailgun.net"}}
end
```

```
import (
   "encoding/json"
   "net/http"
)

type ValidationResponse struct {
   Address       string   `json:"address"`
   IsDisposable  bool     `json:"is_disposable_address"`
   IsRoleAddress bool     `json:"is_role_address"`
   Reason        []string `json:"reason"`
   Result        string   `json:"result"`
   Risk          string   `json:"risk"`
}


func validateAddress(email string) (vr ValidationResponse, err error) {

   // creating HTTP request and returning response
```

(continues on next page)

(continued from previous page)

```
    client := &http.Client{}
    req, _ := http.NewRequest("GET", "https://api.mailgun.net/v4/address/validate",␣
↪nil)
    req.SetBasicAuth("api", apiKey)
    param := req.URL.Query()
    param.Add("address", email)
    req.URL.RawQuery = param.Encode()
    response, err := client.Do(req)

    if err != nil {
        return
    }

    // decoding into validation response struct
    err = json.NewDecoder(response.Body).Decode(&vr)
    return
}
```

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class GetValidateChunk
{

    public static void Main (string[] args)
    {
        Console.WriteLine (GetValidate ().Content.ToString ());
    }

    public static IRestResponse GetValidate ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v4");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "PRIVATE_API_KEY");
        RestRequest request = new RestRequest ();
        request.Resource = "/address/validate";
        request.AddParameter ("address", "foo@mailgun.net");
        return client.Execute (request);
    }

}
```

Sample response:

```
{
    "address": "foo@mailgun.net",
    "did_you_mean": null,
    "is_disposable_address": false,
    "is_role_address": false,
    "is_valid": true,
    "parts": {
        "display_name": null,
```

(continues on next page)

(continued from previous page)

```
        "domain": "mailgun.net",
        "local_part": "foo"
    }
}
```

Field Explanation:

Parameter	Type	Description
address	string	Email address being validated
did_you_mean	string	Null if nothing, however if a potential typo is made, the closest suggestion is provided
is_disposable_address	boolean	If the domain is in a list of disposable email addresses, this will be appropriately categorized
is_role_address	boolean	Checks the mailbox portion of the email if it matches a specific role type ('admin', 'sales', 'webmaster')
is_valid	boolean	Runs the email segments across a valid known provider rule list. If a violation occurs this value is false
parts	string	(display_name, domain, local_part): Parsed segments of the provided email address

Parse a list of email addresses:

```
curl -G --user 'api:pubkey-501jygdalut926-6mb1ozo8ay9crlc28' \
    https://api.mailgun.net/v3/address/parse \
    --data-urlencode addressses='Alice <alice@example.com>,bob@example.com'
```

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode parseAddresses() throws UnirestException {

        HttpResponse <JsonNode> request = Unirest.get("https://api.mailgun.net/v3/
↪address/parse")
            .basicAuth("api", API_KEY)
            .queryString("addresses", "bob@example.com, alice@example.com")
            .asJson();

        return request.getBody();
    }
}
```

```
# Include the Autoloader (see "Libraries" for install instructions)
require 'vendor/autoload.php';
use Mailgun\Mailgun;

# Instantiate the client.
$mgClient = Mailgun::create('PRIVATE_API_KEY');
$addressList = 'Alice <alice@example.com>,bob@example.com';
```

(continues on next page)

(continued from previous page)

```
# Issue the call to the client.
$result = $mgClient->emailValidation->parse($addressList);
```

```
def get_parse():
    return requests.get(
        "https://api.mailgun.net/v3/address/parse",
        auth=("api", "pubkey-5ogiflzbnjrljiky49qxsiozqef5jxp7"),
        params={"addresses": "Alice <alice@example.com>,bob@example.com"})
```

```
def get_parse
  url_params = { addresses:  "Alice <alice@example.com>,bob@example.com" }
  public_key = "pubkey-5ogiflzbnjrljiky49qxsiozqef5jxp7"
  parse_url = "https://api:#{public_key}@api.mailgun.net/v3/address/parse"
  RestClient::Request.execute method: :get, url: parse_url,
                                  headers: { params: url_params },
                                  user: 'api', password: public_key
end
```

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class GetParseChunk
{

    public static void Main (string[] args)
    {
        Console.WriteLine (GetParse ().Content.ToString ());
    }

    public static IRestResponse GetParse ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "pubkey-5ogiflzbnjrljiky49qxsiozqef5jxp7");
        RestRequest request = new RestRequest ();
        request.Resource = "/address/parse";
        request.AddParameter ("addresses",
                              "Alice <alice@example.com>,bob@example.com");
        return client.Execute (request);
    }

}
```

```
import (
    "context"
    "github.com/mailgun/mailgun-go/v3"
    "time"
)

func ParseAddress(apiKey string) ([]string, []string, error) {
    mv := mailgun.NewEmailValidator(apiKey)
```

(continues on next page)

(continued from previous page)

```
    ctx, cancel := context.WithTimeout(context.Background(), time.Second*30)
    defer cancel()

    return mv.ParseAddresses(ctx,
        "Alice <alice@example.com>",
        "bob@example.com",
        // ...
    )
}
```

```
var DOMAIN = 'YOUR_DOMAIN_NAME';
var mailgun = require('mailgun-js')({ apiKey: "PUBLIC_API_KEY", domain: DOMAIN });

mailgun.parse([ 'alice@example.com', 'bob@example.com', 'fake@email.com' ], function␣
↪(error, body) {
  console.log(body);
});
```

Sample response:

```
{
    "parsed": [
        "Alice <alice@example.com>",
        "bob@example.com"
    ],
    "unparseable": [
    ]
}
```

2.6.1 Mailbox Verification

Mailgun has the ability, for supported mailbox providers, to check and determine if a mailbox exists on the target domain. This check is an additional safeguard against typos. The mailbox verification check will return true, false, or unknown. Unknown may be returned if the mailbox provider does not support the check or the check times out.

2.6.2 Role-based Address Check

For all validation requests, we provide whether an address is a role-based address (e.g. postmaster@, info@, etc.). These addresses are typically distribution lists with a much higher complaint rate since unsuspecting users on the list can receive a message they were not expecting.

2.6.3 Disposable Mailbox Detection

Disposable mailboxes are commonly used for fraudulent purposes. Mailgun can detect whether the address provided is on a known disposable mailbox provider and given the determination, you may choose how to proceed based on your own risk decisions. It is important to check for disposable mailboxes to ensure communication between user and web application.

Sample response:

```
{
    "address": "fake@throwawaymail.com",
    "did_you_mean": null,
    "is_disposable_address": true,
    "is_role_address": false,
    "is_valid": true,
    "mailbox_verification": true,
    "parts": {
        "display_name": null,
        "domain": "throwawaymail.com",
        "local_part": "fake"
    }
}
```


2.6.4 Reporting Dashboard

Within the validation menu, you can view your usage by day or hour for the validation API in a given time range. Mailgun will also include the type of API call that was made to help measure the impact of email address validation.

2.7 Email Validation V4

Mailgun's email validation service is a multi-point check of an email address to ensure it exists, is not a high-risk address, is not disposable and more. Maintaining a list of valid and deliverable email addresses is important in order to reduce the ratio of bounced emails and prevent negative impacts to your reputation as a sender.

Mailgun offers validations in three forms: performing a **single validation**, validating the email addresses of the members of a **mailing list**, and validating lists in **bulk** via CSV.

Mailgun's revamp of our initial validation service now offers a definitive **result** of our validation check and a **risk** assessment of the given address. The **result** parameter will return one of four values, described in the table below. This is the definitive answer for whether or not a sender should use an address and replaces the **is_valid** parameter in v3 validations.

deliverable	An address that has a high likelihood of being legitimate and has passed all validation checks.
undeliverable	An address that is confirmed invalid and should be discarded from a list or corrected. Sending to this address may damage sender reputation.
do_not_send	An address that may damage sender reputation if messages or sent.
unknown	Unable to make a decision about the validity of the supplied address, usually due to a timeout when attempting to verify an address.

The **risk** parameter will return one of three values, described in the table below. This helps senders understand how sending to an address may impact reputation or the potential impact of allowing a user onto a platform.

low	An address that is likely legitimate and sending has a low likelihood of damaging reputation if the address has been obtained in a legitimate manner - we'll make this assumption based on well known domains (hotmail.com, gmail.com, etc)
medium	The default or neutral state for risk calculation. An address that isn't deemed a low or high risk will default to a medium risk.
high	An address that has a high risk of damaging sender reputation or when used for verification should be challenged for validity.

2.7.1 Role-based Address Check

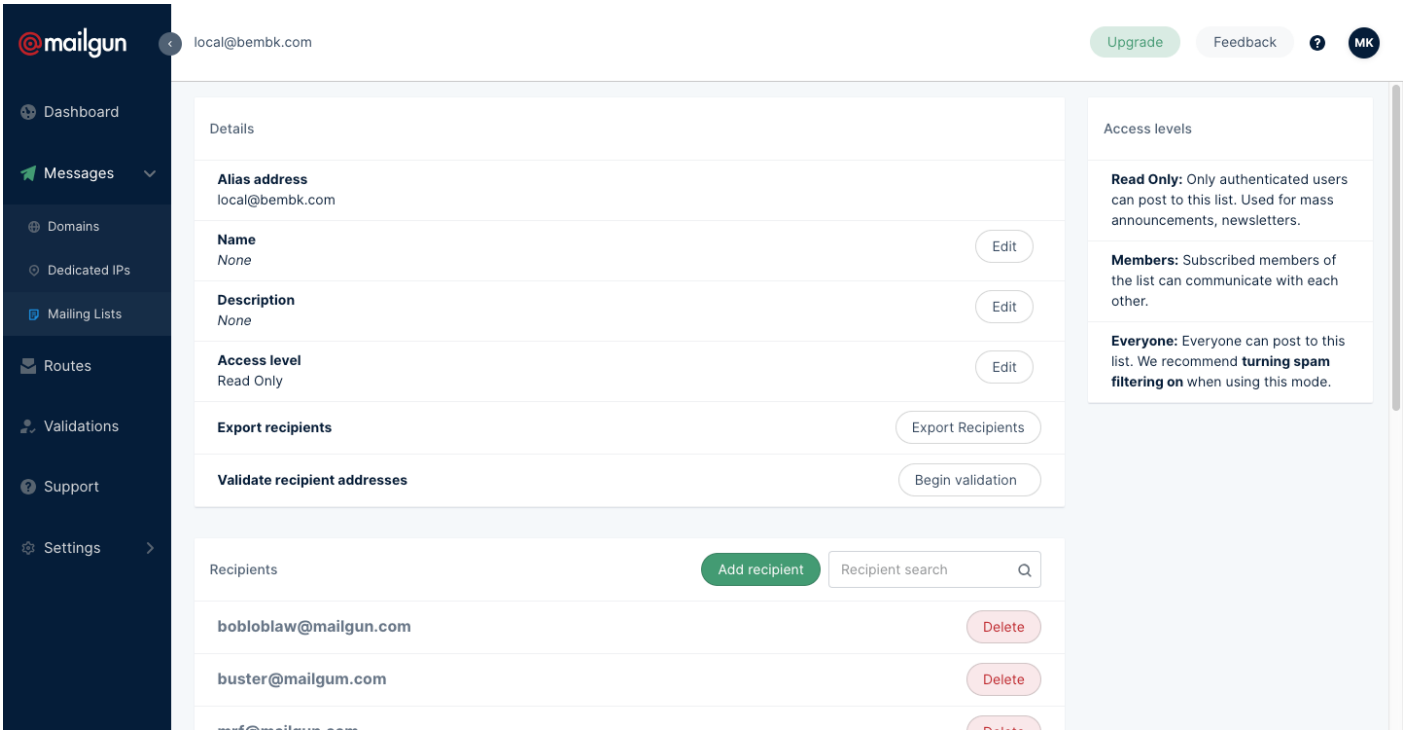
For all validation requests, we provide whether an address is a role-based address (e.g. postmaster@, info@, etc.). These addresses are typically distribution lists with a much higher complaint rate since unsuspecting users on the list can receive a message they were not expecting.

2.7.2 Disposable Mailbox Detection

Disposable mailboxes are commonly used for fraudulent purposes. Mailgun can detect whether the address provided is on a known disposable mailbox provider and given the determination, you may choose how to proceed based on your own risk decisions. It is important to check for disposable mailboxes to ensure communication between user and web application.

2.7.3 List Validation

The members of a Mailing List that exists at Mailgun can be validating with the tap of a button in the Control Panel as demonstrated below:

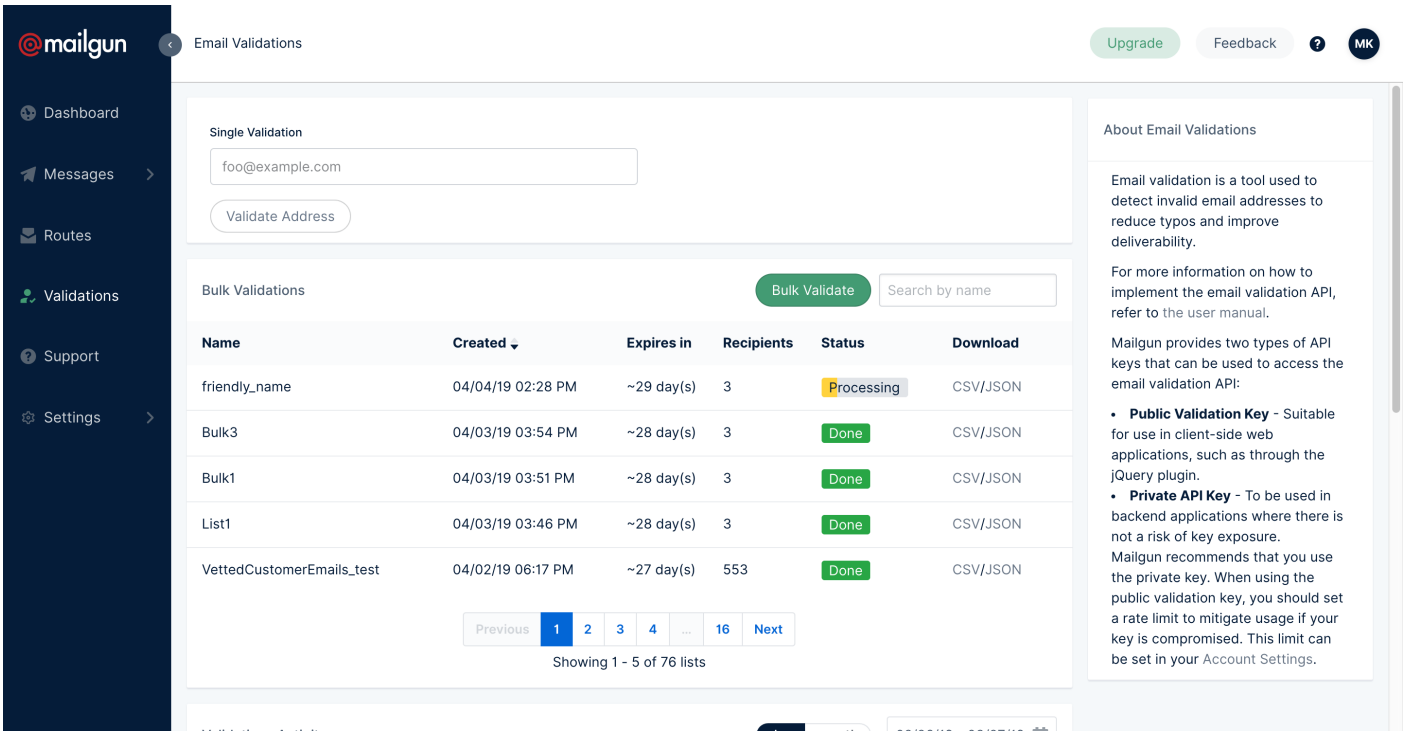


Note that the existing limitation of a maximum of 2.5 million members still exists for Mailing Lists.

2.7.4 Bulk Validation

A CSV no greater than 25 MB can be submitted via API or Control Panel for validation *en masse*. In addition, a gzip no greater than 25 MB containing a single CSV of whatever size can also be submitted which greatly increases the potential size of the list that can take.

Note that the following conditions must be met:



- The column of email addresses must be preceded with a single cell with the text email or email_address in it. Any other columns will be ignored. This text must be all lowercase.
- The size of the CSV must not exceed 25 MB, and the size of the gzip must not exceed 25 MB.
- For best results, do not include a header line in the CSV.
- Make sure the contents of the CSV do not contain any non-ASCII characters. The character set used must be ASCII or UTF-8.

2.8 Inbox Placement

The Inbox Placement product is an email deliverability tool that provides visibility into where an email will land in the mailbox. While mailbox providers (i.e. Gmail, Yahoo, Hotmail, etc.) will provide feedback that an email is delivered or not (i.e. delivery), they do not provide insight into where in the mailbox the email landed (i.e. deliverability). Specifically, an email can land in the inbox, spam/junk folder, or in the case of Gmail, specific tabs within the inbox. The Inbox Placement product utilizes a mechanism known as seed testing to provide visibility in to where emails are landing.

Seed testing roughly works as follows:

1. Mailgun manages a list of seed mailbox accounts with mailbox providers in the market.
2. A test email is sent to the seed list.
3. Mailgun tracks where the test email landed for each mailbox in the seed list and returns the results to the user. The results contain the following information:
    a. “Spam”, “Missed”, or “Inbox” placement (and which tab for Gmail mailboxes) for each individual mailbox in our seed list.

b. A rollup percentage summary for each mailbox provider. The rollup aggregates the results from each mailbox.

c. A rollup percentage summary for each seed test. The rollup aggregates the results from each mailbox provider.

2.8.1 Creating a Test

Creating and running an Inbox Placement test is simple and only requires minimal configuration in either the control panel or the API:

Required

- A sending domain to send a test from. If you haven't set up a domain please see the *Verifying Your Domain* section.
- A message subject.
- A message body.

Optional

- Provide a 'from' address prefix. Mailgun provides 'user' by default.

Once configured, a test can be performed. Please wait for results to come in.

2.9 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

2.9.1 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_require_auth = <; $host_address
        hosts_require_tls = <; $host_address
```

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
```

2.9.2 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.

2.10 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.

2.11 Internationalization

Internationalized Domain Names (IDN)

Our messages API supports sending to addresses that leverage internationalized domain names in the *to* and *from* fields. When necessary, Mailgun will automatically convert the domains to the ASCII equivalent through the use of punycode

At this time, sending domains cannot be created using non-ASCII characters.

Internationalized Email Addresses (SMTPUTF8)

Mailgun supports internationalized email addresses through the use of the SMTPUTF8 extension. An internationalized email address will contain a non-ASCII character in the local-part portion of the email address and may also use an internationalized domain name.

Mailgun supports internationalized email addresses in the following portions of our product:

- Outgoing Messages (HTTP API / SMTP endpoint)
- Inbound SMTP
- Routes (match_recipient and forward action)
- Mailing Lists (list names and member addresses)
- Email Validation
- Suppressions Lists

In order to send messages to an internationalized email address, the receiving mailbox provider must support the SMTPUTF8 extension.

CHAPTER 3

Libraries

The Mailgun API is built on HTTP. Our API is **RESTful_** and it:

- Uses predictable, resource-oriented URLs.
- Uses built-in HTTP capabilities for passing parameters and authentication.
- Responds with standard HTTP response codes to indicate errors.
- Returns **JSON_**.

Mailgun has published libraries for various languages. You may use our libraries, or your favorite/suggested HTTP/REST library available for your programming language, to make HTTP calls to Mailgun.

Most code samples in our docs can be viewed in several programming languages by using the language bar at the top.

Mailgun maintains the following official libraries for interfacing with our platform:

3.1 Official Mailgun Libraries

- mailgun-php
- mailgun-ruby
- mailgun-go
- mailgun-js

CHAPTER 4

API Reference

4.1 Introduction

The Mailgun API is built on HTTP. Our API is RESTful and it:

- Uses predictable, resource-oriented URLs.
- Uses built-in HTTP capabilities for passing parameters and authentication.
- Responds with standard HTTP response codes to indicate errors.
- Returns JSON.

Mailgun has published *Libraries* for various languages. You may use our libraries, or your favorite HTTP/REST library available for your programming language, to make HTTP calls to Mailgun. Visit our *Libraries* page to see HTTP REST libraries we recommend.

To give you an idea of how to use the API, we have annotated our documentation with code samples written in several popular programming languages. Use the language selector at the top to switch between them.

Our samples from *Quickstart Guide*, *User Manual*, and *API Reference* provide examples that will function. You're welcome to copy/paste and run the script to see the API in action.

4.1.1 Base URL

All API calls referenced in our documentation start with a base URL. Mailgun allows the ability to send and receive email in either our US region or our EU region. Be sure to use the appropriate base URL based on which region you've created your domain in.

It's also important to note that Mailgun uses URI versioning for our API endpoints, and some endpoints may have different versions than others. Please reference the version stated in the URL for each endpoint.

For domains created in our US region the base URL is:

```
https://api.mailgun.net/
```

For domains created in our EU region the base URL is:

```
https://api.eu.mailgun.net/
```

Your Mailgun account may contain multiple sending domains. To avoid passing the domain name as a query parameter, most API URLs must include the name of the domain you're interested in:

```
https://api.mailgun.net/v3/mydomain.com
```

4.1.2 Authentication

Authentication to the Mailgun API is done by providing an Authorization header using HTTP Basic Auth; use `api` as the username and your API key as the password . Mailgun provides two types of API keys for authenticating against the API:

Primary account API key

When you sign up for Mailgun, we generate a primary account API key. This key allows you to perform all CRUD operations via our various API endpoints and for any of your sending domains. To view your primary account API key, in the Mailgun dashboard click on **Settings** on the left-hand nav in the Mailgun dashboard and then **API Keys** and click on the eye icon next to **Private API key**.

Domain Sending Keys

Domain Sending Keys are API keys that **only** allow sending messages via a POST call on our /messages and /messages.mime endpoints for the domain in which they are created for. In order to create a sending API key:

- Click on the **Sending** drawer on the left-hand side of the Mailgun dashboard
- Click on **Domains**, select the domain in which you wish to add a sending key to
- Click on **Domain Settings** and navigate to the **Sending API keys** tab
- Click on **Add Sending Key**
- Give your key a suitable description (such as the name of the application or client you're creating the key for) and click **Create Sending Key**
- Copy your sending API key and keep it in a safe place. For security purposes, we will not be able to show you the key again. If you lose your key, you will need to create a new key.

Here is how you use basic HTTP auth with curl:

```
curl --user 'api:YOUR_API_KEY'
```

> **Warning:** Keep your API key secret!

4.1.3 Date Format

Mailgun returns JSON for all API calls. JSON does not have a built-in date type, dates are passed as strings encoded according to **RFC 2822#page-14**. This format is native to JavaScript and is also supported by most programming languages out of the box:

```
'Thu, 13 Oct 2011 18:02:00 GMT'
```

4.1.4 Status codes

Mailgun returns standard HTTP response codes.

Code	Description
200	Everything worked as expected
400	Bad Request - Often missing a required parameter
401	Unauthorized - No valid API key provided
402	Request Failed - Parameters were valid but request failed
404	Not Found - The requested item doesn't exist
413	Request Entity Too Large - Attachment size is too big
429	Too many requests - An API request limit has been reached
500, 502, 503, 504	Server Errors - something is wrong on Mailgun's end

4.1.5 Webhooks

Mailgun can also POST data to your application when events (opens, clicks, bounces, etc.) occur or when you use Routes. You can read more about *Webhooks* and *Routes* in the *User Manual*.

4.1.6 Mailgun Regions

Using a single account and billing plan, you can choose to provision new sending domains in the EU environment. Message data never leaves the region in which it is processed. Only a limited amount of account data is replicated globally, giving you a single account from which to manage domains in both the US and the EU. Here are the specifics on the type of data that is replicated globally versus what is region-bound.

Global	Region-Bound (US / EU)
Account Information, User Accounts, Billing Details (invoices/plan information), API Keys, Domain Names	Domain Metadata (e.g. SMTP credentials), Messages, Event Logs, Suppressions, Mailing Lists, Tags, Statistics, Routes, IP Addresses

Below are the endpoints you will use for sending/receiving/tracking messages in the EU:

Service	US Endpoint	EU Endpoint
REST API	api.mailgun.net	api.eu.mailgun.net
Outgoing SMTP Server	smtp.mailgun.org	smtp.eu.mailgun.org
Inbound SMTP Server (Routes)	mxa.mailgun.org	mxa.eu.mailgun.org
Inbound SMTP Server (Routes)	mxb.mailgun.org	mxb.eu.mailgun.org
Open/Click Tracking Endpoint	mailgun.org	eu.mailgun.org

4.1.7 Postman Integration

Mailgun has a Postman Collection available for quick and easy exercise of our REST-based APIs. Included in the collection is a Mailgun Environment for easy changing of domains, regions and API keys. Use the button below for easy import into Postman. Don't have Postman? Click here.

Read more about Mailgun and Postman on our blog.

4.2 Messages

4.2.1 Sending

There are two ways to send emails using Mailgun API:

```
v3/<domain>/messages
```

- You can pass the components of the messages such as `To`, `From`, `Subject`, HTML and text parts, attachments, etc. Mailgun will build a MIME representation of the message and send it. This is the preferred method.

```
v3/<domain>/messages.mime
```

- You can also build a MIME string yourself using a MIME library for your programming language and submit it to Mailgun.

Note: You can also use good old SMTP to send messages. But you will have to specify all advanced sending options via *MIME headers*

Note: Sending options (those prefixed by o:, h:, or v:) are limited to 16 kB in total.

```
POST /<domain>/messages
```

Sends a message by assembling it from the components. Note that you can specify most parameters multiple times, HTTP supports this out of the box. This makes sense for parameters like `cc`, `to` or `attachment`.

Parameter	Description
from	Email address for `From` header
to	Email address of the recipient(s). Example: `"Bob <bob@host.com>"`. You can use commas to separate multiple recipients.
cc	Same as `To` but for `Cc`
bcc	Same as `To` but for `Bcc`
subject	Message subject
text	Body of the message. (text version)
html	Body of the message. (HTML version)
amp-html	AMP part of the message. Please follow google guidelines to compose and send AMP emails.
attachment	File attachment. You can post multiple `attachment` values. **Important:** You must use `multipart/form-data` encoding when sending attachments.
inline	Attachment with `inline` disposition. Can be used to send inline images (see *example*). You can post multiple `inline` values.
template	Name of a template stored via *template API*. See *Templates* for more information
t:version	Use this parameter to send a message to specific version of a template
t:text	Pass `yes` if you want to have rendered template in the text part of the message in case of template sending
o:tag	Tag string. See *Tagging* for more information.
o:dkim	Enables/disables DKIM signatures on per-message basis. Pass `yes`, `no`, `true` or `false`
o:deliverytime	Desired time of delivery. See *Date Format*. Note: Messages can be scheduled for a maximum of 3 days in the future.
o:deliverytime-optimize-period	Toggles Send Time Optimization (STO) on a per-message basis. String should be set to the number of hours in `[0-9]+h` format, with the minimum being `24h` and the maximum being `72h`. This value defines the time window in which Mailgun will run the optimization algorithm based on prior engagement data of a given recipient. See *Sending a message with STO* for details. *Please note that STO is only available on certain plans. See www.mailgun.com/pricing for more info.*
o:time-zone-localize	Toggles Timezone Optimization (TZO) on a per message basis. String should be set to preferred delivery time in `HH:mm` or `hh:mmaa` format, where `HH:mm` is used for 24 hour format without AM/PM and `hh:mmaa` is used for 12 hour format with AM/PM. See *Sending a message with TZO* for details. *Please note that TZO is only available on certain plans. See www.mailgun.com/pricing for more info.*
o:testmode	Enables sending in test mode. Pass `yes` if needed. See *Sending in Test Mode*
o:tracking	Toggles tracking on a per-message basis, see *Tracking Messages* for details. Pass `yes`, `no`, `true` or `false`
o:tracking-clicks	Toggles clicks tracking on a per-message basis. Has higher priority than domain-level setting. Pass `yes`, `no`, `true`, `false` or `htmlonly`.
o:tracking-opens	Toggles opens tracking on a per-message basis. Has higher priority than domain-level setting. Pass `yes` or `no`, `true` or `false`
o:require-tls	If set to `True` or `yes` 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` or `no`, 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.
o:skip-verification	If set to `True` or `yes`, 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` or `no`, 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`.
h:X-My-Header	`h:` prefix followed by an arbitrary value allows to append a custom MIME header to the message (`X-My-Header` in this case). For example, `h:Reply-To` to specify Reply-To address.
v:my-var	`v:` prefix followed by an arbitrary name allows to attach a custom JSON data to the message. See *Attaching Data to Messages* for more information.
recipient	A valid JSON-encoded dictionary, where key is a plain recipient address and value is a dictionary with

```
POST /<domain>/messages.mime
```

Posts a message in MIME format. Note: you will need to build a MIME string yourself. Use a MIME library for your programming language to do this. Pass the resulting MIME string as `message` parameter.

Note: You must use `multipart/form-data` encoding.

Parameter	Description
to	Email address of the recipient(s). Example: `"Bob <bob@host.com>"`. You can use commas to separate multiple recipients. Make sure to include all `To`, `Cc` and `Bcc` recipients of the message.
message	MIME string of the message. Make sure to use `multipart/form-data` to send this as a file upload.
o:tag	Tag string. See *Tagging* for more information.
o:deliverytime	Desired time of delivery. See *Date Format*. Note: Messages can be scheduled for a maximum of 3 days in the future.
o:dkim	Enables/disabled DKIM signatures on per-message basis. Pass `yes` or `no`
o:testmode	Enables sending in test mode. Pass `yes` if needed. See *Sending in Test Mode*
o:tracking	Toggles tracking on a per-message basis, see *Tracking Messages* for details. Pass `yes` or `no`.
o:tracking-clicks	Toggles clicks tracking on a per-message basis. Has higher priority than domain-level setting. Pass `yes`, `no` or `htmlonly`.
o:tracking-opens	Toggles opens tracking on a per-message basis. Has higher priority than domain-level setting. Pass `yes` or `no`.
h:X-My-Header	`h:` prefix followed by an arbitrary value allows to append a custom MIME header to the message (`X-My-Header` in this case). For example, `h:Reply-To` to specify Reply-To address.
v:my-var	`v:` prefix followed by an arbitrary name allows to attach a custom JSON data to the message. See *Attaching Data to Messages* for more information.

4.2.2 Retrieving Stored Messages

To retrieve an inbound message that has been stored via the `store()` action, use the URL found in the stored event (which you can find through the Events API, or in the notify webhook set when creating the store action (`store(notify="http://mydomain.com/callback")`).

- By default the message will be returned in JSON form with parsed parts. Links to the attachments will be included.
- You can also retrieve the full raw mime message (attachments and all) if you make the request to the URL with the `Accept` header set to `message/rfc2822`.
- Stored messages are encoded with Quoted-printable encoding. Decoding samples are available in the examples section below.

These are the parameters of the JSON returned from a GET request to a stored message url.

Parameter	Type	Description
recipients	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.
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-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: 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.

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

These are the parameters when the `Accept` header is set to `message/rfc2822`

Parameter	Type	Description
recipients	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>".
subject	string	subject string.
body-mime	string	full MIME envelope. You will need a MIME parsing library to process this data.

4.2.3 Deleting Stored Messages

Stored messages are retained in the system for 3 days and automatically purged after this retention period, therefore there is no need to delete messages explicitly.

Note: Mailgun reserves the right to impose a limit on the size and number of stored messages. In the event this is necessary, you will be notified in advance.

4.2.4 Examples

Warning: Some samples are using curl utility for API examples. UNIX shells require that some characters must be escaped, for example `$` becomes `\$`.

If your API key contains unescaped characters you may receive HTTP error 401 (Unauthorized).

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 awesomeness!'
```

```
import java.io.File;

import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;


public class MGSample {

    // ...

    public static JsonNode sendSimpleMessage() throws UnirestException {
```

(continues on next page)

(continued from previous page)

```
        HttpResponse<JsonNode> request = Unirest.post("https://api.mailgun.net/v3/" +␣
↪YOUR_DOMAIN_NAME + "/messages")
            .basicAuth("api", API_KEY)
            .field("from", "Excited User <USER@YOURDOMAIN.COM>")
            .field("to", "artemis@example.com")
            .field("subject", "hello")
            .field("text", "testing")
            .asJson();

        return request.getBody();
    }
}
```

```
# Include the Autoloader (see "Libraries" for install instructions)
require 'vendor/autoload.php';
use Mailgun\Mailgun;

# Instantiate the client.
$mgClient = Mailgun::create('PRIVATE_API_KEY', 'https://API_HOSTNAME');
$domain = "YOUR_DOMAIN_NAME";
$params = array(
  'from'    => 'Excited User <YOU@YOUR_DOMAIN_NAME>',
  'to'      => 'bob@example.com',
  'subject' => 'Hello',
  'text'    => 'Testing some Mailgun awesomness!'
);

# Make the call to the client.
$mgClient->messages()->send($domain, $params);
```

```
def send_simple_message():
    return requests.post(
        "https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/messages",
        auth=("api", "YOUR_API_KEY"),
        data={"from": "Excited User <mailgun@YOUR_DOMAIN_NAME>",
              "to": ["bar@example.com", "YOU@YOUR_DOMAIN_NAME"],
              "subject": "Hello",
              "text": "Testing some Mailgun awesomness!"})
```

```
def send_simple_message
  RestClient.post "https://api:YOUR_API_KEY"\
  "@api.mailgun.net/v3/YOUR_DOMAIN_NAME/messages",
  :from => "Excited User <mailgun@YOUR_DOMAIN_NAME>",
  :to => "bar@example.com, YOU@YOUR_DOMAIN_NAME",
  :subject => "Hello",
  :text => "Testing some Mailgun awesomness!"
end
```

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class SendSimpleMessageChunk
```

(continues on next page)

(continued from previous page)

```
{

    public static void Main (string[] args)
    {
        Console.WriteLine (SendSimpleMessage ().Content.ToString ());
    }

    public static IRestResponse SendSimpleMessage ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.AddParameter ("domain", "YOUR_DOMAIN_NAME", ParameterType.UrlSegment);
        request.Resource = "{domain}/messages";
        request.AddParameter ("from", "Excited User <mailgun@YOUR_DOMAIN_NAME>");
        request.AddParameter ("to", "bar@example.com");
        request.AddParameter ("to", "YOU@YOUR_DOMAIN_NAME");
        request.AddParameter ("subject", "Hello");
        request.AddParameter ("text", "Testing some Mailgun awesomness!");
        request.Method = Method.POST;
        return client.Execute (request);
    }

}
```

```
import (
    "context"
    "github.com/mailgun/mailgun-go/v3"
    "time"
)

func SendSimpleMessage(domain, apiKey string) (string, error) {
    mg := mailgun.NewMailgun(domain, apiKey)
    m := mg.NewMessage(
        "Excited User <mailgun@YOUR_DOMAIN_NAME>",
        "Hello",
        "Testing some Mailgun awesomeness!",
        "YOU@YOUR_DOMAIN_NAME",
    )

    ctx, cancel := context.WithTimeout(context.Background(), time.Second*30)
    defer cancel()

    _, id, err := mg.Send(ctx, m)
    return id, err
}
```

```
var API_KEY = 'YOUR_API_KEY';
var DOMAIN = 'YOUR_DOMAIN_NAME';
var mailgun = require('mailgun-js')({apiKey: API_KEY, domain: DOMAIN});

const data = {
  from: 'Excited User <me@samples.mailgun.org>',
```

(continues on next page)

(continued from previous page)

```
  to: 'foo@example.com, bar@example.com',
  subject: 'Hello',
  text: 'Testing some Mailgun awesomeness!'
};

mailgun.messages().send(data, (error, body) => {
  console.log(body);
});
```

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
```

```
import java.io.File;

import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode sendComplexMessage() throws UnirestException {

        HttpResponse<JsonNode> request = Unirest.post("https://api.mailgun.net/v3/" +␣
↪YOUR_DOMAIN_NAME + "/messages")
            .basicAuth("api", API_KEY)
            .field("from", "Excited User <USER@YOURDOMAIN.COM>")
            .field("to", "alice@example.com")
            .field("cc", "bob@example.com")
            .field("bcc", "joe@example.com")
            .field("subject", "Hello")
            .field("text", "Testing out some Mailgun awesomeness!")
            .field("html", "<html>HTML version </html>")
            .field("attachment", new File("/temp/folder/test.txt"))
            .asJson();
```

(continues on next page)

(continued from previous page)

```
        return request.getBody();
    }
}
```

```
# Include the Autoloader (see "Libraries" for install instructions)
require 'vendor/autoload.php';
use Mailgun\Mailgun;

# Instantiate the client.
$mgClient = Mailgun::create('PRIVATE_API_KEY', 'https://API_HOSTNAME');
$domain = "YOUR_DOMAIN_NAME";
$params = array(
      'from'    => 'Excited User <YOU@YOUR_DOMAIN_NAME>',
      'to'      => 'bob@example.com',
      'cc'      => 'alice@example.com',
      'bcc'     => 'john@example.com',
      'subject' => 'Hello',
      'text'    => 'Testing some Mailgun awesomness!',
      'html'    => '<html>HTML version of the body</html>',
      'attachment' => array(
             array(
                 'filePath' => 'test.txt',
                 'filename' => 'test_file.txt'
             )
      )
    );

# Make the call to the client.
$result = $mgClient->messages()->send($domain, $params);
```

```
def send_complex_message():
    return requests.post(
        "https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/messages",
        auth=("api", "YOUR_API_KEY"),
        files=[("attachment", ("test.jpg", open("files/test.jpg","rb").read())),
               ("attachment", ("test.txt", open("files/test.txt","rb").read()))],
        data={"from": "Excited User <YOU@YOUR_DOMAIN_NAME>",
              "to": "foo@example.com",
              "cc": "baz@example.com",
              "bcc": "bar@example.com",
              "subject": "Hello",
              "text": "Testing some Mailgun awesomness!",
              "html": "<html>HTML version of the body</html>"})
```

```
def send_complex_message
  data = {}
  data[:from] = "Excited User <YOU@YOUR_DOMAIN_NAME>"
  data[:to] = "foo@example.com"
  data[:cc] = "baz@example.com"
  data[:bcc] = "bar@example.com"
  data[:subject] = "Hello"
  data[:text] = "Testing some Mailgun awesomness!"
  data[:html] = "<html>HTML version of the body</html>"
  data[:attachment] = []
  data[:attachment] << File.new(File.join("files", "test.jpg"))
```

(continues on next page)

(continued from previous page)

```
  data[:attachment] << File.new(File.join("files", "test.txt"))
  RestClient.post "https://api:YOUR_API_KEY"\
  "@api.mailgun.net/v3/YOUR_DOMAIN_NAME/messages", data
end
```

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class SendComplexMessageChunk
{

    public static void Main (string[] args)
    {
        Console.WriteLine (SendComplexMessage ().Content.ToString ());
    }

    public static IRestResponse SendComplexMessage ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.AddParameter ("domain", "YOUR_DOMAIN_NAME", ParameterType.UrlSegment);
        request.Resource = "{domain}/messages";
        request.AddParameter ("from", "Excited User <YOU@YOUR_DOMAIN_NAME>");
        request.AddParameter ("to", "foo@example.com");
        request.AddParameter ("cc", "baz@example.com");
        request.AddParameter ("bcc", "bar@example.com");
        request.AddParameter ("subject", "Hello");
        request.AddParameter ("text", "Testing some Mailgun awesomness!");
        request.AddParameter ("html",
                              "<html>HTML version of the body</html>");
        request.AddFile ("attachment", Path.Combine ("files", "test.jpg"));
        request.AddFile ("attachment", Path.Combine ("files", "test.txt"));
        request.Method = Method.POST;
        return client.Execute (request);
    }

}
```

```
import (
    "context"
    "github.com/mailgun/mailgun-go/v3"
    "time"
)

func SendComplexMessage(domain, apiKey string) (string, error) {
    mg := mailgun.NewMailgun(domain, apiKey)
    m := mg.NewMessage(
        "Excited User <YOU@YOUR_DOMAIN_NAME>",
        "Hello",
        "Testing some Mailgun awesomeness!",
```

(continues on next page)

(continued from previous page)

```
        "foo@example.com",
    )
    m.AddCC("baz@example.com")
    m.AddBCC("bar@example.com")
    m.SetHtml("<html>HTML version of the body</html>")
    m.AddAttachment("files/test.jpg")
    m.AddAttachment("files/test.txt")

    ctx, cancel := context.WithTimeout(context.Background(), time.Second*30)
    defer cancel()

    _, id, err := mg.Send(ctx, m)
    return id, err
}
```

```
const path = require('path');
var DOMAIN = 'YOUR_DOMAIN_NAME';
var api_key = 'YOUR_API_KEY';
var mailgun = require('mailgun-js')({ apiKey: api_key, domain: DOMAIN });

var filepath = path.join(__dirname, 'sample.jpg');

var data = {
  from: 'Excited User <me@samples.mailgun.org>',
  to: 'foo@example.com, baz@example.com, bar@example.com',
  cc: 'baz@example.com',
  bcc: 'bar@example.com',
  subject: 'Complex',
  text: 'Testing some Mailgun awesomness!',
  html: "<html>HTML version of the body</html>",
  attachment: filepath
};

mailgun.messages().send(data, function (error, body) {
  console.log(body);
});
```

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
```

```
import java.io.File;

import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode sendMIMEMessage() throws UnirestException {
```

(continues on next page)

(continued from previous page)

```
        HttpResponse<JsonNode> request = Unirest.post("https://api.mailgun.net/v3/" +␣
↪YOUR_DOMAIN_NAME + "/messages.mime")
            .basicAuth("api", API_KEY)
            .header("content-type", "multipart/form-data;")
            .field("from", "Excited User <USER@YOURDOMAIN.COM>")
            .field("to", "Megan@example.com")
            .field("subject", "Bah-weep-graaaaagnah wheep nini bong.")
            .field("message", new File("/temp/folder/file.mime"))
            .asJson();

        return request.getBody();
    }
}
```

```
# Include the Autoloader (see "Libraries" for install instructions)
require 'vendor/autoload.php';
use Mailgun\Mailgun;

# Instantiate the client.
$mgClient = Mailgun::create('PRIVATE_API_KEY', 'https://API_HOSTNAME');
$domain = "YOUR_DOMAIN_NAME";

$recipients = array(
    'bob@example.com',
    'alice@example.com',
    'john@example.com;
);
$params = array(
    'from' => 'Excited User <YOU@YOUR_DOMAIN_NAME>'
);

$mime_string = '<Pass fully formed MIME string here>'

# Make the call to the client.
$result = $mgClient->messages()->sendMime($domain, $recipients, $mime_string,
↪$params);
```

```
def send_mime_message():
    return requests.post(
        "https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/messages.mime",
        auth=("api", "YOUR_API_KEY"),
        data={"to": "bar@example.com"},
        files={"message": open("files/message.mime")})
```

```
def send_mime_message
  RestClient.post "https://api:YOUR_API_KEY"\
  "@api.mailgun.net/v3/YOUR_DOMAIN_NAME/messages.mime",
  :to => "bar@example.com",
  :message => File.new(File.join("files", "message.mime"))
end
```

```
using System;
using System.IO;
using RestSharp;
```

(continues on next page)

(continued from previous page)

```
using RestSharp.Authenticators;

public class SendMimeMessageChunk
{

    public static void Main (string[] args)
    {
        Console.WriteLine (SendMimeMessage ().Content.ToString ());
    }

    public static IRestResponse SendMimeMessage ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.AddParameter ("domain", "YOUR_DOMAIN_NAME", ParameterType.UrlSegment);
        request.Resource = "{domain}/messages.mime";
        request.AddParameter ("to", "bar@example.com");
        request.AddFile ("message", Path.Combine ("files", "message.mime"));
        request.Method = Method.POST;
        return client.Execute (request);
    }

}
```

```
import (
    "context"
    "github.com/mailgun/mailgun-go/v3"
    "os"
    "time"
)

func SendMimeMessage(domain, apiKey string) (string, error) {
    mg := mailgun.NewMailgun(domain, apiKey)
    mimeMsgReader, err := os.Open("files/message.mime")
    if err != nil {
        return "", err
    }

    m := mg.NewMIMEMessage(mimeMsgReader, "bar@example.com")

    ctx, cancel := context.WithTimeout(context.Background(), time.Second*30)
    defer cancel()

    _, id, err := mg.Send(ctx, m)
    return id, err
}
```

```
var DOMAIN = 'YOUR_DOMAIN_NAME';
var mailgun = require('mailgun-js')({ apiKey: "YOUR_API_KEY", domain: DOMAIN });
var MailComposer = require('nodemailer/lib/mail-composer');

var mailOptions = {
```

(continues on next page)

(continued from previous page)

```
  from: 'YOU@YOUR_DOMAIN_NAME',
  to: 'bob@example.com',
  subject: 'Hello',
  text: 'Testing some Mailgun awesomeness!'
};

var mail = new MailComposer(mailOptions);

mail.compile().build(function(mailBuildError, message) {

    var dataToSend = {
        to: 'bob@example.com',
        message: message.toString('ascii')
    };

    mailgun.messages().sendMime(dataToSend, function (sendError, body) {
        if (sendError) {
            console.log(sendError);
            return;
        }
    });
});
```

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
```

```
import java.io.File;

import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode sendMessageNoTracking() throws UnirestException {

        HttpResponse<JsonNode> request = Unirest.post("https://api.mailgun.net/v3/" +␣
↪YOUR_DOMAIN_NAME + "/messages")
            .basicAuth("api", API_KEY)
            .field("from", "Excited User <YOU@YOUR_DOMAIN_NAME>")
            .field("to", "alice@example.com")
            .field("subject", "Hello")
            .field("text", "Testing some Mailgun awesomeness")
            .field("o:tracking", "False")
            .asJson();
```

(continues on next page)

(continued from previous page)

```
        return request.getBody();
    }
}
```

```
# Include the Autoloader (see "Libraries" for install instructions)
require 'vendor/autoload.php';
use Mailgun\Mailgun;

# Instantiate the client.
$mgClient = Mailgun::create('PRIVATE_API_KEY', 'https://API_HOSTNAME');
$domain = "YOUR_DOMAIN_NAME";

$params = array(
    'from'       => 'Excited User <YOU@YOUR_DOMAIN_NAME>',
    'to'         => 'foo@example.com',
    'subject'    => 'Hello',
    'text'       => 'Testing some Mailgun awesomness!',
    'o:tracking' => false
);

# Make the call to the client.
$result = $mgClient->messages()->send($domain, $params);
```

```
def send_message_no_tracking():
    return requests.post(
        "https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/messages",
        auth=("api", "YOUR_API_KEY"),
        data={"from": "Excited User <YOU@YOUR_DOMAIN_NAME>",
              "to": ["bar@example.com", "baz@example.com"],
              "subject": "Hello",
              "text": "Testing some Mailgun awesomness!",
              "o:tracking": False})
```

```
def send_message_no_tracking
  RestClient.post "https://api:YOUR_API_KEY"\
  "@api.mailgun.net/v3/YOUR_DOMAIN_NAME/messages",
  :from => "Excited User <YOU@YOUR_DOMAIN_NAME>",
  :to => "bar@example.com, baz@example.com",
  :subject => "Hello",
  :text => "Testing some Mailgun awesomness!",
  "o:tracking" => false
end
```

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class SendMessageNoTrackingChunk
{

    public static void Main (string[] args)
    {
        Console.WriteLine (SendMessageNoTracking ().Content.ToString ());
```

(continues on next page)

(continued from previous page)

```
    }

    public static IRestResponse SendMessageNoTracking ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.AddParameter ("domain", "YOUR_DOMAIN_NAME", ParameterType.UrlSegment);
        request.Resource = "{domain}/messages";
        request.AddParameter ("from", "Excited User <YOU@YOUR_DOMAIN_NAME>");
        request.AddParameter ("to", "bar@example.com");
        request.AddParameter ("to", "baz@example.com");
        request.AddParameter ("subject", "Hello");
        request.AddParameter ("text", "Testing some Mailgun awesomness!");
        request.AddParameter ("o:tracking", false);
        request.Method = Method.POST;
        return client.Execute (request);
    }

}
```

```
import (
    "context"
    "github.com/mailgun/mailgun-go/v3"
    "time"
)

func SendMessageNoTracking(domain, apiKey string) (string, error) {
    mg := mailgun.NewMailgun(domain, apiKey)
    m := mg.NewMessage(
        "Excited User <YOU@YOUR_DOMAIN_NAME>",
        "Hello",
        "Testing some Mailgun awesomeness!",
        "foo@example.com",
    )
    m.SetTracking(false)

    ctx, cancel := context.WithTimeout(context.Background(), time.Second*30)
    defer cancel()

    _, id, err := mg.Send(ctx, m)
    return id, err
}
```

```
var mailgun = require("mailgun-js");
var api_key = 'YOUR_API_KEY';
var DOMAIN = 'YOUR_DOMAIN_NAME';
var mailgun = require('mailgun-js')({apiKey: api_key, domain: DOMAIN});

var data = {
  from: 'Excited User <me@samples.mailgun.org>',
  to: 'alice@example.com',
  subject: 'Hello',
```

(continues on next page)

(continued from previous page)

```
  text: 'Testing some Mailgun awesomeness!',
  "o:tracking": 'False'
};

mailgun.messages().send(data, function (error, body) {
  console.log(body);
});
```

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'
```

```
import java.io.File;

import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode sendScheduledMessage() throws UnirestException {

        HttpResponse<JsonNode> request = Unirest.post("https://api.mailgun.net/v3/" +␣
↪YOUR_DOMAIN_NAME + "/messages")
            .basicAuth("api", API_KEY)
            .field("from", "Excited User <USER@YOURDOMAIN.COM>")
            .field("to", "bruce@example")
            .field("subject", "Bah-weep-graaaaagnah wheep nini bong.")
            .field("text", "Testing some MailGun awesomeness")
            .field("o:deliverytime", "Sat, 20 May 2017 2:50:00 -0000")
            .asJson();

        return request.getBody();
    }
}
```

```
# Include the Autoloader (see "Libraries" for install instructions)
require 'vendor/autoload.php';
use Mailgun\Mailgun;

# Instantiate the client.
$mgClient = Mailgun::create('PRIVATE_API_KEY', 'https://API_HOSTNAME');
$domain = "YOUR_DOMAIN_NAME";
$params = array(
    'from'           => 'Excited User <YOU@YOUR_DOMAIN_NAME>',
    'to'             => 'bob@example.com',
    'subject'        => 'Hello',
```

(continues on next page)

(continued from previous page)

```
    'text'           => 'Testing some Mailgun awesomness!',
    'o:deliverytime' => 'Wed, 01 Jan 2020 09:00:00 -0000'
);

# Make the call to the client.
$result = $mgClient->messages()->send($domain, $params);
```

```
def send_scheduled_message():
    return requests.post(
        "https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/messages",
        auth=("api", "YOUR_API_KEY"),
        data={"from": "Excited User <YOU@YOUR_DOMAIN_NAME>",
              "to": "bar@example.com",
              "subject": "Hello",
              "text": "Testing some Mailgun awesomness!",
              "o:deliverytime": "Fri, 25 Oct 2011 23:10:10 -0000"})
```

```
def send_scheduled_message
  RestClient.post "https://api:YOUR_API_KEY"\
  "@api.mailgun.net/v3/YOUR_DOMAIN_NAME/messages",
  :from => "Excited User <YOU@YOUR_DOMAIN_NAME>",
  :to => "bar@example.com",
  :subject => "Hello",
  :text => "Testing some Mailgun awesomeness!",
  "o:deliverytime" => "Fri, 25 Oct 2011 23:10:10 -0000"
end
```

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class SendScheduledMessageChunk
{

    public static void Main (string[] args)
    {
        Console.WriteLine (SendScheduledMessage ().Content.ToString ());
    }

    public static IRestResponse SendScheduledMessage ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.AddParameter ("domain", "YOUR_DOMAIN_NAME", ParameterType.UrlSegment);
        request.Resource = "{domain}/messages";
        request.AddParameter ("from", "Excited User <YOU@YOUR_DOMAIN_NAME>");
        request.AddParameter ("to", "bar@example.com");
        request.AddParameter ("subject", "Hello");
        request.AddParameter ("text", "Testing some Mailgun awesomness!");
        request.AddParameter ("o:deliverytime",
```

(continues on next page)

(continued from previous page)

```
                                "Fri, 14 Oct 2011 23:10:10 -0000");
        request.Method = Method.POST;
        return client.Execute(request);
    }

}
```

```
import (
    "context"
    "github.com/mailgun/mailgun-go/v3"
    "time"
)

func SendScheduledMessage(domain, apiKey string) (string, error) {
    mg := mailgun.NewMailgun(domain, apiKey)
    m := mg.NewMessage(
        "Excited User <YOU@YOUR_DOMAIN_NAME>",
        "Hello",
        "Testing some Mailgun awesomeness!",
        "bar@example.com",
    )
    m.SetDeliveryTime(time.Now().Add(5 * time.Minute))

    ctx, cancel := context.WithTimeout(context.Background(), time.Second*30)
    defer cancel()

    _, id, err := mg.Send(ctx, m)
    return id, err
}
```

```
const API_KEY = 'YOUR_API_KEY';
const DOMAIN = 'YOUR_DOMAIN_NAME';
const mailgun = require('mailgun-js')({apiKey: API_KEY, domain: DOMAIN});

const data = {
  from: 'Excited User <me@samples.mailgun.org>',
  to: 'bar@example.com',
  subject: 'Scheduled Message',
  text: 'Testing some Mailgun awesomeness!',
  "o:deliverytime": 'Fri, 6 Jul 2017 18:10:10 -0000'
};

mailgun.messages().send(data, (error, body) => {
  console.log(body);
});
```

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'
```

```java
import java.io.File;

import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode sendTaggedMessage() throws UnirestException {

        HttpResponse<JsonNode> request = Unirest.post("https://api.mailgun.net/v3/" +␣
↪YOUR_DOMAIN_NAME + "/messages")
            .basicAuth("api", API_KEY)
            .field("from", "Excited User <YOU@YOUR_DOMAIN_NAME>")
            .field("to", "alice@example")
            .field("subject", "Hello.")
            .field("text", "Testing some Mailgun awesomeness")
            .field("o:tag", "newsletters")
            .field("o:tag", "September newsletter")
            .asJson();

        return request.getBody();
    }
}
```

```php
# Include the Autoloader (see "Libraries" for install instructions)
require 'vendor/autoload.php';
use Mailgun\Mailgun;

# Instantiate the client.
$mgClient = Mailgun::create('PRIVATE_API_KEY', 'https://API_HOSTNAME');
$domain = "YOUR_DOMAIN_NAME";
$params = array(
    'from'    => 'Excited User <YOU@YOUR_DOMAIN_NAME>',
    'to'      => 'bob@example.com',
    'subject' => 'Hello',
    'text'    => 'Testing some Mailgun awesomness!',
    'o:tag'   => array('Tag1', 'Tag2', 'Tag3')
);

# Make the call to the client.
$result = $mgClient->messages()->send($domain, $params);
```

```python
def send_tagged_message():
    return requests.post(
        "https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/messages",
        auth=("api", "YOUR_API_KEY"),
        data={"from": "Excited User <YOU@YOUR_DOMAIN_NAME>",
              "to": "bar@example.com",
              "subject": "Hello",
              "text": "Testing some Mailgun awesomness!",
              "o:tag": ["September newsletter", "newsletters"]})
```

```ruby
def send_tagged_message
  data = {}
  data[:from] = "Excited User <YOU@YOUR_DOMAIN_NAME>"
  data[:to] = "bar@example.com"
  data[:subject] = "Hello"
  data[:text] = "Testing some Mailgun awesomness!"
  data["o:tag"] = []
  data["o:tag"] << "September newsletter"
  data["o:tag"] << "newsletters"
  RestClient.post "https://api:YOUR_API_KEY"\
  "@api.mailgun.net/v3/YOUR_DOMAIN_NAME/messages", data
end
```

```csharp
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class SendTaggedMessageChunk
{

    public static void Main (string[] args)
    {
        Console.WriteLine (SendTaggedMessage ().Content.ToString ());
    }

    public static IRestResponse SendTaggedMessage ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.AddParameter ("domain", "YOUR_DOMAIN_NAME", ParameterType.UrlSegment);
        request.Resource = "{domain}/messages";
        request.AddParameter ("from", "Excited User <YOU@YOUR_DOMAIN_NAME>");
        request.AddParameter ("to", "bar@example.com");
        request.AddParameter ("subject", "Hello");
        request.AddParameter ("text", "Testing some Mailgun awesomness!");
        request.AddParameter ("o:tag", "September newsletter");
        request.AddParameter ("o:tag", "newsletters");
        request.Method = Method.POST;
        return client.Execute (request);
    }

}
```

```go
import (
    "context"
    "github.com/mailgun/mailgun-go/v3"
    "time"
)

func SendTaggedMessage(domain, apiKey string) (string, error) {
    mg := mailgun.NewMailgun(domain, apiKey)
    m := mg.NewMessage(
```

(continues on next page)

(continued from previous page)

```
        "Excited User <YOU@YOUR_DOMAIN_NAME>",
        "Hello",
        "Testing some Mailgun awesomeness!",
        "bar@example.com",
    )

    err := m.AddTag("FooTag", "BarTag", "BlortTag")
    if err != nil {
        return "", err
    }

    ctx, cancel := context.WithTimeout(context.Background(), time.Second*30)
    defer cancel()

    _, id, err := mg.Send(ctx, m)
    return id, err
}
```

```
const API_KEY = 'YOUR_API_KEY';
const DOMAIN = 'YOUR_DOMAIN_NAME';
const mailgun = require('mailgun-js')({apiKey: API_KEY, domain: DOMAIN});

const data = {
  from: 'Excited User <me@samples.mailgun.org>',
  to: 'alice@example',
  subject: 'Tagged',
  text: 'Testing some Mailgun awesomeness!',
  "o:tag" : ['newsletters', 'September newsletter']
};

mailgun.messages().send(data, (error, body) => {
  console.log(body);
});
```

An example of how to resend a message:

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

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode resendSimpleMessge() throws UnirestException {

        HttpResponse<JsonNode> request = Unirest.post("https://se.api.mailgun.net/v3/
↪domains/" + YOUR_DOMAIN_NAME + "/messages/{storage_url}")
            .basicAuth("api", API_KEY)
            .field("to", "user@samples.mailgun.org")
            .asJson();
```

(continues on next page)

(continued from previous page)

```
        return request.getBody();
    }
}
```

```
# Currently, the PHP SDK does not support Resend Messages endpoint.
# Consider using the following php curl function.
function resend_message() {
  $ch = curl_init();

  curl_setopt($ch, CURLOPT_HTTPAUTH, CURLAUTH_BASIC);
  curl_setopt($ch, CURLOPT_USERPWD, 'api:PRIVATE_API_KEY');
  curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);

  curl_setopt($ch, CURLOPT_CUSTOMREQUEST, 'POST');
  curl_setopt($ch, CURLOPT_URL, MESSAGE_STORAGE_URL);
  curl_setopt($ch, CURLOPT_POSTFIELDS, array(
      'to'=> 'bob@example.com'
      )
  );

  $result = curl_exec($ch);
  curl_close($ch);

  return $result;
}
```

```
def resend_simple_message():
    return requests.post(
        "https://se.api.mailgun.net/v3/YOUR_DOMAIN_NAME/messages/STORAGE_URL",
        auth=("api", "YOUR_API_KEY"),
        data={"to": ["bar@example.com", "YOU@YOUR_DOMAIN_NAME"] })
```

```
def resend_simple_message
    RestClient.post "https://api:YOUR_API_KEY"\
    "@se.api.mailgun.net/v3/domains/YOUR_DOMAIN_NAME/messages/STORAGE_URL",
    :to => "bar@example.com, YOU@YOUR_DOMAIN_NAME"
end
```

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class SendSimpleMessageChunk
{

    public static void Main (string[] args)
    {
        Console.WriteLine (ResendSimpleMessage ().Content.ToString ());
    }

    public static IRestResponse ResendSimpleMessage ()
    {
        RestClient client = new RestClient ();
```

(continues on next page)

(continued from previous page)

```
        client.BaseUrl = new Uri ("https://se.api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.AddParameter ("domain", "YOUR_DOMAIN_NAME", ParameterType.UrlSegment);
        request.Resource = "domains/{domain}/messages/STORAGE_URL";
        request.AddParameter ("to", "bar@example.com");
        request.Method = Method.POST;
        return client.Execute (request);
    }

}
```

```
import (
    "context"
    "github.com/mailgun/mailgun-go/v3"
    "time"
)

func ResendMessage(domain, apiKey string) (string, string, error) {
    mg := mailgun.NewMailgun(domain, apiKey)

    ctx, cancel := context.WithTimeout(context.Background(), time.Second*30)
    defer cancel()

    return mg.ReSend(ctx, "STORAGE_URL", "bar@example.com")
}
```

```
var mailgun = require("mailgun-js");
var Request = require('mailgun-js/lib/request');
var api_key = 'YOUR_API_KEY';
var DOMAIN = 'YOUR_DOMAIN_NAME';
var mailgun = require('mailgun-js')({apiKey: api_key, domain: DOMAIN});

var data = {
  "to": 'bar@example.com, alice@example.com',
};

var options = {
   host: 'se.api.mailgun.net',
   endpoint: '/v3',
   protocol: 'https:',
   port: 443,
   auth: ['api', api_key].join(':'),
   retry: 1
 };

var req = new Request(options);

req.request('POST', `/domains/${DOMAIN}/messages/STORAGE_URL`, data, function (error ␣
↪body) {
  console.log(body);
});
```

An example of how to decode Quoted-printable encoded messages:

```
# with utility from `quoted-printable` nodejs lib (`npm install -g quoted-printable`)
decoded_message=$(quoted-printable --decode <<< "${encoded_message}")
```

```
// using Apache Commons / commons-codec
import org.apache.commons.codec.net.QuotedPrintableCodec;

// ...
String decodedMessage = new QuotedPrintableCodec().decode(encodedMessage);
```

```
$decoded_message = quoted_printable_decode($encoded_message);
```

```
import quopri

# ...
decoded_message = quopri.decodestring(encoded_message)
```

```
decoded_message = encoded_message.unpack('M*')
```

```
using System;
using System.Text.RegularExpressions;

public class DecodeMessageSample
{
    // from <http://www.dpit.co.uk/decoding-quoted-printable-email-in-c/>
    private string DecodeQuotedPrintable(string input)
    {
      var occurences = new Regex(@"(=[0-9A-Z][0-9A-Z])+", RegexOptions.Multiline);
      var matches = occurences.Matches(input);
      foreach (Match m in matches)
      {
        byte[] bytes = new byte[m.Value.Length / 3];
        for (int i = 0; i < bytes.Length; i++)
        {
           string hex = m.Value.Substring(i * 3 + 1, 2);
           int iHex = Convert.ToInt32(hex, 16);
           bytes[i] = Convert.ToByte(iHex);
        }
        input = input.Replace(m.Value, Encoding.Default.GetString(bytes));
      }
      return input.Replace("=rn", "");
    }

    public void doThings()
    {
          // ...
       var decodedMessage = DecodeQuotedPrintable(encodedMessage);
    }
}
```

```
import (
   "io/ioutil"
   "mime/quotedprintable"
   "strings"
)
```

(continues on next page)

(continued from previous page)

```
func DecodeMessage(encodedMessage string) (string) {
   decodedMessage, err := ioutil.ReadAll(quotedprintable.NewReader(strings.
↪NewReader(encodedMessage)))
   if err != nil {
       panic(err);
   }
   return decodedMessage
}
```

4.3 Domains

The domains endpoint is available at:

```
v3/domains
```

The domains API allows you to create, access, and validate domains programmatically.

```
GET /domains
```

Returns a list of domains under your account in JSON. See examples below.

Parameter	Description
authority	Filter the list by a given DKIM authority name
state	Filter the list by a given state. Can be `active`, `unverified` or `disabled`
limit	Maximum number of records to return. (100 by default)
skip	Number of records to skip. (0 by default)

```
GET /domains/<domain>
```

Returns a single domain, including credentials and DNS records. See examples below.

```
PUT /domains/<domain>/verify
```

Verifies and returns a single domain, including credentials and DNS records. If the domain is successfully verified the domain's state will be 'active'. For more information on verifying domains, visit the Mailgun User Manual.

```
POST /domains
```

Create a new domain. See examples below.

Parameter	Description
name	Name of the domain (ex. domain.com)
smtp_password	Password for SMTP authentication
spam_action	`disabled`, `block`, or `tag` If `disabled`, no spam filtering will occur for inbound messages. If `block`, inbound spam messages will not be delivered. If `tag`, inbound messages will be tagged with a spam header. See *Spam Filter*. The default is `disabled`.
wildcard	`true` or `false` Determines whether the domain will accept email for sub-domains when sending messages. The default is `false`.
force_dkim_authority	`true` or `false` If set to `true`, the domain will be the DKIM authority for itself even if the root domain is registered on the same mailgun account If set to `false`, the domain will have the same DKIM authority as the root domain registered on the same mailgun account The default is `false`.
dkim_key_size	`1024` or `2048` Set the length of your domain's generated DKIM key The default is `1024`
ips	An optional, comma-separated list of IP addresses to be assigned to this domain. If not specified, all dedicated IP addresses on the account will be assigned. If the request cannot be fulfilled (e.g. a requested IP is not assigned to the account, etc), a `400` will be returned.
pool_id	The id of the IP Pool that you wish to assign to the domain. The pool must contain at least 1 IP. (Note: IP Pools are only available on certain plans; see http://mailgun.com/pricing)
web_scheme	`http` or `https` Set your open, click and unsubscribe URLs to use http or https The default is `http`

```
DELETE /domains/<domain>
```

Delete a domain from your account.

```
GET /domains/<domain>/credentials
```

Returns a list of SMTP credentials for the defined domain.

Parameter	Description
limit	Maximum number of records to return. (100 by default)
skip	Number of records to skip. (0 by default)

```
POST /domains/<domain>/credentials
```

Creates a new set of SMTP credentials for the defined domain.

Parameter	Description
login	The user name, for example `bob.bar`
password	A password for the SMTP credentials. (Length Min 5, Max 32)

```
PUT /domains/<domain>/credentials/<login>
```

Updates the specified SMTP credentials. Currently only the password can be changed.

Parameter	Description
password	A password for the SMTP credentials. (Length Min 5, Max 32)

```
DELETE /domains/<domain>/credentials/<login>
```

Deletes the defined SMTP credentials.

Note: Mailgun imposes a rate limit for the Domains API endpoint. Users may issue no more than 300 requests per minute, per account. See the resultant rate limit response below.

```
GET /domains/<domain>/connection
```

Returns delivery connection settings for the defined domain.

```
PUT /domains/<domain>/connection
```

Updates the specified delivery connection settings for the defined domain.

Parameter	Description
require_tls	*true* or *false* 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 cannot, the message will be delivered over a plaintext SMTP connection. The default is *false*.
skip_verification	*true* or *false* 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*.

```
GET /domains/<domain>/tracking
```

Returns tracking settings for a domain.

```
PUT /domains/<domain>/tracking/open
```

Updates the open tracking settings for a domain.

Parameter	Description
active	`yes` or `no`

```
PUT /domains/<domain>/tracking/click
```

Updates the click tracking settings for a domain.

Parameter	Description
active	`yes`, `no`, or `htmlonly` If set to *yes*, links will be overwritten and pointed to our servers so we can track clicks. If set to *htmlonly*, links will only be rewritten in the HTML part of a message.

```
PUT /domains/<domain>/tracking/unsubscribe
```

Updates unsubscribe tracking settings for a domain.

Parameter	Description
active	*true* or *false*.
html_footer	Custom HTML version of unsubscribe footer.
text_footer	Custom text version of unsubscribe footer. *Mailgun can automatically provide an unsubscribe footer in each email you send and also provides you with several unsubscribe variables. You can customize your unsubscribe footer by editing the settings in the Control Panel.* See *Tracking Unsubscribes* for more details.

```
PUT /domains/<domain>/dkim_authority
```

Change the DKIM authority for a domain.

Parameter	Description
self	*true* or *false* Change the DKIM authority for a domain. If set to *true*, the domain will be the DKIM authority for itself even if the root domain is registered on the same mailgun account If set to *false*, the domain will have the same DKIM authority as the root domain registered on the same mailgun account

Note: Use with caution: Do not forget to change the corresponding DNS record. It can take 24-48 hours for DNS changes to propagate. Changing the DKIM authority of an active domain affects its current deliverability.

Update the DKIM selector for a domains

```
PUT /domains/<domain>/dkim_selector
```

Parameter	Description
dkim_selector	change the DKIM selector for a domain.

Update the CNAME used for tracking opens and clicks

```
PUT /domains/<domain>/web_prefix
```

Parameter	Description
web_prefix	change the tracking CNAME for a domain.

4.3.1 Example

Get a list of all domains.

```
curl -s --user 'api:YOUR_API_KEY' -G \
    https://api.mailgun.net/v3/domains \
    -d skip=0 \
    -d limit=3
```

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;


public class MGSample {

    // ...

    public static JsonNode getDomains() throws UnirestException {

        HttpResponse <JsonNode> request = Unirest.get("https://api.mailgun.net/v3/
↪domains")
            .basicAuth("api", API_KEY)
            .queryString("skip", 0)
            .queryString("limit", 3)
            .asJson();

        return request.getBody();
    }
}
```

```
# Include the Autoloader (see "Libraries" for install instructions)
require 'vendor/autoload.php';
use Mailgun\Mailgun;

# Instantiate the client.
$mgClient = Mailgun::create('PRIVATE_API_KEY', 'https://API_HOSTNAME');

# Issue the call to the client.
$result = $mgClient->domains()->index();
```

```
def get_domains():
    return requests.get(
        "https://api.mailgun.net/v3/domains",
        auth=("api", "YOUR_API_KEY"),
        params={"skip": 0,
                "limit": 3})
```

```
def get_domains
  RestClient.get "https://api:YOUR_API_KEY"\
  "@api.mailgun.net/v3/domains", :params => {
    :skip => 0,
    :limit => 3
  }
end
```

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class GetDomainsChunk
{

    public static void Main (string[] args)
    {
        Console.WriteLine (GetDomains ().Content.ToString ());
    }

    public static IRestResponse GetDomains ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.Resource = "domains";
        request.AddParameter ("skip", 0);
        request.AddParameter ("limit", 3);
        return client.Execute (request);
    }

}
```

```
import (
    "context"
    "github.com/mailgun/mailgun-go/v3"
    "time"
)

func ListDomains(domain, apiKey string) ([]mailgun.Domain, error) {
    mg := mailgun.NewMailgun(domain, apiKey)
    it := mg.ListDomains(nil)

    ctx, cancel := context.WithTimeout(context.Background(), time.Second*30)
    defer cancel()
```

(continues on next page)

(continued from previous page)

```
    var page, result []mailgun.Domain
    for it.Next(ctx, &page) {
        result = append(result, page...)
    }

    if it.Err() != nil {
        return nil, it.Err()
    }
    return result, nil
}
```

```
var DOMAIN = 'YOUR_DOMAIN_NAME';
var mailgun = require('mailgun-js')({ apiKey: "YOUR_API_KEY", domain: DOMAIN });

mailgun.get('/domains', function (error, body) {
  console.log(body);
});
```

Sample response:

```
{
  "total_count": 1,
  "items": [
    {
      "created_at": "Wed, 10 Jul 2013 19:26:52 GMT",
      "smtp_login": "postmaster@samples.mailgun.org",
      "name": "samples.mailgun.org",
      "smtp_password": "4rtqo4p6rrx9",
      "wildcard": true,
      "spam_action": "disabled",
      "state": "active"
    }
  ]
}
```

Get a single domain.

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

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode getDomain() throws UnirestException {

        HttpResponse<JsonNode> request = Unirest.get("https://api.mailgun.net/v3/
↪domains/" + YOUR_DOMAIN_NAME)
            .basicAuth("api", API_KEY)
            .queryString("skip", 0)
```

(continues on next page)

(continued from previous page)

```
            .queryString("limit", 3)
            .asJson();

        return request.getBody();
    }
}
```

```
# Include the Autoloader (see "Libraries" for install instructions)
require 'vendor/autoload.php';
use Mailgun\Mailgun;

# Instantiate the client.
$mgClient = Mailgun::create('PRIVATE_API_KEY', 'https://API_HOSTNAME');
$domain = "YOUR_DOMAIN_NAME";

# Issue the call to the client.
$result = $mgClient->domains()->show($domain);
```

```
def get_domain():
    return requests.get(
        "https://api.mailgun.net/v3/domains/YOUR_DOMAIN_NAME",
        auth=("api", "YOUR_API_KEY"))
```

```
def get_domain
  RestClient.get("https://api:YOUR_API_KEY"\
                 "@api.mailgun.net/v3/domains/YOUR_DOMAIN_NAME"\
                 ){|response, request, result| response }
end
```

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class GetDomainChunk
{

    public static void Main (string[] args)
    {
        Console.WriteLine (GetDomain ().Content.ToString ());
    }

    public static IRestResponse GetDomain ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.AddParameter ("domain", "YOUR_DOMAIN_NAME", ParameterType.UrlSegment);
        request.Resource = "/domains/{domain}";
        return client.Execute (request);
    }
```

(continues on next page)

(continued from previous page)

```
}
```

```
import (
    "context"
    "github.com/mailgun/mailgun-go/v3"
    "time"
)

func GetDomain(domain, apiKey string) (mailgun.DomainResponse, error) {
    mg := mailgun.NewMailgun(domain, apiKey)

    ctx, cancel := context.WithTimeout(context.Background(), time.Second*30)
    defer cancel()

    return mg.GetDomain(ctx, domain)
}
```

```
var DOMAIN = 'YOUR_DOMAIN_NAME';
var mailgun = require('mailgun-js')({ apiKey: "YOUR_API_KEY", domain: DOMAIN });

mailgun.get(`/domain/${DOMAIN}`, function (error, body) {
  console.log(body);
});
```

Sample response:

```
{
  "domain": {
    "created_at": "Wed, 10 Jul 2013 19:26:52 GMT",
    "smtp_login": "postmaster@domain.com",
    "name": "domain.com",
    "smtp_password": "4rtqo4p6rrx9",
    "wildcard": false,
    "spam_action": "tag",
    "state": "active"
  },
  "receiving_dns_records": [
    {
      "priority": "10",
      "record_type": "MX",
      "valid": "valid",
      "value": "mxa.mailgun.org"
    },
    {
      "priority": "10",
      "record_type": "MX",
      "valid": "valid",
      "value": "mxb.mailgun.org"
    }
  ],
  "sending_dns_records": [
    {
      "record_type": "TXT",
      "valid": "valid",
      "name": "domain.com",
      "value": "v=spf1 include:mailgun.org ~all"
```

(continues on next page)

(continued from previous page)

```
    },
    {
      "record_type": "TXT",
      "valid": "valid",
      "name": "domain.com",
      "value": "k=rsa; p=MIGfMA0GCSqGSIb3DQEBAQUA...."
    },
    {
      "record_type": "CNAME",
      "valid": "valid",
      "name": "email.domain.com",
      "value": "mailgun.org"
    }
  ]
}
```

Adding a domain.

```
curl -s --user 'api:YOUR_API_KEY' \
  -X POST https://api.mailgun.net/v3/domains \
  -F name='YOUR_NEW_DOMAIN_NAME' \
  -F smtp_password='supersecretpassword'
```

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode addDomain() throws UnirestException {

        HttpResponse<JsonNode> jsonResponse = Unirest.post("https://api.mailgun.net/
↪v3/domains")
            .basicAuth("api", API_KEY)
            .field("name", "YOUR_NEW_DOMAIN_NAME")
            .field("smtp_password", "supersecretpassword")
            .asJson();

        return jsonResponse.getBody();
    }
}
```

```
# Include the Autoloader (see "Libraries" for install instructions)
require 'vendor/autoload.php';
use Mailgun\Mailgun;

# Instantiate the client.
$mgClient = Mailgun::create('PRIVATE_API_KEY', 'https://API_HOSTNAME');
$domain   = 'YOUR_DOMAIN_NAME';

# Issue the call to the client.
$result = $mgClient->domains()->create($domain);
```

```python
def add_domain():
    return requests.post(
        "https://api.mailgun.net/v3/domains",
        auth=("api", "YOUR_API_KEY"),
        data={'name':'YOUR_NEW_DOMAIN_NAME', 'smtp_password':'supersecretpassword'})
```

```ruby
def add_domain
  RestClient.post("https://api:YOUR_API_KEY"\
                  "@api.mailgun.net/v3/domains",
                  :name => 'YOUR_NEW_DOMAIN_NAME',
                  :smtp_password => 'supersecretpassword')
end
```

```c#
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class AddDomainChunk
{

    public static void Main (string[] args)
    {
        Console.WriteLine (AddDomain ().Content.ToString ());
    }

    public static IRestResponse AddDomain ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3/");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.Resource = "domains";
        request.AddParameter ("name", "YOUR_NEW_DOMAIN_NAME");
        request.AddParameter ("smtp_password", "supersecretpassword");
        request.Method = Method.POST;
        return client.Execute (request);
    }

}
```

```go
import (
    "context"
    "github.com/mailgun/mailgun-go/v3"
    "time"
)

func AddDomain(domain, apiKey string) error {
    mg := mailgun.NewMailgun(domain, apiKey)

    ctx, cancel := context.WithTimeout(context.Background(), time.Second*30)
    defer cancel()

    return mg.CreateDomain(ctx, "example.com", &mailgun.CreateDomainOptions{
```

(continues on next page)

(continued from previous page)

```
        Password: "super_secret",
        SpamAction: mailgun.SpamActionTag,
        Wildcard:   false,
    })
}
```

```
var DOMAIN = 'YOUR_DOMAIN_NAME';
var mailgun = require('mailgun-js')({ apiKey: "YOUR_API_KEY", domain: DOMAIN });

mailgun.post('/domains', {"name" : "YOUR_NEW_DOMAIN_NAME", "smtp_password" :
↪"supersecret"}, function (error, body) {
  console.log(body);
});
```

Sample response:

```
{
  "domain": {
    "name": "example.com",
    "created_at": "Fri, 22 Nov 2013 18:42:33 GMT",
    "wildcard": false,
    "spam_action": "disabled",
    "smtp_login": "postmaster@example.com",
    "smtp_password": "thiswontwork",
    "state": "active"
  },
  "receiving_dns_records": [
    {
      "priority": "10",
      "record_type": "MX",
      "valid": "valid",
      "value": "mxa.mailgun.org"
    },
    {
      "priority": "10",
      "record_type": "MX",
      "valid": "valid",
      "value": "mxb.mailgun.org"
    }
  ],
  "message": "Domain has been created",
  "sending_dns_records": [
    {
      "record_type": "TXT",
      "valid": "valid",
      "name": "example.com",
      "value": "v=spf1 include:mailgun.org ~all"
    },
    {
      "record_type": "TXT",
      "valid": "valid",
      "name": "k1._domainkey.example.com",
      "value": "k=rsa; p=MIGfMA0GCSqGSIb3DQEBAQUAA4G...."
    },
    {
      "record_type": "CNAME",
```

(continues on next page)

(continued from previous page)

```
      "valid": "valid",
      "name": "email.example.com",
      "value": "mailgun.org"
    }
  ]
}
```

Deleting a domain.

```
curl -s --user 'api:YOUR_API_KEY' -X DELETE \
    https://api.mailgun.net/v3/domains/example.mailgun.org
```

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode deleteDomain() throws UnirestException {

        HttpResponse<JsonNode> request = Unirest.delete("https://api.mailgun.net/v3/
↪domains/domain.example.com")
            .basicAuth("api", API_KEY)
            .asJson();

        return request.getBody();
    }
}
```

```
# Include the Autoloader (see "Libraries" for install instructions)
require 'vendor/autoload.php';
use Mailgun\Mailgun;

# Instantiate the client.
$mgClient = Mailgun::create('PRIVATE_API_KEY', 'https://API_HOSTNAME');
$domain   = 'YOUR_DOMAIN_NAME';

# Issue the call to the client.
$result = $mgClient->domains()->delete($domain);
```

```
def delete_domain():
    return requests.delete(
        "https://api.mailgun.net/v3/domains/example.mailgun.org",
        auth=("api", "YOUR_API_KEY"))
```

```
def delete_domain
  RestClient.delete "https://api:YOUR_API_KEY"\
  "@api.mailgun.net/v3/domains/example.mailgun.org"
end
```

```
using System;
using System.IO;
```

(continues on next page)

(continued from previous page)

```
using RestSharp;
using RestSharp.Authenticators;

public class DeleteDomainChunk
{

    public static void Main (string[] args)
    {
        Console.WriteLine (DeleteDomain ().Content.ToString ());
    }

    public static IRestResponse DeleteDomain ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.Resource = "/domains/{name}";
        request.AddUrlSegment ("name", "example.mailgun.org");
        request.Method = Method.DELETE;
        return client.Execute (request);
    }

}
```

```
import (
    "context"
    "github.com/mailgun/mailgun-go/v3"
    "time"
)

func DeleteDomain(domain, apiKey string) error {
    mg := mailgun.NewMailgun(domain, apiKey)

    ctx, cancel := context.WithTimeout(context.Background(), time.Second*30)
    defer cancel()

    return mg.DeleteDomain(ctx, "example.com")
}
```

```
var DOMAIN = 'YOUR_DOMAIN_NAME';
var mailgun = require('mailgun-js')({ apiKey: "YOUR_API_KEY", domain: DOMAIN });

mailgun.delete('/domains/example.mailgun.org', function (error, body) {
  console.log(body);
});
```

Sample response:

```
{
  "message": "Domain has been deleted"
}
```

Rate Limit Response:

```
{
  "retry-seconds": 60,
}
```

Listing all SMTP credentials:

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

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode getCredentials() throws UnirestException {

        HttpResponse<JsonNode> request = Unirest.get("https://api.mailgun.net/v3/
↪domains/" + YOUR_DOMAIN_NAME + "/credentials")
            .basicAuth("api", API_KEY)
            .asJson();

        return request.getBody();
    }
}
```

```
# Include the Autoloader (see "Libraries" for install instructions)
require 'vendor/autoload.php';
use Mailgun\Mailgun;

# Instantiate the client.
$mgClient = Mailgun::create('PRIVATE_API_KEY', 'https://API_HOSTNAME');
$domain = "YOUR_DOMAIN_NAME";

# Issue the call to the client.
$result = $mgClient->domains()->credentials($domain);
```

```
def get_credentials():
    return requests.get(
        "https://api.mailgun.net/v3/domains/YOUR_DOMAIN_NAME/credentials",
        auth=("api", "YOUR_API_KEY"))
```

```
def get_credentials
  RestClient.get "https://api:YOUR_API_KEY"\
  "@api.mailgun.net/v3/domains/YOUR_DOMAIN_NAME/credentials"
end
```

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

```

(continues on next page)

(continued from previous page)

```
public class GetCredentialsChunk
{

    public static void Main (string[] args)
    {
        Console.WriteLine (GetCredentials ().Content.ToString ());
    }

    public static IRestResponse GetCredentials ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.AddParameter ("domain", "YOUR_DOMAIN_NAME", ParameterType.UrlSegment);
        request.Resource = "domains/{domain}/credentials";
        return client.Execute (request);
    }

}
```

```
import (
    "context"
    "github.com/mailgun/mailgun-go/v3"
    "time"
)

func ListCredentials(domain, apiKey string) ([]mailgun.Credential, error) {
    mg := mailgun.NewMailgun(domain, apiKey)
    it := mg.ListCredentials(nil)

    ctx, cancel := context.WithTimeout(context.Background(), time.Second*30)
    defer cancel()

    var page, result []mailgun.Credential
    for it.Next(ctx, &page) {
        result = append(result, page...)
    }

    if it.Err() != nil {
        return nil, it.Err()
    }
    return result, nil
}
```

```
var DOMAIN = 'YOUR_DOMAIN_NAME';
var mailgun = require('mailgun-js')({ apiKey: "YOUR_API_KEY", domain: DOMAIN });

mailgun.get(`/domains/${DOMAIN}/credentials`, function (error, body) {
  console.log(body);
});
```

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 new SMTP credentials:

```
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'
```

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode createCredentials() throws UnirestException {

        HttpResponse<JsonNode> jsonResponse = Unirest.post("https://api.mailgun.net/
↪v3/domains/" + YOUR_DOMAIN_NAME +"/credentials")
            .basicAuth("api", API_KEY)
            .field("login", "alice@YOUR_DOMAIN_NAME.com")
            .field("password", "supersecretpassword")
            .asJson();

        return jsonResponse.getBody();
    }
}
```

```
# Include the Autoloader (see "Libraries" for install instructions)
require 'vendor/autoload.php';
use Mailgun\Mailgun;

# Instantiate the client.
$mgClient = Mailgun::create('PRIVATE_API_KEY', 'https://API_HOSTNAME');
$domain = 'YOUR_DOMAIN_NAME';
$smtpUser = 'bob';
$smtpPass = 'new_password';
```

(continues on next page)

(continued from previous page)

```
# Issue the call to the client.
$result = $mgClient->domains()->createCredential($domain, $smtpUser, $smtpPass);
```

```
def create_credentials():
    return requests.post(
        "https://api.mailgun.net/v3/domains/YOUR_DOMAIN_NAME/credentials",
        auth=("api", "YOUR_API_KEY"),
        data={"login": "alice@YOUR_DOMAIN_NAME",
              "password": "secret"})
```

```
def create_credentials
  RestClient.post "https://api:YOUR_API_KEY"\
  "@api.mailgun.net/v3/domains/YOUR_DOMAIN_NAME/credentials",
  :login => "alice@YOUR_DOMAIN_NAME",
  :password => "secret"
end
```

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class CreateCredentialsChunk
{

    public static void Main (string[] args)
    {
        Console.WriteLine (CreateCredentials ().Content.ToString ());
    }

    public static IRestResponse CreateCredentials ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.AddParameter ("domain", "YOUR_DOMAIN_NAME", ParameterType.UrlSegment);
        request.Resource = "domains/{domain}/credentials";
        request.AddParameter ("login", "alice@YOUR_DOMAIN_NAME");
        request.AddParameter ("password", "secret");
        request.Method = Method.POST;
        return client.Execute (request);
    }

}
```

```
import (
    "context"
    "github.com/mailgun/mailgun-go/v3"
    "time"
)

func CreateCredential(domain, apiKey string) error {
```

(continues on next page)

(continued from previous page)

```
    mg := mailgun.NewMailgun(domain, apiKey)

    ctx, cancel := context.WithTimeout(context.Background(), time.Second*30)
    defer cancel()

    return mg.CreateCredential(ctx, "alice@example.com", "secret")
}
```

```
var DOMAIN = 'YOUR_DOMAIN_NAME';
var mailgun = require('mailgun-js')({ apiKey: "YOUR_API_KEY", domain: DOMAIN });

mailgun.post(`/domains/${DOMAIN}/credentials`, {"login": "alice@YOUR_DOMAIN_NAME",
↪"password": "secret"}, function (error, body) {
  console.log(body);
});
```

Sample response:

```
{
  "message": "Created 1 credentials pair(s)"
}
```

Updating the password for a given credential pair:

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

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode updatePassword() throws UnirestException {

        HttpResponse<JsonNode> jsonResponse = Unirest.put("https://api.mailgun.net/v3/
↪domains/YOUR_DOMAIN_NAME/credentials/alice")
            .basicAuth("api", API_KEY)
            .field("password", "supersecret")
            .asJson();

        return jsonResponse.getBody();
    }
}
```

```
# Include the Autoloader (see "Libraries" for install instructions)
require 'vendor/autoload.php';
use Mailgun\Mailgun;

# Instantiate the client.
$mgClient = Mailgun::create('PRIVATE_API_KEY', 'https://API_HOSTNAME');
```

(continues on next page)

(continued from previous page)

```
$domain   = 'YOUR_DOMAIN_NAME';
$smtpUser = 'bob';
$smtpPass = 'new_password';

# Issue the call to the client.
$result = $mgClient->domains()->updateCredential($domain, $smtpUser, $smtpPass);
```

```
def change_credential_password():
    return requests.put(
        "https://api.mailgun.net/v3/domains/YOUR_DOMAIN_NAME/credentials/alice",
        auth=("api", "YOUR_API_KEY"),
        data={"password": "supersecret"})
```

```
def change_credential_password
  RestClient.put "https://api:YOUR_API_KEY"\
  "@api.mailgun.net/v3/domains/YOUR_DOMAIN_NAME/credentials/alice",
  :password => "supersecret"
end
```

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class ChangePwdCredentialsChunk
{

    public static void Main (string[] args)
    {
        Console.WriteLine (ChangeCredentialPassword ().Content.ToString ());
    }

    public static IRestResponse ChangeCredentialPassword ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.AddParameter ("domain", "YOUR_DOMAIN_NAME", ParameterType.UrlSegment);
        request.Resource = "domains/{domain}/credentials/{username}";
        request.AddUrlSegment ("username", "alice");
        request.AddParameter ("password", "supersecret");
        request.Method = Method.PUT;
        return client.Execute (request);
    }


}
```

```
import (
    "context"
    "github.com/mailgun/mailgun-go/v3"
    "time"
)
```

(continues on next page)

(continued from previous page)

```
func ChangePassword(domain, apiKey string) error {
    mg := mailgun.NewMailgun(domain, apiKey)

    ctx, cancel := context.WithTimeout(context.Background(), time.Second*30)
    defer cancel()

    return mg.ChangeCredentialPassword(ctx, "alice", "super_secret")
}
```

```
var DOMAIN = 'YOUR_DOMAIN_NAME';
var mailgun = require('mailgun-js')({ apiKey: "YOUR_API_KEY", domain: DOMAIN });

mailgun.put(`/domains/${DOMAIN}/credentials/alice`, {"password" : "supersecret"},␣
↪function (error, body) {
  console.log(body);
});
```

Sample response:

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

Deleting a given credential pair:

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

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode deleteCredentials() throws UnirestException {

        HttpResponse<JsonNode> request = Unirest.delete("https://api.mailgun.net/v3/
↪domains/"+ YOUR_DOMAIN_NAME +"/credentials/user")
            .basicAuth("api", API_KEY)
            .asJson();

        return request.getBody();
    }
}
```

```
# Include the Autoloader (see "Libraries" for install instructions)
require 'vendor/autoload.php';
use Mailgun\Mailgun;

# Instantiate the client.
$mgClient = Mailgun::create('PRIVATE_API_KEY', 'https://API_HOSTNAME');
```

(continues on next page)

(continued from previous page)

```
$domain   = 'YOUR_DOMAIN_NAME';
$smtpUser = 'bob';

# Issue the call to the client.
$result = $mgClient->domains()->deleteCredential($domain, $smtpUser);
```

```
def delete_credentials():
    return requests.delete(
        "https://api.mailgun.net/v3/domains/YOUR_DOMAIN_NAME/credentials/alice",
        auth=("api", "YOUR_API_KEY"))
```

```
def delete_credentials
  RestClient.delete "https://api:YOUR_API_KEY"\
  "@api.mailgun.net/v3/domains/YOUR_DOMAIN_NAME/credentials/alice"
end
```

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class DeleteCredentialsChunk
{

    public static void Main (string[] args)
    {
        Console.WriteLine (DeleteCredentials ().Content.ToString ());
    }

    public static IRestResponse DeleteCredentials ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.AddParameter ("domain", "YOUR_DOMAIN_NAME", ParameterType.UrlSegment);
        request.Resource = "domains/{domain}/credentials/{login}";
        request.AddUrlSegment ("login", "alice");
        request.Method = Method.DELETE;
        return client.Execute (request);
    }

}
```

```
import (
    "context"
    "github.com/mailgun/mailgun-go/v3"
    "time"
)

func DeleteCredential(domain, apiKey string) error {
    mg := mailgun.NewMailgun(domain, apiKey)
```

(continues on next page)

(continued from previous page)

```
    ctx, cancel := context.WithTimeout(context.Background(), time.Second*30)
    defer cancel()

    return mg.DeleteCredential(ctx, "alice")
}
```

```
var DOMAIN = 'YOUR_DOMAIN_NAME';
var mailgun = require('mailgun-js')({ apiKey: "YOUR_API_KEY", domain: DOMAIN });

mailgun.put(`/domains/${DOMAIN}/credentials/alice`, function (error, body) {
  console.log(body);
});
```

Sample response:

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

Listing connection settings:

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

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode getConnections() throws UnirestException {

        HttpResponse<JsonNode> request = Unirest.get("https://api.mailgun.net/v3/
↪domains/"+ YOUR_DOMAIN_NAME +"/connection")
            .basicAuth("api", API_KEY)
            .asJson();

        return request.getBody();
    }
}
```

```
# Include the Autoloader (see "Libraries" for install instructions)
require 'vendor/autoload.php';
use Mailgun\Mailgun;

# Instantiate the client.
$mgClient = Mailgun::create('PRIVATE_API_KEY', 'https://API_HOSTNAME');
$domain = "YOUR_DOMAIN_NAME";

# Issue the call to the client.
$result = $mgClient->domains()->connection($domain);
```

```python
def get_connection():
    return requests.get(
        "https://api.mailgun.net/v3/domains/YOUR_DOMAIN_NAME/connection",
        auth=("api", "YOUR_API_KEY"))
```

```ruby
def get_connection
  RestClient.get "https://api:YOUR_API_KEY"\
  "@api.mailgun.net/v3/domains/YOUR_DOMAIN_NAME/connection"
end
```

```csharp
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class GetConnectionChunk
{

    public static void Main (string[] args)
    {
        Console.WriteLine (GetConnection ().Content.ToString ());
    }

    public static IRestResponse GetConnection ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.AddParameter ("domain", "YOUR_DOMAIN_NAME", ParameterType.UrlSegment);
        request.Resource = "domains/{domain}/connection";
        return client.Execute (request);
    }

}
```

```golang
import (
    "context"
    "github.com/mailgun/mailgun-go/v3"
    "time"
)

func GetDomainConnection(domain, apiKey string) (mailgun.DomainConnection, error) {
    mg := mailgun.NewMailgun(domain, apiKey)

    ctx, cancel := context.WithTimeout(context.Background(), time.Second*30)
    defer cancel()

    return mg.GetDomainConnection(ctx, domain)
}
```

```nodejs
var DOMAIN = 'YOUR_DOMAIN_NAME';
var mailgun = require('mailgun-js')({ apiKey: "YOUR_API_KEY", domain: DOMAIN });
```

(continues on next page)

(continued from previous page)

```
mailgun.get(`/domains/${DOMAIN}/connection`, function (error, body) {
  console.log(body);
});
```

Sample response:

```
{
  "connection": {
    "require_tls": false,
    "skip_verification": false
}
```

Update connection settings:

```
curl -s --user 'api:YOUR_API_KEY' -X PUT \
    https://api.mailgun.net/v3/domains/YOUR_DOMAIN_NAME/connection
    -F require_tls='true' \
    -F skip_verification='false'
```

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode updateConnections() throws UnirestException {

        HttpResponse<JsonNode> jsonResponse = Unirest.put("https://api.mailgun.net/v3/
↪domains/"+ YOUR_DOMAIN_NAME +"/connection")
            .basicAuth("api", API_KEY)
            .field("require_tls", true)
            .field("skip_verification", false)
            .asJson();

        return jsonResponse.getBody();
    }
}
```

```
# Include the Autoloader (see "Libraries" for install instructions)
require 'vendor/autoload.php';
use Mailgun\Mailgun;

# Instantiate the client.
$mgClient = Mailgun::create('PRIVATE_API_KEY', 'https://API_HOSTNAME');
$domain = "YOUR_DOMAIN_NAME";
$require_tls = true;
$skip_verification = false;

# Issue the call to the client.
$result = $mgClient->domains()->updateConnection($domain, $require_tls, $skip_
↪verification);
```

```
def update_connection():
    return requests.put(
        ("https://api.mailgun.net/v3/domains/YOUR_DOMAIN_NAME/connection"),
        auth=('api', 'YOUR_API_KEY'),
        data={'require_tls': True,
              'skip_verification': False})
```

```
def update_member
  RestClient.put("https://api:YOUR_API_KEY" \
                 "@api.mailgun.net/v3/domains/YOUR_DOMAIN_NAME/connection",
                 :require_tls => true,
                 :skip_verification => false)
end
```

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class UpdateConnectionChunk
{

    public static void Main (string[] args)
    {
        Console.WriteLine (UpdateConnection ().Content.ToString ());
    }

    public static IRestResponse UpdateConnection ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.Resource = "domains/YOUR_DOMAIN_NAME/connection";
        request.AddParameter ("require_tls", true);
        request.AddParameter ("skip_verification", false);
        request.Method = Method.PUT;
        return client.Execute (request);
    }

}
```

```
import (
    "context"
    "github.com/mailgun/mailgun-go/v3"
    "time"
)

func UpdateDomainConnection(domain, apiKey string) error {
    mg := mailgun.NewMailgun(domain, apiKey)

    ctx, cancel := context.WithTimeout(context.Background(), time.Second*30)
    defer cancel()
```

(continues on next page)

(continued from previous page)

```
    return mg.UpdateDomainConnection(ctx, domain, mailgun.DomainConnection{
        RequireTLS:       true,
        SkipVerification: true,
    })
}
```

```
var DOMAIN = 'YOUR_DOMAIN_NAME';
var mailgun = require('mailgun-js')({ apiKey: "YOUR_API_KEY", domain: DOMAIN });

mailgun.put(`/domain/${DOMAIN}/connection`, {"require_tls": true, "skip_verification
↪": false}, function (error, body) {
  console.log(body);
});
```

Sample response:

```
{
  "message": "Domain connection settings have been updated, may take 10 minutes to␣
↪fully propagate",
  "require-tls": true,
  "skip-verification": false
}
```

Get tracking settings for a domain:

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

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode getDomainTracking() throws UnirestException {

        HttpResponse<JsonNode> request = Unirest.get("https://api.mailgun.net/v3/
↪domains/" + YOUR_DOMAIN_NAME + "/tracking")
            .basicAuth("api", API_KEY)
            .asJson();

        return request.getBody();
    }
}
```

```
# Currently, the PHP SDK does not support Mailing List validations.
# Consider using the following php curl function.
function get_domain_tracking_settings() {
  $ch = curl_init();

  curl_setopt($ch, CURLOPT_HTTPAUTH, CURLAUTH_BASIC);
  curl_setopt($ch, CURLOPT_USERPWD, 'api:PRIVATE_API_KEY');
```

(continues on next page)

(continued from previous page)

```
  curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);

  curl_setopt($ch, CURLOPT_CUSTOMREQUEST, 'GET');
  curl_setopt($ch, CURLOPT_URL, 'https://api.mailgun.net/v3/domains/YOUR_DOMAIN_NAME/
↪tracking');

  $result = curl_exec($ch);
  curl_close($ch);

  return $result;
}
```

```
def get_domain_tracking():
    return requests.get(
        "https://api.mailgun.net/v3/domains/YOUR_DOMAIN_NAME/tracking",
        auth=("api", "YOUR_API_KEY"))
```

```
def get_domain_tracking
  RestClient.get("https://api:YOUR_API_KEY"\
                 "@api.mailgun.net/v3/domains/YOUR_DOMAIN_NAME/tracking"\
                 {|response, request, result| response }
end
```

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class GetDomainTrackingChunk
{

    public static void Main (string[] args)
    {
        Console.WriteLine (GetDomainTracking ().Content.ToString ());
    }

    public static IRestResponse GetDomainTracking ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.AddParameter ("domain", "YOUR_DOMAIN_NAME", ParameterType.UrlSegment);
        request.Resource = "/domains/{domain}/tracking";
        return client.Execute (request);
    }

}
```

```
import (
    "context"
    "github.com/mailgun/mailgun-go/v3"
    "time"
```

(continues on next page)

(continued from previous page)

```
}

func GetDomainTracking(domain, apiKey string) (mailgun.DomainTracking, error) {
    mg := mailgun.NewMailgun(domain, apiKey)

    ctx, cancel := context.WithTimeout(context.Background(), time.Second*30)
    defer cancel()

    return mg.GetDomainTracking(ctx, domain)
}
```

```
var DOMAIN = 'YOUR_DOMAIN_NAME';
var mailgun = require('mailgun-js')({ apiKey: "YOUR_API_KEY", domain: DOMAIN });

mailgun.get(`/domains/${DOMAIN}/tracking`, function (error, body) {
  console.log(body);
});
```

Sample response:

```
{
"tracking": {
  "click": {
    "active": false
  },
  "open": {
    "active": false
  },
  "unsubscribe": {
    "active": false,
    "html_footer": "\n<br>\n<p><a href=\"%unsubscribe_url%\">unsubscribe</a></p>\n",
    "text_footer": "\n\nTo unsubscribe click: <%unsubscribe_url%>\n\n"
  }
 }
}
```

4.4 IPs

The IP API endpoint allows you to access information regarding the IPs allocated to your Mailgun account that are used for outbound sending. The IP endpoint is available at:

```
v3/ips
```

Note: You can manage your IPs from the Control Panel. Click on **IP Management** in the settings dropdown menu.

```
GET /ips
```

Returns a list of IPs assigned to your account. See examples below.

Parameter	Description
dedicated	Return only dedicated IPs if set to *true*. (all are returned by default)

```
GET /ips/<ip>
```

Returns information about the specified IP. See examples below.

```
GET /domains/<domain>/ips
```

Returns a list of IPs currently assigned to the specified domain.

```
POST /domains/<domain>/ips
```

Assign a dedicated IP to the domain specified.

Note: Only dedicated IPs can be assigned to a domain.

Parameter	Description
ip	IP address that should be assigned to the domain pool.

```
DELETE /domains/<domain>/ips/<ip>
```

Unassign an IP from the domain specified.

4.4.1 Example

Get a list of all IPs.

```
curl -s --user 'api:YOUR_API_KEY' -G \
    https://api.mailgun.net/v3/ips \
    -d dedicated="true"
```

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode getIPs() throws UnirestException {

        HttpResponse<JsonNode> request = Unirest.get("https://api.mailgun.net/v3/ips")
            .basicAuth("api", API_KEY)
            .queryString("dedicated", "true")
            .asJson();

        return request.getBody();
    }
}
```

```
# Include the Autoloader (see "Libraries" for install instructions)
require 'vendor/autoload.php';
use Mailgun\Mailgun;
```

(continues on next page)

(continued from previous page)

```
# Instantiate the client.
$mgClient = Mailgun::create('PRIVATE_API_KEY', 'https://API_HOSTNAME');

# Issue the call to the client.
$result = $mgClient->ips->index();
```

```
def get_ips():
    return requests.get(
        "https://api.mailgun.net/v3/ips",
        params={"dedicated": "true"},
        auth=("api", "YOUR_API_KEY"))
```

```
def get_ips
  RestClient.get("https://api:YOUR_API_KEY"\
                 "@api.mailgun.net/v3/ips?dedicated=true"\
                 {|response, request, result| response }
end
```

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class GetIPsChunk
{

    public static void Main (string[] args)
    {
        Console.WriteLine (GetIPs ().Content.ToString ());
    }

    public static IRestResponse GetIPs ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.Resource = "ips";
        request.AddParameter ("dedicated", "true");
        return client.Execute (request);
    }


}
```

```
import (
    "context"
    "github.com/mailgun/mailgun-go/v3"
    "time"
)

func ListIPS(domain, apiKey string) ([]mailgun.IPAddress, error) {
    mg := mailgun.NewMailgun(domain, apiKey)
```

(continues on next page)

(continued from previous page)

```
    ctx, cancel := context.WithTimeout(context.Background(), time.Second*30)
    defer cancel()

    // Pass 'true' to only return dedicated ips
    return mg.ListIPS(ctx, true)
}
```

```
var DOMAIN = 'YOUR_DOMAIN_NAME';
var mailgun = require('mailgun-js')({ apiKey: "YOUR_API_KEY", domain: DOMAIN });

mailgun.get('ips', function (error, body) {
  console.log(body);
});
```

```
{
  "items": ["192.161.0.1", "192.168.0.2"],
  "total_count": 2
}
```

Get a single IP.

```
curl -s --user 'api:YOUR_API_KEY' -G \
    https://api.mailgun.net/v3/ips/127.0.01
```

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode getIP() throws UnirestException {

        HttpResponse<JsonNode> request = Unirest.get("https://api.mailgun.net/v3/ips/
↪127.0.0.1")
            .basicAuth("api", API_KEY)
            .asJson();

        return request.getBody();
    }
}
```

```
# Include the Autoloader (see "Libraries" for install instructions)
require 'vendor/autoload.php';
use Mailgun\Mailgun;

# Instantiate the client.
$mgClient = Mailgun::create('PRIVATE_API_KEY', 'https://API_HOSTNAME');
$ip       = '127.0.0.1';

# Issue the call to the client.
$result = $mgClient->ips()->show($ip);
```

```python
def get_ip():
    return requests.get(
        "https://api.mailgun.net/v3/ips/127.0.0.1",
        auth=("api", "YOUR_API_KEY"))
```

```ruby
def get_ip
  RestClient.get("https://api:YOUR_API_KEY"\
                 "@api.mailgun.net/v3/ips/127.0.0.1"\
                 ){|response, request, result| response }
end
```

```c#
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class GetIPChunk
{

    public static void Main (string[] args)
    {
        Console.WriteLine (GetIP ().Content.ToString ());
    }

    public static IRestResponse GetIP ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.AddParameter ("ip", "127.0.0.1", ParameterType.UrlSegment);
        request.Resource = "/ips/{ip}";
        return client.Execute (request);
    }

}
```

```go
import (
    "context"
    "github.com/mailgun/mailgun-go/v3"
    "time"
)

func GetIP(domain, apiKey string) (mailgun.IPAddress, error) {
    mg := mailgun.NewMailgun(domain, apiKey)

    ctx, cancel := context.WithTimeout(context.Background(), time.Second*30)
    defer cancel()

    return mg.GetIP(ctx, "127.0.0.1")
}
```

```node
var DOMAIN = 'YOUR_DOMAIN_NAME';
var mailgun = require('mailgun-js')({ apiKey: "YOUR_API_KEY", domain: DOMAIN });
```

(continues on next page)

(continued from previous page)

```
mailgun.get('/ips/127.0.0.1', function (error, body) {
  console.log(body);
});
```

```
{
  "ip": "192.161.0.1",
  "dedicated": true,
  "rdns": "luna.mailgun.net"
}
```

Get a list of IPs from the domain pool.

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

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode getDomainIPs() throws UnirestException {

        HttpResponse<JsonNode> request = Unirest.get("https://api.mailgun.net/v3/
↪domains/" + YOUR_DOMAIN_NAME + "/ips")
            .basicAuth("api", API_KEY)
            .asJson();

        return request.getBody();
    }
}
```

```
# Include the Autoloader (see "Libraries" for install instructions)
require 'vendor/autoload.php';
use Mailgun\Mailgun;

# Instantiate the client.
$mgClient = Mailgun::create('PRIVATE_API_KEY', 'https://API_HOSTNAME');
$domain = "YOUR_DOMAIN_NAME";

# Issue the call to the client.
$result = $mgClient->ips()->domainIndex($domain);
```

```
def get_domain_ips():
    return requests.get(
        "https://api.mailgun.net/v3/domains/YOUR_DOMAIN_NAME/ips",
        auth=("api", "YOUR_API_KEY"))
```

```
def get_domain_ips
  RestClient.get("https://api:YOUR_API_KEY"\
                 "@api.mailgun.net/v3/domains/YOUR_DOMAIN_NAME/ips"\
```

(continues on next page)

(continued from previous page)

```
                {|response, request, result| response }
end
```

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class GetDomainIPsChunk
{

    public static void Main (string[] args)
    {
        Console.WriteLine (GetDomainIPs ().Content.ToString ());
    }

    public static IRestResponse GetDomainIPs ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.AddParameter ("domain", "YOUR_DOMAIN_NAME", ParameterType.UrlSegment);
        request.Resource = "/domains/{domain}/ips";
        return client.Execute (request);
    }

}
```

```
import (
    "context"
    "github.com/mailgun/mailgun-go/v3"
    "time"
)

func ListDomainIPS(domain, apiKey string) ([]mailgun.IPAddress, error) {
    mg := mailgun.NewMailgun(domain, apiKey)

    ctx, cancel := context.WithTimeout(context.Background(), time.Second*30)
    defer cancel()

    return mg.ListDomainIPS(ctx)
}
```

```
var DOMAIN = 'YOUR_DOMAIN_NAME';
var mailgun = require('mailgun-js')({ apiKey: "YOUR_API_KEY", domain: DOMAIN });

mailgun.get(`/domains/${DOMAIN}/ips`, function (error, body) {
  console.log(body);
});
```

```
{
  "items": ["192.161.0.1", "192.168.0.2"],
```

(continues on next page)

(continued from previous page)

```
  "total_count": 2
}
```

Assign a dedicated IP to the domain pool.

```
curl -s --user 'api:YOUR_API_KEY' \
   -X POST https://api.mailgun.net/v3/domains/YOUR_DOMAIN_NAME/ips \
   -F ip='127.0.0.1'
```

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode addDomainIP() throws UnirestException {

        HttpResponse<JsonNode> jsonResponse = Unirest.post("https://api.mailgun.net/
↪v3/domains/YOUR_DOMAIN_NAME/ips")
            .basicAuth("api", API_KEY)
            .field("ip", "127.0.0.1")
            .asJson();

        return jsonResponse.getBody();
    }
}
```

```
# Include the Autoloader (see "Libraries" for install instructions)
require 'vendor/autoload.php';
use Mailgun\Mailgun;

# Instantiate the client.
$mgClient = Mailgun::create('PRIVATE_API_KEY', 'https://API_HOSTNAME');
$domain   = 'YOUR_DOMAIN_NAME';
$ip       = '127.0.0.1';

# Issue the call to the client.
$result = $mgClient->ips()->assign($domain, $ip);
```

```
def add_domain_ip():
    return requests.post(
        "https://api.mailgun.net/v3/domains/YOUR_NEW_DOMAIN_NAME/ips",
        auth=("api", "YOUR_API_KEY"),
        data={"smtp_password": "127.0.0.1"})
```

```
def add_domain_ip
  RestClient.post("https://api:YOUR_API_KEY"\
                  "@api.mailgun.net/v3/domains/YOUR_NEW_DOMAIN_NAME/ips",
                  :ip => '127.0.0.1')
end
```

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class AddDomainIPChunk
{

    public static void Main (string[] args)
    {
        Console.WriteLine (AddDomainIP ().Content.ToString ());
    }

    public static IRestResponse AddDomainIP ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3/");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.Resource = "{domain}/ips";
        request.AddParameter ("domain", "YOUR_DOMAIN_NAME", ParameterType.UrlSegment);
        request.AddParameter ("ip", "127.0.0.1");
        request.Method = Method.POST;
        return client.Execute (request);
    }

}
```

```
import (
    "context"
    "github.com/mailgun/mailgun-go/v3"
    "time"
)

func AddDomainIPS(domain, apiKey string) error {
    mg := mailgun.NewMailgun(domain, apiKey)

    ctx, cancel := context.WithTimeout(context.Background(), time.Second*30)
    defer cancel()

    return mg.AddDomainIP(ctx, "127.0.0.1")
}
```

```
var DOMAIN = 'YOUR_DOMAIN_NAME';
var mailgun = require('mailgun-js')({ apiKey: "YOUR_API_KEY", domain: DOMAIN });

mailgun.post(`/${DOMAIN}/ips`, {'ip': '127.0.0.1'}, function (error, body) {
  console.log(body);
});
```

```
{
  "message": "success"
}
```

Remove an IP from the domain pool.

```
curl -s --user 'api:YOUR_API_KEY' -X DELETE \
    https://api.mailgun.net/v3/domains/YOUR_DOMAIN_NAME/ips/127.0.0.1
```

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode deleteDomainIP() throws UnirestException {

        HttpResponse<JsonNode> request = Unirest.delete("https://api.mailgun.net/v3/
↪domains/YOUR_DOMAIN_NAME/ips/127.0.0.1")
            .basicAuth("api", API_KEY)
            .asJson();

        return request.getBody();
    }
}
```

```
# Include the Autoloader (see "Libraries" for install instructions)
require 'vendor/autoload.php';
use Mailgun\Mailgun;

# Instantiate the client.
$mgClient = Mailgun::create('PRIVATE_API_KEY', 'https://API_HOSTNAME');
$domain   = 'YOUR_DOMAIN_NAME';
$ip       = '127.0.0.1';

# Issue the call to the client.
$result = $mgClient->ips->unassign($domain, $ip);
```

```
def delete_domain_ip():
    return requests.delete(
        "https://api.mailgun.net/v3/domains/YOUR_DOMAIN_NAME/ips/127.0.0.1",
        auth=("api", "YOUR_API_KEY"))
```

```
def delete_domain_ip
  RestClient.delete "https://api:YOUR_API_KEY"\
  "@api.mailgun.net/v3/domains/YOUR_DOMAIN_NAME/ips/127.0.0.1"
end
```

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class DeleteDomainIPChunk
{

    public static void Main (string[] args)
    {
```

(continues on next page)

(continued from previous page)

```
        Console.WriteLine (DeleteDomainIP ().Content.ToString ());
    }

    public static IRestResponse DeleteDomainIP ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.AddParameter ("domain", "YOUR_DOMAIN_NAME", ParameterType.UrlSegment);
        request.Resource = "{domain}/ips/{ip}";
        request.AddUrlSegment ("ip", "127.0.0.1");
        request.Method = Method.DELETE;
        return client.Execute (request);
    }

}
```

```
import (
    "context"
    "github.com/mailgun/mailgun-go/v3"
    "time"
)

func DeleteDomainIP(domain, apiKey string) error {
    mg := mailgun.NewMailgun(domain, apiKey)

    ctx, cancel := context.WithTimeout(context.Background(), time.Second*30)
    defer cancel()

    return mg.DeleteDomainIP(ctx, "127.0.0.1")
}
```

```
var DOMAIN = 'YOUR_DOMAIN_NAME';
var mailgun = require('mailgun-js')({ apiKey: "YOUR_API_KEY", domain: DOMAIN });

mailgun.delete(`/${DOMAIN}/ips/127.0.0.1`, function (error, body) {
  console.log(body);
});
```

```
{
  "message": "success"
}
```

4.5 IP Pools

IP Pools allow you to group your dedicated IPs into customized “pools” to help manage your sending reputation for different mail sending streams. The IP Pools endpoint is available at:

```
v1/ip_pools
```

Note: You can manage IP Pools from the Control Panel. Click on **IP Pools** in the Settings dropdown menu.

4.5.1 Create IP Pool

```
POST /v1/ip_pools
```

Creates a new IP Pool and returns a unique ID

Note: Only dedicated IPs that are not on a warmup can be added to an IP Pool.

Parameter		Description
name	*string*	Name of the IP Pool being created
description	*string*	(Optional) Description of the IP Pool being created
ips	*string*	(Optional) A comma separated list of IP addresses to be assigned to this IP Pool

4.5.2 Get IP Pools

```
GET /v1/ip_pools
```

Returns a list of all IP Pools on an account

4.5.3 Update an IP Pool

```
PATCH /v1/ip_pools/{pool_id}
```

Update the name, description, or dedicated IPs assigned to an IP Pool.

Note: Only dedicated IPs that are not on a warmup can be added to an IP Pool.

Parameter		Description
name	*string*	Name of the IP Pool being created
description	*string*	Description of the IP Pool being created
add_ip	*string*	IP address that you want to add to the pool. Can be specified any number of times
remove_ip	*string*	IP address that you want to remove from the pool. Can be specified any number of times

4.5.4 Delete an IP Pool

```
DELETE /v1/ip_pools/{pool_id}
```

Deletes an IP Pool. If an IP Pool is assigned to a domain, you must provide a replacement IP option (shared, dedicated or another IP Pool).

Parameter		Description
ip	*string*	Provide a replacement dedicated IP or *shared* to use a shared IP (automatically assigned) as a replacement
pool_id	*string*	Replacement IP Pool

4.5.5 Link an IP Pool

```
POST /v3/domains/{domain_name}/ips
```

Links an IP Pool to a sendign domain. Linking an IP Pool to a domain will replace any IPs already assigned (shared or dedicated) with the IPs assigned to the pool. Only a pool with dedicated IPs can be linked to a sendign domain.

Parameter		Description
pool_id	*string*	id of the pool to assign

4.5.6 Unlink an IP Pool

```
DELETE /v3/domains/{domain_name}/ips/ip_pool
```

Removes an IP Pool from a domain. You will need to supply a replacement IP option (shared, dedicated or another IP Pool).

Parameter		Description
ip	*string*	Provide a replacement dedicated IP or *shared* to use a shared IP (automatically assigned) as a replacement
pool_id	*string*	Replacement IP Pool

4.6 Events

Mailgun tracks every event that happens to your emails and makes this data available to you through the Events API. Mailgun retains this detailed data for **one day** for free accounts and up to **30 days** for paid accounts based on subscription plan. You can query the data and traverse through the result pages as explained below.

The Events endpoint is available at:

```
v3/<domain>/events
```

```
GET /<domain>/events
```

Note: The Events API replaces the deprecated Logs API endpoint. You should use the Events API moving forward.

A request should define a time range and can specify a set of filters to apply. In response, a page of events is returned along with URLs that can be used to retrieve the next and previous result pages. To traverse the entire range, you should keep requesting the next page URLs returned along with result pages until an empty result page is reached.

Both next and previous page URLs are always returned, even when retrieving one of them makes no sense. There are two such cases: previous page URL for the first result page, and next page URL for the last result page; requesting these URLs always returns an empty result page.

4.6.1 Viewing Stored Messages

To access the contents of the stored messages, copy the API URL of the message into a browser. The API URL can be found in the expanded log entry under the "storage" section. For the username, enter "api" and provide an API key for the password in order to view the parsed message.

To view the raw MIME, the message's Mailgun storage key will be needed. 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
```

4.6.2 Time Range

The request time range should be given by a beginning timestamp and either an end timestamp or a search direction. If an end timestamp is not given, a search direction must be provided.

If the range end timestamp is provided then the relation between the beginning and the end timestamps determines the direction - ascending or descending - in which events are going to be traversed. E.g. if the end timestamp is less (older) than the beginning timestamp, then result pages are returned from newer to older and events on the pages are sorted in the descending order of their timestamps.

If the end timestamp is not provided, the direction must be specified. Depending on the range direction, the result page traversal behaves differently:

- If the range is descending then the end timestamp is determined by the user tariff plan retention period.
- If the range is ascending then events will continue to be recorded but will not show in the current request time range pages that are provided. So after the most recent events have been retrieved and an empty result page has been reached, then requesting next page URL returned with the last page some time later will return events that occurred since then. And this can go on indefinitely.

Warning: Even though it seems that real-time event polling can be implemented by traversing next URLs of an ascending time range that has no explicit end timestamp, it is not that simple! Please refer to *Event Polling* for the proper way to do it.

If both the end range date and the direction of the search are specified then they should agree with each other, otherwise the request will return an error.

4.6.3 Event Polling

In our system, events are generated by physical hosts and follow different routes to the event storage. Therefore, the order in which they appear in the storage and become retrievable - via the events API - does not always correspond to the order in which they occur. Consequently, this system behavior makes straight forward implementation of event polling miss some events. The page of most recent events returned by the events API may not contain all the events that occurred at that time because some of them could still be on their way to the storage engine. When the events arrive and are eventually indexed, they are inserted into the already retrieved pages which could result in the event being missed if the pages are accessed too early (i.e. before all events for the page are available).

To ensure that all your events are retrieved and accounted for please implement polling the following way:

1. Make a request to the events API specifying an ascending time range that begins some time in the past (e.g. half an hour ago);
2. Retrieve a result page;
3. Check the timestamp of the last event on the result page. If it is older than some threshold age (e.g. half an hour) then go to step (4), otherwise proceed with step (6);
4. The result page is trustworthy, use events from the page as you please;
5. Make a request using the next page URL retrieved with the result page, proceed with step (2);
6. Discard the result page for it is not trustworthy;
7. Pause for some time (at least 15 seconds);
8. Repeat the previous request, and proceed with step (2).

4.6.4 Query Options

URL parameters allow you to manipulate the results of your query:

Parameter	Description
begin	The beginning of the search time range. It can be specified as a string (see *Date Format*) or linux epoch seconds. Refer to *Time Range* for details.
end	The end of the search time range. It can be specified as a string (see *Date Format*) or linux epoch seconds. Refer to *Time Range* for details.
ascending	Defines the direction of the search time range and must be provided if the range end time is not specified. Can be either `yes` or `no`. Refer to *Time Range* for details.
limit	Number of entries to return. (300 max)
<field>	**<field>** is the name of the *Filter Field*. The value of the parameter should be a valid *Filter Expression*. Several field filters can be specified in one request. If the same field is mentioned, more then once, then all its filter expressions are combined with **AND** operator.

4.6.5 Filter Field

Log records can be filtered by the following fields:

Fields	Description
event	An event type. For a complete list of all events written to the log see the *Event Types* table below.
list	The email address of a mailing list the message was originally sent to.
attachment	A name of an attached file.
from	An email address mentioned in the *from* MIME header.
message-id	A Mailgun message id returned by the messages API.
subject	A subject line.
to	An email address mentioned in the *to* MIME header.
size	Message size. Mostly intended to be used with range filtering expressions (see below).
recipient	An email address of a particular recipient. While messages are addressable to one or more recipients, each event (with one exception) tracks one recipient. See *stored* events for use of *recipients*.
recipients	Specific to *stored* events, this field tracks all of the potential message recipients.
tags	User defined tags.
severity	Temporary or Permanent. Used to filter events based on severity, if exists. (Currently failed events only)

4.6.6 Filter Expression

Possible filtering expressions are listed below:

Expression	Description
foo bar	Matches field values that contain both term `foo` and term `bar`.
foo AND bar	Same as above.
foo OR bar	Matches field values that contain either term `foo` or term `bar`.
"foo bar"	Matches field values that literally contain `foo bar`.
NOT foo	Matches field values that do not contain term `foo`.
>10000	Matches values that greater then `10000`. This filter can be applied to numeric fields only.
>10000 <20000	Matches values that are greater then `10000` and less then `20000`. This filter can be applied to numeric fields only.

Note that more then one expression can be used as a filter value and parentheses can be used to specify grouping. E.g.: `(Hello AND NOT Rachel) OR (Farewell AND Monica)`.

4.6.7 Event Types

Mailgun tracks all of the events that occur throughout the system. Below are listed the events that you can retrieve using this API.

Event Type	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. severity=permanent when a message is not delivered. There are several reasons why Mailgun stops attempting to deliver messages and drops them including: hard bounces, messages that reached their retry limit, previously unsubscribed/bounced/complained addresses, or addresses rejected by an ESP. severity=temporary when a message is temporary rejected by an ESP.
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

4.6.8 Event Structure

Events are represented as loosely structured JSON documents. The exact event structure depends on the event type. But there are three fields that every event has. They are as follows:

Field	Description
event	The type of the event. Events of a particular type have an identical structure, though some fields may be optional. For the list of event types please refer to *Event Types*.
timestamp	The time when the event was generated in the system provided as Unix epoch seconds.
id	Event id. It is guaranteed to be unique within a day. It can be used to distinguish events that have already been retrieved when requests with overlapping time ranges are made.

Events can change their structure as new features are added to Mailgun, but we only add new fields and never modify or remove existing ones.

Below you can find sample events of various types:

Accepted:


```
{
  "event": "accepted",
  "id": "jxVuhYlhReaK3QsggHfFRA",
  "timestamp": 1529692198.641821,
  "log-level": "info",
  "method": "smtp",
  "envelope": {
    "targets": "team@example.org",
    "transport": "smtp",
    "sender": "sender@example.org"
  },
  "flags": {
    "is-authenticated": false
  },
  "message": {
    "headers": {
      "to": "team@example.org",
      "message-id": "20180622182958.1.48906CB188F1A454@exmple.org",
      "from": "sender@example.org",
      "subject": "Test Subject"
    },
    "attachments": [],
    "recipients": [
      "team@example.org"
    ],
    "size": 586
  },
  "storage": {
    "url": "https://se.api.mailgun.net/v3/domains/example.org/messages/eyJwI...",
    "key": "eyJwI..."
  },
  "recipient": "team@example.org",
  "recipient-domain": "example.org",
  "campaigns": [],
  "tags": [],
  "user-variables": {}
}
```


Accepted (Routed):


```
{
  "event": "accepted",
```


(continues on next page)

(continued from previous page)


```
  "id": "cWgTfzV6QQiXY4PqhlLClw",
  "timestamp": 1529692198.719447,
  "log-level": "info",
  "method": "smtp",
  "routes": [
    {
      "expression": "match_recipient(\"team@example.org\")",
      "id": "5b295a4aa4764a000108508c",
      "match": {
        "recipient": "team@example.org"
      }
    }
  ],
  "envelope": {
    "sender": "sender@example.org",
    "transport": "smtp",
    "targets": "john@example.com"
  },
  "flags": {
    "is-routed": true,
    "is-authenticated": false,
    "is-system-test": false,
    "is-test-mode": false
  },
  "message": {
    "headers": {
      "to": "team@example.org",
      "message-id": "20180622182958.1.48906CB188F1A454@exmple.org",
      "from": "sender@exmple.org",
      "subject": "Test Subject"
    },
    "attachments": [],
    "recipients": [
      "team@example.org"
    ],
    "size": 586
  },
  "storage": {
    "url": "https://se.api.mailgun.net/v3/domains/example.org/messages/eyJwI...",
    "key": "eyJwI..."
  },
  "recipient": "john@example.com",
  "recipient-domain": "example.com",
  "campaigns": [],
  "tags": [],
  "user-variables": {}
}
```


Delivered:


```
{
  "event": "delivered",
  "id": "hK7mQVt1QtqRiOfQXta4sw",
  "timestamp": 1529692199.626182,
  "log-level": "info",
  "envelope": {
    "transport": "smtp",
```


(continues on next page)

(continued from previous page)


```
    "sender": "sender@example.org",
    "sending-ip": "123.123.123.123",
    "targets": "john@example.com"
  },
  "flags": {
    "is-routed": false,
    "is-authenticated": false,
    "is-system-test": false,
    "is-test-mode": false
  },
  "delivery-status": {
    "tls": true,
    "mx-host": "aspmx.l.example.com",
    "code": 250,
    "description": "",
    "session-seconds": 0.4367079734802246,
    "utf8": true,
    "attempt-no": 1,
    "message": "OK",
    "certificate-verified": true
  },
  "message": {
    "headers": {
      "to": "team@example.org",
      "message-id": "20180622182958.1.48906CB188F1A454@exmple.org",
      "from": "sender@exmple.org",
      "subject": "Test Subject"
    },
    "attachments": [],
    "size": 586
  },
  "storage": {
    "url": "https://se.api.mailgun.net/v3/domains/example.org/messages/eyJwI...",
    "key": "eyJwI..."
  },
  "recipient": "john@example.com",
  "recipient-domain": "example.com",
  "campaigns": [],
  "tags": [],
  "user-variables": {}
}
```


Failed (Permanent):


```
{
  "event": "failed",
  "id": "pl271FzxTTmGRW8Uj3dUWw",
  "timestamp": 1529701969.818328,
  "log-level": "error",
  "severity": "permanent",
  "reason": "suppress-bounce",
  "envelope": {
    "sender": "john@example.org",
    "transport": "smtp",
    "targets": "joan@example.com"
  },
  "flags": {
```


(continues on next page)

(continued from previous page)


```
    "is-routed": false,
    "is-authenticated": true,
    "is-system-test": false,
    "is-test-mode": false
  },
  "delivery-status": {
    "attempt-no": 1,
    "message": "",
    "code": 605,
    "description": "Not delivering to previously bounced address",
    "session-seconds": 0.0
  },
  "message": {
    "headers": {
      "to": "joan@example.com",
      "message-id": "20180622211249.1.2A6098970A380E12@example.org",
      "from": "john@example.org",
      "subject": "Test Subject"
    },
    "attachments": [],
    "size": 867
  },
  "storage": {
    "url": "https://se.api.mailgun.net/v3/domains/example.org/messages/eyJwI...",
    "key": "eyJwI..."
  },
  "recipient": "slava@mailgun.com",
  "recipient-domain": "mailgun.com",
  "campaigns": [],
  "tags": [],
  "user-variables": {}
}
```


Failed (Permanent, Delayed Bounce):


```
{
  "event": "failed",
  "id": "igw2SXYxTOabezGjqA9_xw",
  "timestamp": 1529700261.747814,
  "log-level": "error",
  "severity": "permanent",
  "reason": "bounce",
  "delivery-status": {
    "message": "smtp; 550-5.1.1 The email account that you tried to reach does not␣
↪exist.",
    "code": "5.1.1",
    "description": ""
  },
  "flags": {
    "is-delayed-bounce": true,
    "is-test-mode": false
  },
  "message": {
    "headers": {
      "to": "joan@example.com",
      "message-id": "20180521184023.49972.E75804E1DC9E5F16@example.org",
      "from": "john@example.org"
```


(continues on next page)

(continued from previous page)


```
      "subject": "Test Subject"
    },
    "attachments": [],
    "size": 771
  },
  "recipient": "joan@example.com",
  "campaigns": [],
  "tags": [],
  "user-variables": {}
}
```


Failed (Temporary):


```
{
  "event": "failed",
  "id": "U2kZkAiuScqcMTq-8Atz-Q",
  "timestamp": 1529439955.794033,
  "log-level": "warn",
  "severity": "temporary",
  "reason": "generic",
  "envelope": {
    "transport": "smtp",
    "sender": "john@example.org",
    "sending-ip": "123.123.123.123",
    "targets": "joan@example.com"
  },
  "flags": {
    "is-routed": false,
    "is-authenticated": true,
    "is-system-test": false,
    "is-test-mode": false
  },
  "delivery-status": {
    "tls": true,
    "mx-host": "mx.example.com",
    "code": 421,
    "description": "",
    "session-seconds": 0.09020090103149414,
    "retry-seconds": 600,
    "attempt-no": 1,
    "message": "4.4.2 mxfront9g.mail.example.com Error: timeout exceeded",
    "certificate-verified": true
  },
  "message": {
    "headers": {
      "to": "joan@example.com",
      "message-id": "20180619202554.1.C370AA02DD7DDF22@example.org",
      "from": "john@example.org",
      "subject": "Test Subject"
    },
    "attachments": [],
    "size": 810
  },
  "storage": {
    "url": "https://se.api.mailgun.net/v3/domains/example.org/messages/eyJwI...",
    "key": "eyJwI..."
  },
```


(continues on next page)

(continued from previous page)

```
  "recipient": "joan@example.com",
  "recipient-domain": "example.com",
  "campaigns": [],
  "tags": [],
  "user-variables": {}
}
```

Opened:

```
{
  "event": "opened",
  "id": "-laxIqj9QWubsjY_3pTq_g",
  "timestamp": 1377047343.042277,
  "log-level": "info",
  "recipient": "recipient@example.com",
  "geolocation": {
    "country": "US",
    "region": "Texas",
    "city": "Austin"
  },
  "tags": [],
  "campaigns": [],
  "user-variables": {},
  "ip": "111.111.111.111",
  "client-info": {
    "client-type": "mobile browser",
    "client-os": "iOS",
    "device-type": "mobile",
    "client-name": "Mobile Safari",
    "user-agent": "Mozilla/5.0 (iPhone; CPU iPhone OS 6_1 like Mac OS X) AppleWebKit/
↪536.26 (KHTML, like Gecko) Mobile/10B143"
  },
  "message": {
    "headers": {
      "message-id": "20130821005614.19826.35976@samples.mailgun.org"
    }
  },
}
```

Clicked:

```
{
  "event": "clicked",
  "id": "G5zMz2ysS6OxZ2C8xb2Tqg",
  "timestamp": 1377075564.094891,
  "log-level": "info",
  "recipient": "recipient@example.com",
  "geolocation": {
    "country": "US",
    "region": "TX",
    "city": "Austin"
  },
  "tags": [],
  "url": "http://example.com/signup",
  "ip": "123.123.123.321",
  "campaigns": [],
  "user-variables": {},
```

(continues on next page)

(continued from previous page)

```
  "client-info": {
    "client-type": "browser",
    "client-os": "Linux",
    "device-type": "desktop",
    "client-name": "Chromium",
    "user-agent": "Mozilla/5.0 (X11; Linux i686) AppleWebKit/537.36 (KHTML, like␣
↪Gecko) Ubuntu Chromium/28.0.1500.71 Chrome/28.0.1500.71 Safari/537.36"
  },
  "message": {
    "headers": {
      "message-id": "20130821085807.30688.67706@samples.mailgun.org"
    }
  },
}
```

Unsubscribed:

```
{
  "event": "unsubscribed",
  "id": "W3X4JOhFT-OZidZGKKr9iA",
  "timestamp": 1377213791.421473,
  "log-level": "info",
  "recipient": "recipient@example.com",
  "geolocation": {
    "country": "US",
    "region": "TX",
    "city": "San Antonio"
  },
  "campaigns": [],
  "tags": [],
  "user-variables": {},
  "ip": "23.23.23.345",
  "client-info": {
    "client-type": "browser",
    "client-os": "OS X",
    "device-type": "desktop",
    "client-name": "Chrome",
    "user-agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_8_4) AppleWebKit/537.36␣
↪(KHTML, like Gecko) Chrome/28.0.1500.95 Safari/537.36"
  },
  "message": {
    "headers": {
      "message-id": "20130822232216.13966.79700@samples.mailgun.org"
    }
  },
}
```

Complained:

```
{
  "event": "complained",
  "id": "ncV2XwymRUKbPek_MIM-Gw",
  "timestamp": 1377214260.049634,
  "log-level": "warn",
  "recipient": "foo@example.com",
  "tags": [],
  "campaigns": [],
```

(continues on next page)

(continued from previous page)

```
  "user-variables": {},
  "flags": {
    "is-test-mode": false
  },
  "message": {
    "headers": {
      "to": "foo@example.com",
      "message-id": "20130718032413.263EE2E0926@example.com",
      "from": "John Doe <sender@example.com>",
      "subject": "This is the subject."
    },
    "attachments": [],
    "size": 18937
  },
}
```

Stored:

```
{
  "event": "stored",
  "id": "WRVmVc47QYi4DHth_xpRUA",
  "timestamp": 1529692198.691758,
  "log-level": "info",
  "flags": {
    "is-test-mode": false
  },
  "message": {
    "headers": {
      "to": "team@example.org",
      "message-id": "20180622182958.1.48906CB188F1A454@exmple.org",
      "from": "sender@example.org",
      "subject": "Test Subject"
    },
    "attachments": [],
    "recipients": [
      "team@example.org"
    ],
    "size": 586
  },
  "storage": {
    "url": "https://se.api.mailgun.net/v3/domains/example.org/messages/eyJwI...",
    "key": "eyJwI..."
  },
  "campaigns": [],
  "tags": [],
  "user-variables": {}
}
```

Rejected:

```
{
  "event": "rejected"
  "id": "OMTXD3-sSmKIQa1gSKkYVA",
  "timestamp": 1529704976.104692,
  "log-level": "warn",
  "flags": {
    "is-test-mode": false
```

(continues on next page)

(continued from previous page)

```
  },
  "reject": {
    "reason": "Sandbox subdomains are for test purposes only. Please add your own␣
↪domain or add the address to authorized recipients in Account Settings.",
    "description": ""
  },
  "message": {
    "headers": {
      "to": "joan@example.org",
      "message-id": "20180622220256.1.
↪B31A451A2E5422BB@sandbox55887fac92de874df5ae0023b75fd62f1d.mailgun.org",
      "from": "john@sandbox55887fac92de874df5ae0023b75fd62f1d.mailgun.org",
      "subject": "Test Subject"
    },
    "attachments": [],
    "size": 867
  },
  "campaigns": [],
  "tags": [],
  "user-variables": {}
}
```

List Member Uploaded:

List Member Upload Error

List Uploaded:

4.6.9 Examples

Fetches the first page of log records that are older than `Fri, 3 May 2013 09:00:00 -0000` and correspond to delivery to `joe@example.com`:

```
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
```

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode getLogs() throws UnirestException {

        HttpResponse<JsonNode> request = Unirest.get("https://api.mailgun.net/v3/" +␣
↪YOUR_DOMAIN_NAME + "/events")
            .basicAuth("api", API_KEY)
```

(continues on next page)

(continued from previous page)

```
            .queryString("begin", "Thurs, 18 May 2017 09:00:00 -0000")
            .queryString("ascending", "yes")
            .queryString("limit", 1)
            .asJson();

        return request.getBody();
    }
}
```

```
# Include the Autoloader (see "Libraries" for install instructions)
require 'vendor/autoload.php';
use Mailgun\Mailgun;

# Instantiate the client.
$mgClient    = Mailgun::create('PRIVATE_API_KEY', 'https://API_HOSTNAME');
$domain      = 'YOUR_DOMAIN_NAME';
$queryString = array(
    'begin'        => 'Wed, 1 Jan 2020 09:00:00 -0000',
    'ascending'    => 'yes',
    'limit'        =>  25,
    'pretty'       => 'yes',
    'recipient'    => 'bob@example.com'
);

# Issue the call to the client.
$result = $mgClient->events()->get($domain, $queryString);
```

```
def get_logs():
    return requests.get(
        "https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/events",
        auth=("api", "YOUR_API_KEY"),
        params={"begin"        : "Fri, 3 May 2013 09:00:00 -0000",
                "ascending"    : "yes",
                "limit"        :  25,
                "pretty"       : "yes",
                "recipient"    : "joe@example.com"})
```

```
def get_logs
  RestClient.get "https://api:YOUR_API_KEY"\
  "@api.mailgun.net/v3/YOUR_DOMAIN_NAME/events",
   :params => {
    :'begin'        => 'Fri, 3 May 2013 09:00:00 -0000',
    :'ascending'    => 'yes',
    :'limit'        =>  25,
    :'pretty'       => 'yes',
    :'recipient' => 'joe@example.com'
  }
end
```

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class EventsDateTimeRecipientChunk
```

(continues on next page)

(continued from previous page)

```
{

    public static void Main (string[] args)
    {
        Console.WriteLine (EventsDateTimeRecipient ().Content.ToString ());
    }

    public static IRestResponse EventsDateTimeRecipient ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.AddParameter ("domain", "YOUR_DOMAIN_NAME", ParameterType.UrlSegment);
        request.Resource = "{domain}/events";
        request.AddParameter ("begin", "Fri, 3 May 2013 09:00:00 -0000");
        request.AddParameter ("ascending", "yes");
        request.AddParameter ("limit", 25);
        request.AddParameter ("pretty", "yes");
        request.AddParameter ("recipient", "joe@example.com");
        return client.Execute (request);
    }

}
```

```
import (
    "context"
    "github.com/mailgun/mailgun-go/v3"
    "time"
)

func PrintEventLog(domain, apiKey string) error {
    mg := mailgun.NewMailgun(domain, apiKey)

    // Create an iterator
    it := mg.ListEvents(&mailgun.ListEventOptions{
        Begin: time.Now().Add(-50 * time.Minute),
        Limit: 100,
        Filter: map[string]string{
            "recipient": "joe@example.com",
        },
    })

    ctx, cancel := context.WithTimeout(context.Background(), time.Second*30)
    defer cancel()

    // Iterate through all the pages of events
    var page []mailgun.Event
    for it.Next(ctx, &page) {
        for _, event := range page {
            fmt.Printf("%+v\n", event)
        }
    }
```

(continues on next page)

(continued from previous page)

```
    // Did iteration end because of an error?
    if it.Err() != nil {
        return it.Err()
    }

    return nil
}
```

```
var DOMAIN = 'YOUR_DOMAIN_NAME';
var mailgun = require('mailgun-js')({ apiKey: "YOUR_API_KEY", domain: DOMAIN });

mailgun.get(`/${DOMAIN}/events`, {"begin": "Thurs, 06 July 2017 09:00:00 -0000",
↪"ascending": "yes", "limit": 1},  function (error, body) {
  console.log(body);
});
```

Sample response:

```
{
  "items": [
    {
      "tags": [],
      "id": "czsjqFATSlC3QtAK-C80nw",
      "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": "foo@example.com",
          "message-id": "20130812164300.28108.52546@samples.mailgun.org",
          "from": "Excited User <me@samples.mailgun.org>",
          "subject": "Hello"
        },
        "attachments": [],
        "recipients": [
          "foo@example.com",
          "baz@example.com",
          "bar@example.com"
        ],
        "size": 69
      },
      "recipient": "baz@example.com",
      "method": "http"
    }
  ],
  "paging": {
    "next":
```

(continues on next page)

(continued from previous page)

```
        "https://api.mailgun.net/v3/samples.mailgun.org/events/W3siY...",
    "previous":
        "https://api.mailgun.net/v3/samples.mailgun.org/events/Lkawm..."
  }
}
```

Fetches the first page of log records that contain different types of failure, starting from the most recent:

```
curl -s --user 'api:YOUR_API_KEY' -G \
    https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/events \
    --data-urlencode event='rejected OR failed'
```

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode getLogs() throws UnirestException {

        HttpResponse<JsonNode> request = Unirest.get("https://api.mailgun.net/v3/" +␣
↪YOUR_DOMAIN_NAME + "/events")
            .basicAuth("api", API_KEY)
            .queryString("event", "failed")
            .asJson();

        return request.getBody();
    }
}
```

```
# Include the Autoloader (see "Libraries" for install instructions)
require 'vendor/autoload.php';
use Mailgun\Mailgun;

# Instantiate the client.
$mgClient = Mailgun::create('PRIVATE_API_KEY', 'https://API_HOSTNAME');
$domain      = 'YOUR_DOMAIN_NAME';
$queryString = array(
    'begin'        => 'Wed, 1 Jan 2020 09:00:00 -0000',
    'ascending'    => 'yes',
    'limit'        =>  25,
    'pretty'       => 'yes',
    'event'        => 'failed'
);

# Issue the call to the client.
$result = $mgClient->events()->get($domain, $queryString);
```

```
def get_logs():
    return requests.get(
        "https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/events",
        auth=("api", "YOUR_API_KEY"),
        params={"event" : "rejected OR failed"})
```

```ruby
def get_logs
  RestClient.get "https://api:YOUR_API_KEY"\
  "@api.mailgun.net/v3/YOUR_DOMAIN_NAME/events",
  :params => {
    :"event" => 'rejected OR failed'
  }
end
```

```csharp
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class EventsFailureChunk
{

    public static void Main (string[] args)
    {
        Console.WriteLine (EventsFailure ().Content.ToString ());
    }

    public static IRestResponse EventsFailure ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.AddParameter ("domain", "YOUR_DOMAIN_NAME", ParameterType.UrlSegment);
        request.Resource = "{domain}/events";
        request.AddParameter ("event", "rejected OR failed");
        return client.Execute (request);
    }

}
```

```go
import (
    "context"
    "fmt"
    "github.com/mailgun/mailgun-go/v3"
    "github.com/mailgun/mailgun-go/v3/events"
    "time"
)

func PrintFailedEvents(domain, apiKey string) error {
    mg := mailgun.NewMailgun(domain, apiKey)

    // Create an iterator
    it := mg.ListEvents(&mailgun.ListEventOptions{
        Filter: map[string]string{
            "event": "rejected OR failed",
        },
    })

    ctx, cancel := context.WithTimeout(context.Background(), time.Second*30)
    defer cancel()
```

(continues on next page)

(continued from previous page)

```
    // Iterate through all the pages of events
    var page []mailgun.Event
    for it.Next(ctx, &page) {
        for _, event := range page {
            switch e := event.(type){
            case *events.Failed:
                fmt.Printf("Failed Reason: %s", e.Reason)
            case *events.Rejected:
                fmt.Printf("Rejected Reason: %s", e.Reject.Reason)
            }
        }
    }

    // Did iteration end because of an error?
    if it.Err() != nil {
        return it.Err()
    }
    return nil
}
```

```
var DOMAIN = 'YOUR_DOMAIN_NAME';
var mailgun = require('mailgun-js')({ apiKey: "YOUR_API_KEY", domain: DOMAIN });

mailgun.get(`/${DOMAIN}/events`, {"event": "failed"},  function (error, body) {
  console.log(body);
});
```

```
{
  "items": [
    {
      "severity": "temporary",
      "tags": [],
      "id": "czsjqFATSlC3QtAK-C80nw",
      "envelope": {
        "sender": "me@samples.mailgun.org",
        "transport": ""
      },
      "delivery-status": {
        "code": 498,
        "message": "No MX for [example.com]",
        "retry-seconds": 900,
        "description": "No MX for [example.com]"
      },
      "campaigns": [],
      "reason": "generic",
      "user-variables": {},
      "flags": {
        "is-authenticated": true,
        "is-test-mode": false
      },
      "timestamp": 1376435471.10744,
      "message": {
        "headers": {
          "to": "baz@example.com, bar@example.com",
          "message-id": "20130813230036.10303.40433@samples.mailgun.org",
```

(continues on next page)

(continued from previous page)

```
        "from": "Excited User <me@samples.mailgun.org>",
        "subject": "Hello"
      },
      "attachments": [],
      "recipients": [
        "baz@example.com",
        "bar@example.com"
      ],
      "size": 370
    },
    "recipient": "bar@example.com",
    "event": "failed"
  }
 ],
 "paging": {
   "next":
      "https://api.mailgun.net/v3/samples.mailgun.org/events/W3siY...",
   "previous":
      "https://api.mailgun.net/v3/samples.mailgun.org/events/Lkawm..."
 }
}
```

Fetches the first page of log records written so far. Traversal is performed starting from the most recent records to the oldest:

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

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode getLogs() throws UnirestException {

        HttpResponse<JsonNode> request = Unirest.get("https://api.mailgun.net/v3/" +␣
↪YOUR_DOMAIN_NAME + "/events")
            .basicAuth("api", API_KEY)
            .asJson();

        return request.getBody();
    }
}
```

```
# Include the Autoloader (see "Libraries" for install instructions)
require 'vendor/autoload.php';
use Mailgun\Mailgun;

# Instantiate the client.
$mgClient = Mailgun::create('PRIVATE_API_KEY', 'https://API_HOSTNAME');
$domain   = 'YOUR_DOMAIN_NAME';
```

(continues on next page)

(continued from previous page)

```
# Issue the call to the client.
$result = $mgClient->events()->($domain);
```

```
def get_logs():
    return requests.get(
        "https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/events",
        auth=("api", "YOUR_API_KEY"))
```

```
def get_logs
  RestClient.get "https://api:YOUR_API_KEY"\
  "@api.mailgun.net/v3/YOUR_DOMAIN_NAME/events"
end
```

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class EventsTraversalChunk
{

    public static void Main (string[] args)
    {
        Console.WriteLine (EventsTraversal ().Content.ToString ());
    }

    public static IRestResponse EventsTraversal ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.AddParameter ("domain", "YOUR_DOMAIN_NAME", ParameterType.UrlSegment);
        request.Resource = "{domain}/events";
        return client.Execute (request);
    }

}
```

```
import (
    "context"
    "fmt"
    "github.com/mailgun/mailgun-go/v3"
    "github.com/mailgun/mailgun-go/v3/events"
    "time"
)

func PrintEvents(domain, apiKey string) error {
    mg := mailgun.NewMailgun(domain, apiKey)

    // Create an iterator
    it := mg.ListEvents(nil)
```

(continues on next page)

(continued from previous page)

```
    ctx, cancel := context.WithTimeout(context.Background(), time.Second*30)
    defer cancel()

    // Iterate through all the pages of events
    var page []mailgun.Event
    for it.Next(ctx, &page) {
        for _, event := range page {
            switch e := event.(type) {
            case *events.Accepted:
                fmt.Printf("Accepted ID: %s", e.Message.Headers.MessageID)
            case *events.Rejected:
                fmt.Printf("Rejected Reason: %s", e.Reject.Reason)
            // Add other event types here
            }
            fmt.Printf("%+v\n", event.GetTimestamp())
        }
    }

    // Did iteration end because of an error?
    if it.Err() != nil {
        return it.Err()
    }
    return nil
}
```

```
var DOMAIN = 'YOUR_DOMAIN_NAME';
var mailgun = require('mailgun-js')({ apiKey: "YOUR_API_KEY", domain: DOMAIN });

mailgun.get(`/${DOMAIN}/events`, function (error, body) {
  console.log(body);
});
```

Fetches the next page of log records, assuming that the URL was returned by the previous request:

```
curl -s --user 'api:YOUR_API_KEY' -G \
    https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/events/W3siYSI6IGZhbHNlLC
```

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode getLogsPagination() throws UnirestException {

        HttpResponse<JsonNode> request = Unirest.get("https://api.mailgun.net/v3/" +␣
↪YOUR_DOMAIN_NAME + "/events/W3siYSI6IGZhbHNlLCAi")
            .basicAuth("api", API_KEY)
            .asJson();

        return request.getBody();
    }
}
```

```php
# Include the Autoloader (see "Libraries" for install instructions)
require 'vendor/autoload.php';
use Mailgun\Mailgun;

# Instantiate the client.
$mgClient    = Mailgun::create('PRIVATE_API_KEY', 'https://API_HOSTNAME');
$domain      = 'YOUR_DOMAIN_NAME';
$queryString = array(
    'begin'        => 'Wed, 1 Jan 2020 09:00:00 -0000',
    'ascending'    => 'yes',
    'limit'        =>  25,
    'pretty'       => 'yes'
);

# Issue the call to the client.
$result = $mgClient->events()->get($domain, $queryString);

# Request the next page.
$nextPage = $mgClient->events()->nextPage($result);
```

```python
def get_logs():
    return requests.get(
        "https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/events/W3siYSI6IGZhbHNlLC",
        auth=("api", "YOUR_API_KEY"))
```

```ruby
def get_logs
  RestClient.get "https://api:YOUR_API_KEY"\
  "@api.mailgun.net/v3/YOUR_DOMAIN_NAME/events/W3siYSI6IGZhbHNlLC"
end
```

```c#
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class EventsPaginationChunk
{

    public static void Main (string[] args)
    {
        Console.WriteLine (EventsPagination ().Content.ToString ());
    }

    public static IRestResponse EventsPagination ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.AddParameter ("domain", "YOUR_DOMAIN_NAME", ParameterType.UrlSegment);
        request.Resource = "{domain}/events/W3siYSI6IGZhbHNlLC";
        return client.Execute (request);
    }

}
```

```
import (
    "context"
    "fmt"
    "github.com/mailgun/mailgun-go/v3"
    "github.com/mailgun/mailgun-go/v3/events"
    "time"
)

func PrintEvents(domain, apiKey string) error {
    mg := mailgun.NewMailgun(domain, apiKey)

    // Create an iterator
    it := mg.ListEvents(nil)

    ctx, cancel := context.WithTimeout(context.Background(), time.Second*30)
    defer cancel()

    // Iterate through all the pages of events
    var page []mailgun.Event
    for it.Next(ctx, &page) {
        for _, event := range page {
            switch e := event.(type) {
            case *events.Accepted:
                fmt.Printf("Accepted ID: %s", e.Message.Headers.MessageID)
            case *events.Rejected:
                fmt.Printf("Rejected Reason: %s", e.Reject.Reason)
            // Add other event types here
            }
            fmt.Printf("%+v\n", event.GetTimestamp())
        }
    }

    // Did iteration end because of an error?
    if it.Err() != nil {
        return it.Err()
    }
    return nil
}
```

```
var DOMAIN = 'YOUR_DOMAIN_NAME';
var mailgun = require('mailgun-js')({ apiKey: "YOUR_API_KEY", domain: DOMAIN });

mailgun.get(`/${DOMAIN}/events/W3siYSI6IGZhbHNlLCAi`, function (error, body) {
  console.log(body);
});
```

4.7 Stats

Mailgun collects many different events and generates event statistics which are available in your Control Panel. This data is also available via our stats API endpoint. The stats endpoint is available at:

```
v3/<domain>/stats
```

The statistics are calculated in hourly, daily and monthly resolution in UTC timezone.

The following retention policy is applied to the statistics:

- Hourly stats are preserved for a month.
- Daily stats are preserved for a year.
- Monthly stats are stored throughout the lifespan of the domain.

```
GET /<domain>/stats/total
```

Returns total stats for a given domain.

Parameter	Description
event	The type of the event. For a complete list of all events written to the log see the *Event Types* table below. (Required)
start	The starting time. Should be in **RFC 2822#page-14** or unix epoch format. Default: 7 days from the current time.
end	The ending date. Should be in **RFC 2822#page-14** or unix epoch format. Default: current time.
resolution	Can be either `hour`, `day` or `month`. Default: `day`
duration	Period of time with resoluton encoded. See *Duration* for more info. If provided, overwrites the start date.

4.7.1 Duration

Duration is a string that represents a period of time with some resolution. It has a format `[0-9]+[m,d,h]` where

- `h` - an hour
- `d` - a day
- `m` - a month

Examples:

- *24h* - a period of 24 hours (a day) with hourly resolution
- *1d* - a period of 1 day with daily resolution
- *2m* - a period of 2 months with monthly resolution

4.7.2 Event Types

Mailgun tracks all of the events that occur throughout the system. Below are listed the events that you can retrieve using this API.

Event Type	Description
accepted	Mailgun accepted the request to send/forward the email and the message has been placed in queue.
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

4.7.3 Examples

Get stats for 'accepted', 'delivered', and 'failed' events for the past month:

```
curl -s --user 'api:YOUR_API_KEY' -G \
   https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/stats/total \
   -d event='accepted' \
   -d event='delivered' \
   -d event='failed' \
   -d duration='1m'
```

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode getStats() throws UnirestException {

        HttpResponse<JsonNode> request = Unirest.get("https://api.mailgun.net/v3/" +
↪YOUR_DOMAIN_NAME + "/stats/total")
            .basicAuth("api", API_KEY)
            .queryString("event", "accepted")
            .queryString("event", "delivered")
            .queryString("event", "failed")
            .queryString("duration","1m")
            .asJson();

        return request.getBody();
    }
}
```

```
# Include the Autoloader (see "Libraries" for install instructions)
require 'vendor/autoload.php';
use Mailgun\Mailgun;

# Instantiate the client.
$mgClient = Mailgun::create('PRIVATE_API_KEY', 'https://API_HOSTNAME');
$domain   = 'YOUR_DOMAIN_NAME';

# Define your Event types
$params = array(
  "event" => "accepted",
  "event" => "delivered",
  "event" => "failed",
  "event" => "complained"
);
```

$response = $mgClient->stats()->total($domain, $params);

```
def get_stats():
    return requests.get(
        "https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/stats/total",
        auth=("api", "YOUR_API_KEY"),
        params={"event": ["accepted", "delivered", "failed"],
                "duration": "1m"})
```

```
def get_stats
  url_params = {}
  url_params[:duration] = "1m"
  url_params[:event] = []
  url_params[:event] << "accepted"
  url_params[:event] << "delivered"
  url_params[:event] << "failed"
  query_string = url_params.collect {|k, v| "#{k.to_s}=#{CGI::escape(v.to_s)}"}.
    join("&")
  RestClient.get "https://api:YOUR_API_KEY"\
  "@api.mailgun.net/v3/YOUR_DOMAIN_NAME/stats/total?#{query_string}"
end
```

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class GetStatsChunk
{

    public static void Main (string[] args)
    {
        Console.WriteLine (GetStats ().Content.ToString ());
    }

    public static IRestResponse GetStats ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
```

(continues on next page)

(continued from previous page)

```
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.AddParameter ("domain", "YOUR_DOMAIN_NAME", ParameterType.UrlSegment);
        request.Resource = "{domain}/stats/total";
        request.AddParameter ("event", "accepted");
        request.AddParameter ("event", "delivered");
        request.AddParameter ("event", "failed");
        request.AddParameter ("duration", "1m");
        return client.Execute (request);
    }

}
```

```
import (
    "context"
    "github.com/mailgun/mailgun-go/v3"
    "time"
)

func GetStats(domain, apiKey string) ([]mailgun.Stats, error) {
    mg := mailgun.NewMailgun(domain, apiKey)

    ctx, cancel := context.WithTimeout(context.Background(), time.Second*30)
    defer cancel()

    return mg.GetStats(ctx, []string{"accepted", "delivered", "failed"}, &mailgun.
↪GetStatOptions{
        Duration: "1m",
    })
}
```

```
var DOMAIN = 'YOUR_DOMAIN_NAME';
var mailgun = require('mailgun-js')({ apiKey: "YOUR_API_KEY", domain: DOMAIN });

mailgun.get(`/${DOMAIN}/stats/total`, {"event": ['accepted', 'delivered', 'failed'],
↪"duration": '1m'}, function (error, body) {
  console.log(body);
});
```

Sample response:

```
{
  "end": "Fri, 01 Apr 2012 00:00:00 UTC",
  "resolution": "month",
  "start": "Tue, 14 Feb 2012 00:00:00 UTC",
  "stats": [
    {
      "time": "Tue, 14 Feb 2012 00:00:00 UTC",
      "accepted": {
        "outgoing": 10,   // authenticated
        "incoming": 5,    // unauthenticated
        "total": 15
      },
      "delivered": {
          "smtp": 15,   // delivered over SMTP
          "http": 5,    // delivered over HTTP
```

(continues on next page)

(continued from previous page)

```
        "total": 20
      },
      "failed": {
        "permanent": {
          "bounce": 4,
          "delayed-bounce": 1,
          "suppress-bounce": 1,         // recipients previously bounced
          "suppress-unsubscribe": 2,    // recipients previously unsubscribed
          "suppress-complaint": 3,      // recipients previously complained
          "total": 10                   // failed permanently and dropped
        },
        "temporary": {
          "espblock": 1    // failed temporary due to ESP block, will be retried
        }
      },
    }
  ]
}
```

4.8 Tags

Mailgun lets you tag each outgoing message with a custom value and provides statistics on the tag level. To tag a message you need to provide one or more `o:tag` parameter in the API. Tags are created on the fly but they are subject to a limit.

The tags API endpoint is available at:

```
v3/<domain>/tags
```

Note: Unicode characters are not allowed in tags. Any unicode characters found in tags are stripped from the tag. If tag is entirely unicode characters, the tag is ignored.

```
GET /<domain>/tags
```

Returns a list of tags for a domain. Provides pagination urls if the result set is too long to be returned in a single response.

Parameter	Description
domain	Name of the domain
limit	Number of entries to return. Default: 100.

```
GET /<domain>/tags/<tag>
```

Returns a given tag.

Parameter	Description
domain	Name of the domain
tag	Name of the tag

```
PUT /<domain>/tags/<tag>
```

Updates a given tag with the information provided.

Parameter	Description
domain	Name of the domain
tag	Name of the tag
description	Optional description of a tag.

```
GET /<domain>/tags/<tag>/stats
```

Returns statistics for a given tag.

Parameter	Description
event	The type of the event. For a complete list of all events written to the log see the *Event Types* table below. (Required)
start	The starting time. Should be in **RFC 2822#page-14** or unix epoch format. Default: 7 days from the current time.
end	The ending date. Should be in **RFC 2822#page-14** or unix epoch time in seconds. Default: current time.
resolution	Can be either `hour`, `day` or `month`. Default: `day`
duration	Period of time with resolution encoded. See *Duration* for more info. If provided, overwrites the start date and resolution.

```
DELETE /<domain>/tags/<tag>
```

Deletes the tag. **Note:** The statistics for the tag are not destroyed.

Parameter	Description
domain	Name of the domain

```
GET /<domain>/tags/<tag>/stats/aggregates/countries
```

Returns a list of countries of origin for a given domain for different event types.

```
GET /<domain>/tags/<tag>/stats/aggregates/providers
```

Returns a list of email providers for a given domain for different event types.

```
GET /<domain>/tags/<tag>/stats/aggregates/devices
```

Returns a list of devices for a given domain that have triggered event types.

4.8.1 Duration

Duration is a string that represents a period of time with some resolution. It has a format `[0-9]+[m,d,h]` where

- `h` - an hour
- `d` - a day

- `m` - a month

Examples:

- `24h` - a period of 24 hours (a day) with hourly resolution
- `1d` - a period of 1 day with daily resolution
- `2m` - a period of 2 months with monthly resolution

4.8.2 Event Types

Mailgun tracks all of the events that occur throughout the system. Below are listed the events that you can retrieve using this API.

Event Type	Description
accepted	Mailgun accepted the request to send/forward the email and the message has been placed in queue.
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

4.8.3 Examples

Retrieve all tags for a domain:

```
curl -s --user 'api:YOUR_API_KEY' -G \
   https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/tags \
   -d limit=10
```

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode getTags() throws UnirestException {

        HttpResponse <JsonNode> request = Unirest.get("https://api.mailgun.net/v3/"+␣
↪YOUR_DOMAIN_NAME + "/tags")
            .basicAuth("api", API_KEY)
```

(continues on next page)

(continued from previous page)

```
            .queryString("limit", 10)
            .asJson();

        return request.getBody();
    }
}
```

```
# Include the Autoloader (see "Libraries" for install instructions)
require 'vendor/autoload.php';
use Mailgun\Mailgun;

# Instantiate the client.
$mgClient = Mailgun::create('PRIVATE_API_KEY', 'https://API_HOSTNAME');
$domain = 'YOUR_DOMAIN_NAME';

# Issue the call to the client.
$result = $mgClient->tags()->index($domain);
```

```
def get_stats():
    return requests.get(
        "https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/tags",
        auth=("api", "YOUR_API_KEY"),
        params={"limit": 10})
```

```
def get_stats
  url_params = {}
  url_params[:limit] = 10
  query_string = url_params.collect {|k, v| "#{k.to_s}=#{CGI::escape(v.to_s)}"}.
    join("&")
  RestClient.get "https://api:YOUR_API_KEY"\
  "@api.mailgun.net/v3/YOUR_DOMAIN_NAME/tags?#{query_string}"
end
```

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class GetTagsChunk
{

    public static void Main (string[] args)
    {
        Console.WriteLine (GetTags ().Content.ToString ());
    }

    public static IRestResponse GetTags ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.AddParameter ("domain", "YOUR_DOMAIN_NAME", ParameterType.UrlSegment);
```

(continues on next page)

(continued from previous page)

```
        request.Resource = "{domain}/tags";
        request.AddParameter ("limit", 10);
        return client.Execute (request);
    }

}
```

```
import (
    "context"
    "github.com/mailgun/mailgun-go/v3"
    "time"
)

func ListTags(domain, apiKey string) ([]mailgun.Tag, error) {
    mg := mailgun.NewMailgun(domain, apiKey)
    it := mg.ListTags(nil)

    ctx, cancel := context.WithTimeout(context.Background(), time.Second*30)
    defer cancel()

    var page, result []mailgun.Tag
    for it.Next(ctx, &page) {
        result = append(result, page...)
    }

    if it.Err() != nil {
        return nil, it.Err()
    }
    return result, nil
}
```

```
var DOMAIN = 'YOUR_DOMAIN_NAME';
var mailgun = require('mailgun-js')({ apiKey: "YOUR_API_KEY", domain: DOMAIN });

mailgun.get(`/${DOMAIN}/tags`, {"limit": 10}, function (error, body) {
  console.log(body);
});
```

Sample response:

```
{
  "items": [
      {
        "tag": "red",
        "description": "red signup button",
      },
      {
        "tag": "green",
        "description": "green signup button",
      },
  ],
  "paging": {
    "next":
        "https://url_to_next_page",
    "previous":
        "https://url_to_previous_page",
```

(continues on next page)

(continued from previous page)

```
    "first":
        "https://url_to_first_page",
    "last":
        "https://url_to_last_page"
  }
}
```

Delete tag:

```
curl -s --user 'api:YOUR_API_KEY' -X DELETE \
    https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/tags/newsletter
```

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode deleteTag() throws UnirestException {

        HttpResponse <JsonNode> request = Unirest.delete("https://api.mailgun.net/v3/
↪"+ YOUR_DOMAIN_NAME + "/tags/newsletter")
            .basicAuth("api", API_KEY)
            .asJson();

        return request.getBody();
    }
}
```

```
# Include the Autoloader (see "Libraries" for install instructions)
require 'vendor/autoload.php';
use Mailgun\Mailgun;

# Instantiate the client.
$mgClient = Mailgun::create('PRIVATE_API_KEY', 'https://API_HOSTNAME');
$domain = 'YOUR_DOMAIN_NAME';
$tag    = 'my_tag';

# Issue the call to the client.
$result = $mgClient->tags()->delete($domain, $tag);
```

```
def delete_tag():
    return requests.delete(
        "https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/tags/newsletter",
        auth=("api", "YOUR_API_KEY"))
```

```
def delete_tag
  RestClient.delete "https://api:YOUR_API_KEY"\
  "@api.mailgun.net/v3/YOUR_DOMAIN_NAME/tag/newsletter"
end
```

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class DeleteTagChunk
{

    public static void Main (string[] args)
    {
        Console.WriteLine (DeleteTag ().Content.ToString ());
    }

    public static IRestResponse DeleteTag ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.AddParameter ("domain", "YOUR_DOMAIN_NAME", ParameterType.UrlSegment);
        request.Resource = "{domain}/tags/{tag}";
        request.AddUrlSegment ("tag", "newsletter");
        request.Method = Method.DELETE;
        return client.Execute (request);
    }

}
```

```
import (
    "context"
    "github.com/mailgun/mailgun-go/v3"
    "time"
)

func DeleteTag(domain, apiKey string) error {
    mg := mailgun.NewMailgun(domain, apiKey)

    ctx, cancel := context.WithTimeout(context.Background(), time.Second*30)
    defer cancel()

    return mg.DeleteTag(ctx, "newsletter")
}
```

```
var DOMAIN = 'YOUR_DOMAIN_NAME';
var mailgun = require('mailgun-js')({ apiKey: "YOUR_API_KEY", domain: DOMAIN });

mailgun.delete(`/${DOMAIN}/tags/newsletter`, function (error, body) {
  console.log(body);
});
```

Sample response:

```
{
  "message": "Tag deleted"
}
```

Retrieve all countries of origin for a domain with event types:

```
curl -v -s --user 'api:YOUR_API_KEY' \
https://api.mailgun.net/v3/<domain>/tags/<tag>/stats/aggregates/countries
```


```
{
  "countries": {
      "ad": {
          "clicked": 7,
          "complained": 4,
          "opened": 18,
          "unique_clicked": 0,
          "unique_opened": 2,
          "unsubscribed": 0
      },
      "ck": {
          "clicked": 13,
          "complained": 2,
          "opened": 1,
          "unique_clicked": 1,
          "unique_opened": 0,
          "unsubscribed": 2
      }
  },
  "tag": "exampletag"
}
```


Retrieve all providers of a domain with corresponding event types:

```
curl -v -s --user 'api:YOUR_API_KEY' \
https://api.mailgun.net/v3/<domain>/tags/<tag>/stats/aggregates/providers
```


```
{
  "providers": {
      "gmail.com": {
          "accepted": 23,
          "clicked": 15,
          "complained": 0,
          "delivered": 23,
          "opened": 19,
          "unique_clicked": 2,
          "unique_opened": 7,
          "unsubscribed": 1
      },
      "yahoo.com": {
          "accepted": 16,
          "clicked": 8,
          "complained": 2,
          "delivered": 8,
          "opened": 4,
          "unique_clicked": 0,
          "unique_opened": 0,
          "unsubscribed": 0
      }
  },
  "tag": "exampletag"
}
```

Retrieve all devices that have triggered an event type in a domain:

```
curl -v -s --user 'api:YOUR_API_KEY' \
https://api.mailgun.net/v3/<domain>/tags/<tag>/stats/aggregates/devices
```


```
{
  "devices": {
      "desktop": {
          "clicked": 8,
          "complained": 1,
          "opened": 8,
          "unique_clicked": 0,
          "unique_opened": 0,
          "unsubscribed": 0
      },
      "mobile": {
          "clicked": 3,
          "complained": 1,
          "opened": 5,
          "unique_clicked": 0,
          "unique_opened": 0,
          "unsubscribed": 0
      }
  },
  "tag": "exampletag"
}
```


4.9 Suppressions

Mailgun keeps three lists of addresses it blocks the delivery to: bounces, unsubscribes and complaints. These lists are populated automatically as Mailgun detects undeliverable addresses you try to send to and as recipients unsubscribe from your mailings or mark your emails as a spam (for ESPs that provide FBL). You can also add/remove addresses from any of these lists using the API.

It's important to note that these suppression lists are unique to a sending domain and are not an account level (global) suppression list. If you want to add/remove the same address(es) from multiple domains, you'll need to do so for each domain.

4.9.1 Bounces

Bounce list stores events of delivery failures due to permanent recipient mailbox errors such as non-existent mailbox. Soft bounces (for example, mailbox is full) and other failures (for example, ESP rejects an email because it thinks it is spam) are not added to the list.

Subsequent delivery attempts to an address found in a bounce list are prevented to protect your sending reputation.

The bounce suppression API endpoint is available at:

```
v3/<domain>/bounces
```

Mailgun can notify your application every time a message bounces via a *permanent_fail webhook*.

View all bounces

```
GET /<domain>/bounces
```

Paginate over a list of bounces for a domain.

Note: Via this API method bounces are returned in the alphabetical order. If you wish to poll for the recently occurred bounces, please consider using the *Events API*.

Parameter	Description
limit	Maximum number of records to return (optional, default: 100, max: 10000)

Example:

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

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode getBounces() throws UnirestException {

        HttpResponse <JsonNode> request = Unirest.get("https://api.mailgun.net/v3/" +␣
↪YOUR_DOMAIN_NAME + "/bounces")
            .basicAuth("api", API_KEY)
            .asJson();

        return request.getBody();
    }
}
```

```
# Include the Autoloader (see "Libraries" for install instructions)
require 'vendor/autoload.php';
use Mailgun\Mailgun;

# Instantiate the client.
$mgClient = Mailgun::create('PRIVATE_API_KEY', 'https://API_HOSTNAME');
$domain   = 'YOUR_DOMAIN_NAME';

# Issue the call to the client.
$result = $mgClient->suppressions()->bounces()->index($domain);
```

```
def get_bounces():
    return requests.get(
        "https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/bounces",
        auth=("api", "YOUR_API_KEY"))
```

```ruby
def get_bounces
  RestClient.get "https://api:YOUR_API_KEY"\
  "@api.mailgun.net/v3/YOUR_DOMAIN_NAME/bounces"
end
```

```csharp
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class GetBouncesChunk
{

    public static void Main (string[] args)
    {
        Console.WriteLine (GetBounces ().Content.ToString ());
    }

    public static IRestResponse GetBounces ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.AddParameter ("domain", "YOUR_DOMAIN_NAME", ParameterType.UrlSegment);
        request.Resource = "{domain}/bounces";
        return client.Execute (request);
    }

}
```

```go
import (
    "context"
    "github.com/mailgun/mailgun-go/v3"
    "time"
)

func ListBounces(domain, apiKey string) ([]mailgun.Bounce, error) {
    mg := mailgun.NewMailgun(domain, apiKey)
    it := mg.ListBounces(nil)

    ctx, cancel := context.WithTimeout(context.Background(), time.Second*30)
    defer cancel()

    var page, result []mailgun.Bounce
    for it.Next(ctx, &page) {
        result = append(result, page...)
    }

    if it.Err() != nil {
        return nil, it.Err()
    }
    return result, nil
}
```

```
var DOMAIN = 'YOUR_DOMAIN_NAME';
var mailgun = require('mailgun-js')({ apiKey: "YOUR_API_KEY", domain: DOMAIN });

mailgun.get(`/${DOMAIN}/bounces/`, function (error, body) {
  console.log(body);
});
```

Expected response:

```
200
{
  "items":
    [
      {
        "address": "alice@example.com",
        "code": "550",
        "error": "No such mailbox",
        "created_at": "Fri, 21 Oct 2011 11:02:55 GMT"
      },
      ...
    ],
  "paging":
    {
      "first": <first page URL>
      "next": <next page URL>
      "previous": <previous page URL>
      "last": <last page URL>
    }
}
```

View a single bounce

```
GET /<domain>/bounces/<address>
```

Fetch a single bounce event by a given email address. Useful to check if a given email address has bounced before.

Example:

```
curl -s --user 'api:YOUR_API_KEY' -G \
    https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/bounces/foo@bar.com
```

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode getSingleBounce() throws UnirestException {

        HttpResponse <JsonNode> request = Unirest.get("https://api.mailgun.net/v3/" +␣
↪YOUR_DOMAIN_NAME + "/bounces/foo@bar.com")
            .basicAuth("api", API_KEY)
```

(continues on next page)

(continued from previous page)

```
            .asJson();

        return request.getBody();
    }
}
```

```
# Include the Autoloader (see "Libraries" for install instructions)
require 'vendor/autoload.php';
use Mailgun\Mailgun;

# Instantiate the client.
$mgClient   = Mailgun::create('PRIVATE_API_KEY', 'https://API_HOSTNAME');
$domain     = 'YOUR_DOMAIN_NAME';
$recipient  = 'bob@example.com';

# Issue the call to the client.
$result = $mgClient->suppressions()->bounces()->show($domain, $recipient);
```

```
def get_bounce():
    return requests.get(
        "https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/bounces/foo@bar.com",
        auth=("api", "YOUR_API_KEY"))
```

```
def get_bounce
  RestClient.get("https://api:YOUR_API_KEY"\
                 "@api.mailgun.net/v3/YOUR_DOMAIN_NAME/bounces"\
                 "/foo@bar.com"){|response, request, result| response }
end
```

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class GetBounceChunk
{

    public static void Main (string[] args)
    {
        Console.WriteLine (GetBounce ().Content.ToString ());
    }

    public static IRestResponse GetBounce ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.AddParameter ("domain", "YOUR_DOMAIN_NAME", ParameterType.UrlSegment);
        request.Resource = "{domain}/bounces/foo@bar.com";
        return client.Execute (request);
    }
```

(continues on next page)

(continued from previous page)

```
}
```

```
import (
    "context"
    "github.com/mailgun/mailgun-go/v3"
    "time"
)

func GetBounce(domain, apiKey string) (mailgun.Bounce, error) {
    mg := mailgun.NewMailgun(domain, apiKey)

    ctx, cancel := context.WithTimeout(context.Background(), time.Second*30)
    defer cancel()

    return mg.GetBounce(ctx, "foo@bar.com")
}
```

```
var DOMAIN = 'YOUR_DOMAIN_NAME';
var mailgun = require('mailgun-js')({ apiKey: "YOUR_API_KEY", domain: DOMAIN });

mailgun.get(`/${DOMAIN}/bounces`, function (error, body) {
  console.log(body);
});
```

Expected responses:

```
200
{
  "address": "foo@bar.com",
  "code": "550",
  "error": "No such mailbox",
  "created_at": "Fri, 21 Oct 2011 11:02:55 GMT"
}
```

```
404
{
  "message": "Address not found in bounces table"
}
```

Add a single bounce

```
POST /<domain>/bounces
```

Add a bounce record to the bounce list. Updates the existing record if the address is already there.

Parameter	Description
address	Valid email address
code	Error code (optional, default: 550)
error	Error description (optional, default: empty string)
created_at	Timestamp of a bounce event in *RFC2822 format* (optional, default: current time)

Example:

```
curl -s --user 'api:YOUR_API_KEY' \
    https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/bounces \
    -F address='bob@example.com'
```

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode addBounce() throws UnirestException {

        HttpResponse <JsonNode> request = Unirest.post("https://api.mailgun.net/v3/"␣
↪+ YOUR_DOMAIN_NAME + "/bounces")
            .basicAuth("api", YOUR_API_KEY)
            .field("address", "bob@example.com")
            .asJson();

        return request.getBody();
    }
}
```

```
# Include the Autoloader (see "Libraries" for install instructions)
require 'vendor/autoload.php';
use Mailgun\Mailgun;

# Instantiate the client.
$mgClient = Mailgun::create('PRIVATE_API_KEY', 'https://API_HOSTNAME');
$domain   = 'YOUR_DOMAIN_NAME';
$recipient = 'bob@example.com';

# Issue the call to the client.
$result = $mgClient->suppressions()->bounces()->create($domain, $recipient);
```

```
def add_bounce():
    return requests.post(
        "https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/bounces",
        auth=("api", "YOUR_API_KEY"),
        data={'address':'bob@example.com'})
```

```
def add_bounce
  RestClient.post("https://api:YOUR_API_KEY"\
                  "@api.mailgun.net/v3/YOUR_DOMAIN_NAME/bounces",
                  :address => 'bob@example.com')
end
```

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class AddBounceChunk
```

(continues on next page)

(continued from previous page)

```
{

    public static void Main (string[] args)
    {
        Console.WriteLine (AddBounce ().Content.ToString ());
    }

    public static IRestResponse AddBounce ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.Resource = "{domain}/bounces";
        request.AddParameter ("domain", "YOUR_DOMAIN_NAME", ParameterType.UrlSegment);
        request.AddParameter ("address", "bob@example.com");
        request.Method = Method.POST;
        return client.Execute (request);
    }

}
```

```
import (
    "context"
    "github.com/mailgun/mailgun-go/v3"
    "time"
)

func AddBounce(domain, apiKey string) error {
    mg := mailgun.NewMailgun(domain, apiKey)

    ctx, cancel := context.WithTimeout(context.Background(), time.Second*30)
    defer cancel()

    return mg.AddBounce(ctx, "bob@example.com", "550", "Undeliverable message error")
}
```

```
var DOMAIN = 'YOUR_DOMAIN_NAME';
var mailgun = require('mailgun-js')({ apiKey: "YOUR_API_KEY", domain: DOMAIN });

mailgun.post(`/${DOMAIN}/bounces`, {'address': 'bob@example.com'}, function (error,
↪body) {
  console.log(body);
});
```

Expected response:

```
200
{
  "message": "Address has been added to the bounces table",
  "address": "bob@example.com"
}
```

Add multiple bounces

```
POST /<domain>/bounces, Content-Type: application/json
```

Add multiple bounce records to the bounce list in a single API call.

Request body is expected to be a valid JSON encoded string containing up to 1000 bounce records in the following format.

```
[
  {
    "address": "alice@example.com",
    "code": "550",
    "error": "Bounced",
    "created_at": "Thu, 13 Oct 2011 18:02:00 UTC"
  },
  {
    "address": "bob@example.com",
    "code": "550",
    "error": "Bounced"
  },
  {
    "address": "carol@example.com",
    "code": "550"
  },
  {
    "address": "dan@example.com"
  }
]
```

Fields within each individual bounce record are the same as for the "add a single bounce" API method, with the same defaults and optionality rules.

Note: The current versions of our language libraries do not support adding multiple bounces yet.

Expected response:

```
200
{
  "message": "4 addresses have been added to the bounces table"
}
```

Import a list of bounces

```
POST /<domain>/bounces/import, Content-Type: multipart/form-data
```

Import a CSV file containing a list of addresses to add to the bounce list.

CSV file must be 25MB or under and must contain the following column headers: *address,code,error,created_at*

Column	Description
address	Valid email address
code	Error code (optional, default: 550)
error	Error description (optional, default: empty string)
created_at	Timestamp of a bounce event in *RFC2822 format* (optional, default: current time)

Expected response:

```
200
{
  "message": "file uploaded successfully"
}
```

Delete a single bounce

```
DELETE /<domain>/bounces/<address>
```

Clears a given bounce event. The delivery to the deleted email address resumes until it bounces again.

Expected response:

```
200
{
  "message": "Bounced address has been removed"
}
```

Delete an entire bounce list

```
DELETE /<domain>/bounces
```

Clears all bounced email addresses for a domain. Delivery to the deleted email addresses will no longer be suppressed.

Expected response:

```
200
{
  "message": "Bounced addresses for this domain have been removed"
}
```

4.9.2 Unsubscribes

Unsubscribe list stores email addresses of recipients who unsubscribed from your mailings by clicking a Mailgun generated unsubscribe link.

Mailgun allows you to quickly add "Unsubscribe me" feature to your outgoing emails without any programming on your end. You can enable this in your Control Panel under your domain settings.

The ubsubscribe suppression API endpoint is available at:

```
v3/<domain>/unsubscribes
```

Mailgun can notify your application every time a user unsubscribes via an *unsubscribed webhook*.

View all unsubscribes

```
GET /<domain>/unsubscribes
```

Paginate over a list of unsubscribes for a domain.

Note: Via this API method unsubscribes are returned in the alphabetical order. If you wish to poll for the recently occurred unsubscribes, please consider using the *Events API*.

Parameter	Description
limit	Number of records to return (optional, default: 100, max: 10000)

Example:

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

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode getUnsubscribes() throws UnirestException {

        HttpResponse <JsonNode> request = Unirest.get("https://api.mailgun.net/v3/" +␣
↪YOUR_DOMAIN_NAME + "/unsubscribes")
            .basicAuth("api", API_KEY)
            .asJson();

        return request.getBody();
    }
}
```

```
# Include the Autoloader (see "Libraries" for install instructions)
require 'vendor/autoload.php';
use Mailgun\Mailgun;

# Instantiate the client.
$mgClient = Mailgun::create('PRIVATE_API_KEY', 'https://API_HOSTNAME');
$domain = 'YOUR_DOMAIN_NAME';

# Issue the call to the client.
$result = $mgClient->suppressions()->unsubscribes()->index($domain);
```

```
def get_unsubscribes():
    return requests.get(
        "https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/unsubscribes",
        auth=("api", "YOUR_API_KEY"))
```

```
def get_unsubscribes
  RestClient.get "https://api:YOUR_API_KEY"\
  "@api.mailgun.net/v3/YOUR_DOMAIN_NAME/unsubscribes"
end
```

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class GetUnsubscribesChunk
{

    public static void Main (string[] args)
    {
        Console.WriteLine (GetUnsubscribes ().Content.ToString ());
    }

    public static IRestResponse GetUnsubscribes ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.AddParameter ("domain", "YOUR_DOMAIN_NAME", ParameterType.UrlSegment);
        request.Resource = "{domain}/unsubscribes";
        return client.Execute (request);
    }


}
```

```
import (
    "context"
    "github.com/mailgun/mailgun-go/v3"
    "time"
)

func ListUnsubscribes(domain, apiKey string) ([]mailgun.Unsubscribe, error) {
    mg := mailgun.NewMailgun(domain, apiKey)
    it := mg.ListUnsubscribes(nil)

    ctx, cancel := context.WithTimeout(context.Background(), time.Second*30)
    defer cancel()

    var page, result []mailgun.Unsubscribe
    for it.Next(ctx, &page) {
        result = append(result, page...)
    }

    if it.Err() != nil {
        return nil, it.Err()
    }
    return result, nil
}
```

```
var DOMAIN = 'YOUR_DOMAIN_NAME';
var mailgun = require('mailgun-js')({ apiKey: "YOUR_API_KEY", domain: DOMAIN });

mailgun.get(`/${DOMAIN}/unsubscribes`, function (error, body) {
  console.log(body);
```

(continues on next page)

(continued from previous page)

```
});
```

Expected response:

```
200
{
  "items":
    [
      {
        "address": "alice@example.com",
        "tag": "*",
        "created_at": "Fri, 21 Oct 2011 11:02:55 GMT"
      },
      ...
    ],
  "paging":
    {
      "first": <first page URL>,
      "next": <next page URL>
      "previous": <previous page URL>,
      "last": <last page URL>
    }
}
```

View a single unsubscribe

```
GET /<domain>/unsubscribes/<address>
```

Fetch a single unsubscribe record. Can be used to check if a given address is present in the list of unsubscribed users.

Expected responses:

```
200
{
  "address": "alice@example.com",
  "tag": "*",
  "created_at": "Fri, 21 Oct 2011 11:02:55 GMT"
}
```

```
404
{
  "message": "Address not found in unsubscribers table"
}
```

Add a single unsubscribe

```
POST /<domain>/unsubscribes
```

Add an address to the unsubscribe table.

Parameter	Description
address	Valid email address
tag	Tag to unsubscribe from, use * to unsubscribe an address from all domain's correspondence (optional, default: *)
created_at	Timestamp of an unsubscribe event in *RFC2822 format* (optional, default: current time)

Example:

```
curl -s --user 'api:YOUR_API_KEY' \
   https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/unsubscribes \
   -F address='bob@example.com' \
   -F tag='*'
```

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode addUnsubscribeAll() throws UnirestException {

        HttpResponse <JsonNode> request = Unirest.post("https://api.mailgun.net/v3/"␣
↪+ YOUR_DOMAIN_NAME + "/unsubscribes")
            .basicAuth("api", API_KEY)
            .field("address", "bob@example.com")
            .field("tag", "*")
            .asJson();

        return request.getBody();
    }
}
```

```
# Include the Autoloader (see "Libraries" for install instructions)
require 'vendor/autoload.php';
use Mailgun\Mailgun;

# Instantiate the client.
$mgClient    = Mailgun::create('PRIVATE_API_KEY', 'https://API_HOSTNAME');
$domain      = 'YOUR_DOMAIN_NAME';
$recipient   = 'bob@example.com';
$tag         = '*';

# Issue the call to the client.
$result = $mgClient->suppressions()->unsubscribes()->create($domain, $recipient,
↪$tag);
```

```
def unsubscribe_from_all():
    return requests.post(
        "https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/unsubscribes",
        auth=("api", "YOUR_API_KEY"),
        data={'address':'bob@example.com', 'tag': '*'})
```

```ruby
def unsubscribe_from_all
  RestClient.post "https://api:YOUR_API_KEY"\
  "@api.mailgun.net/v3/YOUR_DOMAIN_NAME/unsubscribes",
  :address => 'bob@example.com',
  :tag => '*'
end
```

```csharp
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class AddUnsubscribeAllChunk
{

    public static void Main (string[] args)
    {
        Console.WriteLine (UnsubscribeFromAll ().Content.ToString ());
    }

    public static IRestResponse UnsubscribeFromAll ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.Resource = "{domain}/unsubscribes";
        request.AddParameter ("domain", "YOUR_DOMAIN_NAME", ParameterType.UrlSegment);
        request.AddParameter ("address", "bob@example.com");
        request.AddParameter ("tag", "*");
        request.Method = Method.POST;
        return client.Execute (request);
    }

}
```

```go
import (
    "context"
    "github.com/mailgun/mailgun-go/v3"
    "time"
)

func CreateUnsubscribe(domain, apiKey string) error {
    mg := mailgun.NewMailgun(domain, apiKey)

    ctx, cancel := context.WithTimeout(context.Background(), time.Second*30)
    defer cancel()

    return mg.CreateUnsubscribe(ctx, "bob@example.com", "*")
}
```

```nodejs
var DOMAIN = 'YOUR_DOMAIN_NAME';
var mailgun = require('mailgun-js')({ apiKey: "YOUR_API_KEY", domain: DOMAIN });
```

(continues on next page)

(continued from previous page)

```
mailgun.post(`/${DOMAIN}/unsubscribes`, {"address": 'bob@example.com', "tag":'*'},
↪function (error, body) {
  console.log(body);
});
```

Expected response:

```
200
{
  "message": "Address has been added to the unsubscribes table",
  "address": "bob@example.com"
}
```

Add multiple unsubscribes

```
POST /<domain>/unsubscribes, Content-Type: application/json
```

Add multiple unsubscribe records to the unsubscribe list in a single API call.

Request body is expected to be a valid JSON encoded string containing up to 1000 unsubscribe records in the following format.

```
[
  {
    "address": "alice@example.com",
    "tags": ["some tag"],
    "created_at": "Thu, 13 Oct 2011 18:02:00 UTC"
  },
  {
    "address": "bob@example.com",
    "tags": ["*"],
  },
  {
    "address": "carol@example.com"
  }
]
```

Fields within each individual unsubscribe record are the same as for the "add a single unsubscribe" API method, with the same defaults and optionality rules.

Note: The current versions of our language libraries do not support adding multiple unsubscribes yet.

Expected response:

```
200
{
  "message": "3 addresses have been added to the unsubscribes table"
}
```

Import a list of unsubscribes

```
POST /<domain>/unsubscribes/import, Content-Type: multipart/form-data
```

Import a CSV file containing a list of addresses to add to the unsubscribe list.

CSV file must be 25MB or under and must contain the following column headers: *address,tag,created_at*

Column	Description
address	Valid email address
tags	Tag to unsubscribe from, use * to unsubscribe an address from all domain's correspondence (optional, default: *)
created_at	Timestamp of an unsubscribe event in *RFC2822 format* (optional, default: current time)

Expected response:

```
200
{
  "message": "file uploaded successfully"
}
```

Delete a single unsubscribe

```
DELETE /<domain>/unsubscribes/<address>
```

Remove an address from the unsubscribes list. If `tag` parameter is not provided, completely removes an address from the list.

Parameter	Description
tag	Specific tag to remove (optional)

Expected response:

```
200
{
  "message": "Unsubscribe event has been removed"
}
```

4.9.3 Complaints

Complaint list stores email addresses of recipients who marked your messages as a spam (for ESPs that support FBL).

The complaint API endpoint is available at:

```
v3/<domain>/complaints
```

Mailgun can notify your application every time a recipient flags your message as spam via a *complained webhook*.

View all complaints

```
GET /<domain>/complaints
```

Paginate over a list of complaints for a domain.

Note: Via this API method complaints are returned in the alphabetical order. If you wish to poll for the recently occurred complaints, please consider using the *Events API*.

Parameter	Description
limit	Maximum number of records to return (optional, default: 100, max: 10000)

Example:

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

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode getComplaints() throws UnirestException {

        HttpResponse <JsonNode> request = Unirest.get("https://api.mailgun.net/v3/" +␣
↪YOUR_DOMAIN_NAME + "/complaints")
            .basicAuth("api", API_KEY)
            .queryString("limit", "100")
            .asJson();

        return request.getBody();
    }
}
```

```
# Include the Autoloader (see "Libraries" for install instructions)
require 'vendor/autoload.php';
use Mailgun\Mailgun;

# Instantiate the client.
$mgClient    = Mailgun::create('PRIVATE_API_KEY', 'https://API_HOSTNAME');
$domain      = 'YOUR_DOMAIN_NAME';

# Issue the call to the client.
$result = $mgClient->suppressions()->complaints()->index($domain);
```

```
def get_complaints():
    return requests.get(
        "https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/complaints",
        auth=("api", "YOUR_API_KEY"))
```

```ruby
def get_complaints
  RestClient.get "https://api:YOUR_API_KEY"\
  "@api.mailgun.net/v3/YOUR_DOMAIN_NAME/complaints"
end
```

```csharp
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class GetComplaintsChunk
{

    public static void Main (string[] args)
    {
        Console.WriteLine (GetComplaints ().Content.ToString ());
    }

    public static IRestResponse GetComplaints ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.AddParameter ("domain", "YOUR_DOMAIN_NAME", ParameterType.UrlSegment);
        request.Resource = "{domain}/complaints";
        return client.Execute (request);
    }

}
```

```go
import (
    "context"
    "github.com/mailgun/mailgun-go/v3"
    "time"
)

func ListComplaints(domain, apiKey string) ([]mailgun.Complaint, error) {
    mg := mailgun.NewMailgun(domain, apiKey)
    it := mg.ListComplaints(nil)

    ctx, cancel := context.WithTimeout(context.Background(), time.Second*30)
    defer cancel()

    var page, result []mailgun.Complaint
    for it.Next(ctx, &page) {
        result = append(result, page...)
    }

    if it.Err() != nil {
        return nil, it.Err()
    }
    return result, nil
}
```

```
var DOMAIN = 'YOUR_DOMAIN_NAME';
var mailgun = require('mailgun-js')({ apiKey: "YOUR_API_KEY", domain: DOMAIN });

mailgun.get(`/${DOMAIN}/complaints`, function (error, body) {
  console.log(body);
});
```

Expected response:

```
200
{
  "items":
    [
      {
        "address": "alice@example.com",
        "created_at": "Fri, 21 Oct 2011 11:02:55 GMT"
      },
      ...
    ],
  "paging":
    {
      "first": <first page URL>
      "next": <next page URL>
      "previous": <previous page URL>,
      "last": <last page URL>
    }
}
```

View a single complaint

```
GET /<domain>/complaints/<address>
```

Fetch a single spam complaint by a given email address. This is useful to check if a particular user has complained.

Example:

```
curl -s --user 'api:YOUR_API_KEY' -G \
   https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/complaints/baz@example.com
```

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode getComplaint() throws UnirestException {

        HttpResponse <JsonNode> request = Unirest.get("https://api.mailgun.net/v3/" +␣
↪YOUR_DOMAIN_NAME + "/complaints/baz@example.com")
            .basicAuth("api", API_KEY)
            .asJson();
```

(continues on next page)

(continued from previous page)

```
        return request.getBody();
    }
}
```

```
# Include the Autoloader (see "Libraries" for install instructions)
require 'vendor/autoload.php';
use Mailgun\Mailgun;

# Instantiate the client.
$mgClient   = Mailgun::create('PRIVATE_API_KEY', 'https://API_HOSTNAME');
$domain     = 'YOUR_DOMAIN_NAME';
$recipient  = 'bob@example.com';

# Issue the call to the client.
$result = $mgClient->suppressions()->complaints()->show($domain, $recipient);
```

```
def get_complaint():
    return requests.get(
        "https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/complaints/baz@example.com",
        auth=("api", "YOUR_API_KEY"))
```

```
def get_complaint
  RestClient.get("https://api:YOUR_API_KEY"\
                 "@api.mailgun.net/v3/YOUR_DOMAIN_NAME/complaints/"\
                 "baz@example.com"){|response, request, result| response }
end
```

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class GetComplaintChunk
{

    public static void Main (string[] args)
    {
        Console.WriteLine (GetComplaint ().Content.ToString ());
    }

    public static IRestResponse GetComplaint ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.AddParameter ("domain", "YOUR_DOMAIN_NAME", ParameterType.UrlSegment);
        request.Resource = "{domain}/complaints/baz@example.com";
        return client.Execute (request);
    }

}
```

```
import (
    "context"
    "github.com/mailgun/mailgun-go/v3"
    "time"
)

func GetComplaints(domain, apiKey string) (mailgun.Complaint, error) {
    mg := mailgun.NewMailgun(domain, apiKey)

    ctx, cancel := context.WithTimeout(context.Background(), time.Second*30)
    defer cancel()

    return mg.GetComplaint(ctx, "baz@example.com")
}
```

```
var DOMAIN = 'YOUR_DOMAIN_NAME';
var mailgun = require('mailgun-js')({ apiKey: "YOUR_API_KEY", domain: DOMAIN });

mailgun.get(`/${DOMAIN}/complaints/baz@example.com`, function (error, body) {
  console.log(body);
});
```

Expected response:

```
200
{
  "address": "baz@example.com",
  "created_at": "Fri, 21 Oct 2011 11:02:55 GMT"
}
```

```
404
{
  "message": "No spam complaints found for this address"
}
```

Add a single complaint

```
POST /<domain>/complaints
```

Add an address to the complaints list.

Parameter	Description
address	Valid email address
created_at	Timestamp of a complaint event in *RFC2822 format* (optional, default: current time)

Example:

```
curl -s --user 'api:YOUR_API_KEY' \
   https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/complaints \
   -F address='bob@example.com'
```

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
```

(continues on next page)

(continued from previous page)

```
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    public static JsonNode addComplaint() throws UnirestException {

        HttpResponse <JsonNode> request = Unirest.post("https://api.mailgun.net/v3/"␣
↪+ YOUR_DOMAIN_NAME + "/complaints")
            .basicAuth("api", API_KEY)
            .field("address", "bob@example.com")
            .asJson();
        return request.getBody();
    }
}
```

```
# Include the Autoloader (see "Libraries" for install instructions)
require 'vendor/autoload.php';
use Mailgun\Mailgun;

# Instantiate the client.
$mgClient = Mailgun::create('PRIVATE_API_KEY', 'https://API_HOSTNAME');
$domain = 'YOUR_DOMAIN_NAME';
$recipient = 'bob@example.com';

# Issue the call to the client.
$result = $mgClient->suppressions()->complaints()->create($domain, $recipient);
```

```
def add_complaint():
    return requests.post(
        "https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/complaints",
        auth=("api", "YOUR_API_KEY"),
        data={'address': 'bob@example.com'})
```

```
def add_complaint
  RestClient.post "https://api:YOUR_API_KEY"\
  "@api.mailgun.net/v3/YOUR_DOMAIN_NAME/complaints",
  :address => 'bob@example.com'
end
```

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class AddComplaintChunk
{

    public static void Main (string[] args)
    {
        Console.WriteLine (AddComplaint ().Content.ToString ());
    }

    public static IRestResponse AddComplaint ()
    {
```

(continues on next page)

(continued from previous page)

```
        RestClient client = new RestClient();
        client.BaseUrl = new Uri("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator("api",
                                       "YOUR_API_KEY");
        RestRequest request = new RestRequest();
        request.Resource = "{domain}/complaints";
        request.AddParameter("domain", "YOUR_DOMAIN_NAME", ParameterType.UrlSegment);
        request.AddParameter("address", "bob@example.com");
        request.Method = Method.POST;
        return client.Execute(request);
    }

}
```

```
import (
    "context"
    "github.com/mailgun/mailgun-go/v3"
    "time"
)

func CreateComplaint(domain, apiKey string) error {
    mg := mailgun.NewMailgun(domain, apiKey)

    ctx, cancel := context.WithTimeout(context.Background(), time.Second*30)
    defer cancel()

    return mg.CreateComplaint(ctx, "bob@example.com")
}
```

```
var DOMAIN = 'YOUR_DOMAIN_NAME';
var mailgun = require('mailgun-js')({ apiKey: "YOUR_API_KEY", domain: DOMAIN });

mailgun.post(`/${DOMAIN}/complaints`, {"address" : "bob@example.com"}, function␣
↪(error, body) {
  console.log(body);
});
```

Expected response:

```
200
{
  "message": "Address has been addded to the complaints table",
  "address": "bob@example.com"
}
```

Add multiple complaints

```
POST /<domain>/complaints, Content-Type: application/json
```

Add multiple complaint records to the complaint list in a single API call.

Request body is expected to be a valid JSON encoded string containing up to 1000 complaint records in the following format.

```
[
  {
    "address": "alice@example.com",
    "created_at": "Thu, 13 Oct 2011 18:02:00 UTC"
  },
  {
    "address": "bob@example.com"
  }
]
```

Fields within each individual complaint record are the same as for the "add a single unsubscribe" API method, with the same defaults and optionality rules.

Note: The current versions of our language libraries do not support adding multiple complaints yet.

Expected response:

```
200
{
  "message": "2 complaint addresses have been added to the complaints table"
}
```

Import a list of complaints

```
POST /<domain>/complaints/import, Content-Type: multipart/form-data
```

Import a CSV file containing a list of addresses to add to the complaint list.

CSV file must be 25MB or under and must contain the following column headers: *address,created_at*

Column	Description
address	Valid email address
created_at	Timestamp of a complaint event in *RFC2822 format* (optional, default: current time)

Expected response:

```
200
{
  "message": "file uploaded successfully"
}
```

Delete a single complaint

```
DELETE /<domain>/complaints/<address>
```

Remove a given spam complaint.

Expected response:

```
200
{
  "message": "Spam complaint has been removed"
}
```

4.9.4 Whitelists

The whitelist API provides the ability to whitelist specific addresses from being added to bounce list. You can whitelist by domain name (i.e example.com) or by specific address (i.e. alice@example.com). Mailgun doesn't add an address to bounce list if the address is whitelisted. This API is very useful if you test against your private services and don't want to constantly clean up bounce lists.

View all whitelist records

```
GET /<domain>/whitelists
```

Paginate over a whitelists for a domain.

Parameter	Description
limit	Number of records to return (optional, default: 100, max: 10000)

Example:

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

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode getBounces() throws UnirestException {

        HttpResponse <JsonNode> request = Unirest.get("https://api.mailgun.net/v3/" +␣
↪YOUR_DOMAIN_NAME + "/whitelists")
            .basicAuth("api", API_KEY)
            .asJson();

        return request.getBody();
    }
}
```

```
# Currently, the PHP SDK does not support Suppression Whiteslist endpoint.
# Consider using the following php curl function.
function add_domain_whitelist() {
  $ch = curl_init();

  curl_setopt($ch, CURLOPT_HTTPAUTH, CURLAUTH_BASIC);
```

(continues on next page)

(continued from previous page)

```
  curl_setopt($ch, CURLOPT_USERPWD, 'api:PRIVATE_API_KEY');
  curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);

  curl_setopt($ch, CURLOPT_CUSTOMREQUEST, 'GET');
  curl_setopt($ch, CURLOPT_URL, 'https://api.mailgun.net/v3/domain.tld/whitelists');
  curl_setopt($ch, CURLOPT_POSTFIELDS, array(
      'address'=> 'bob@example.com'
  ));

  $result = curl_exec($ch);
  curl_close($ch);

  return $result;
}
```

```
def get_whitelists():
    return requests.get(
        "https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/whitelists",
        auth=("api", "YOUR_API_KEY"))
```

```
def get_whitelists
  RestClient.get "https://api:YOUR_API_KEY"\
  "@api.mailgun.net/v3/YOUR_DOMAIN_NAME/whitelists"
end
```

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class GetBouncesChunk
{

    public static void Main (string[] args)
    {
        Console.WriteLine (GetBounces ().Content.ToString ());
    }

    public static IRestResponse GetBounces ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.AddParameter ("domain", "YOUR_DOMAIN_NAME", ParameterType.UrlSegment);
        request.Resource = "{domain}/whitelists";
        return client.Execute (request);
    }

}
```

```
// Not supported yet
```

```
var DOMAIN = 'YOUR_DOMAIN_NAME';
var mailgun = require('mailgun-js')({ apiKey: "YOUR_API_KEY", domain: DOMAIN });

mailgun.get(`/${DOMAIN}/whitelists/`, function (error, body) {
  console.log(body);
});
```

Expected response:

```
200
{
  "items":
    [
      {
        "value": "alice@example.com",
        "reason": "reason of white listing"
        "type": "address",
        "createdAt": "Fri, 21 Oct 2011 11:02:55 UTC"
      },
      {
        "value": "test.com",
        "reason": "reason of white listing"
        "type": "domain",
        "createdAt": "Fri, 21 Oct 2012 11:02:56 UTC"
      }
      ...
    ],
  "paging":
    {
      "first": <first page URL>,
      "next": <next page URL>,
      "previous": <previous page URL>,
      "last": <last page URL>
    }
}
```

View a single whitelist record

```
GET /<domain>/whitelists/<address or domain>
```

Fetch a single whitelist record. Can be used to check if a given address or domain is present in the whitelist table

Expected responses:

```
200
{
  "value": "alice@example.com",
  "reason": "why the record was created"
  "type": "address",
  "createdAt": "Fri, 21 Oct 2011 11:02:55 GMT"
}
```

```
404
{
  "message": "Address/Domain not found in whitelists table"
}
```

Add a single whitelist record

```
POST /<domain>/whitelists
```

Add an address or domain to the whitelist table.

Parameter	Description
address	Valid email address if you would like to whitelist email address
domain	Valid domain name if you would like whitlist entire domain name

Note: The single request accepts either one *address* or *domain* parameter

Example:

```
curl -s --user 'api:YOUR_API_KEY' \
    https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/whitelists \
    -F domain='example.com'
```

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode addBounce() throws UnirestException {

        HttpResponse <JsonNode> request = Unirest.post("https://api.mailgun.net/v3/"␣
↪+ YOUR_DOMAIN_NAME + "/whitelists")
            .basicAuth("api", API_KEY)
            .field("domain", "example.com")
            .asJson();

        return request.getBody();
    }
}
```

```
# Currently, the PHP SDK does not support Suppression Whiteslist endpoint.
# Consider using the following php curl function.
function add_domain_whitelist() {
  $ch = curl_init();

  curl_setopt($ch, CURLOPT_HTTPAUTH, CURLAUTH_BASIC);
  curl_setopt($ch, CURLOPT_USERPWD, 'api:PRIVATE_API_KEY');
  curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);

  curl_setopt($ch, CURLOPT_CUSTOMREQUEST, 'POST');
  curl_setopt($ch, CURLOPT_URL, 'https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/
↪whitelists');
  curl_setopt($ch, CURLOPT_POSTFIELDS, array(
      'address'=> 'bob@example.com'
  ));
```

(continues on next page)

(continued from previous page)

```
  $result = curl_exec($ch);
  curl_close($ch);

  return $result;
}
```

```
def add_whitelist():
    return requests.post(
        "https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/whitelists",
        auth=("api", "YOUR_API_KEY"),
        data={'address':'example.com'})
```

```
def add_whitelist
  RestClient.post("https://api:YOUR_API_KEY"\
                  "@api.mailgun.net/v3/YOUR_DOMAIN_NAME/whitelists",
                  :domain => 'example.com')
end
```

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class AddBounceChunk
{

    public static void Main (string[] args)
    {
        Console.WriteLine (AddBounce ().Content.ToString ());
    }

    public static IRestResponse AddBounce ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.Resource = "{domain}/whitelists";
        request.AddParameter ("domain", "YOUR_DOMAIN_NAME", ParameterType.UrlSegment);
        request.AddParameter ("domain", "example.com");
        request.Method = Method.POST;
        return client.Execute (request);
    }

}
```

```
// Not implemented
```

```
var DOMAIN = 'YOUR_DOMAIN_NAME';
var mailgun = require('mailgun-js')({ apiKey: "YOUR_API_KEY", domain: DOMAIN });

mailgun.post(`/${DOMAIN}/whitelists`, {'domain': 'example.com'}, function (error␣
↪body) {
```

(continues on next page)

(continued from previous page)

```
  console.log(body);
});
```

Expected response:

```
200
{
  "message":"Address/Domain has been added to the whitelists table",
  "type":"domain",
  "value":"example.com"
}
```

Import a list of addresses and/or domains

```
POST /<domain>/whitelists/import, Content-Type: multipart/form-data
```

Import a CSV file containing a list of addresses and/or domains to add to the whitelist.

CSV file must be 25MB or under and must contain the following column headers: *address,domain*

Column	Description
address	Valid email address if you would like to whitelist email address
domain	Valid domain name if you would like whitlist entire domain name

Expected response:

```
200
{
  "message": "file uploaded successfully"
}
```

Delete a single record from whitelist table

```
DELETE /<domain>/whitelists/<address or domain>
```

Remove a given record from whitelist table.

Expected response:

```
200
{
  "message":"Whitelist address/domain has been removed",
  "value":"alice@example.com"
}
```

4.10 Routes

Mailgun Routes are a powerful way to handle the incoming traffic. See *Routes* section in the User Manual to learn more about how they work. This API allows you to work with routes programmatically.

The routes API endpoint is available at:

```
v3/routes
```

Routes are comprised of the following arguments:

- A filter (when to do something).
- A priority (in what order).
- An action (what to do).

4.10.1 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 smtp recipient of the incoming message against the regular expression pattern. For example this will match all messages coming to any recipient at @bar.com:

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

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. For this will match any message with a word "support" in its subject:

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

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.

4.10.2 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")
```

store(notification endpoint)

Stores the message temporarily (for up to 3 days) on Mailgun's servers so that you can retrieve them later. This is helpful for large messages that may cause time outs or if you just want to retrieve them later.

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")
```

You can see a full list of parameters we will post to your URL in the *Routes* section of the User Manual. You can also get the locations of messages through the *Events API* and then retrieve the message through the *Messages API*.

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.

```
GET /routes
```

Fetches the list of routes. Note that routes are defined globally, per account, not per domain as most of other API calls.

Parameter	Description
limit	Maximum number of records to return. (100 by default)
skip	Number of records to skip. (0 by default)

```
GET /routes/<id>
```

Returns a single route object based on its ID. See examples below.

Parameter	Description
id	ID of the route

```
POST /routes
```

Creates a new route.

Parameter	Description
priority	Integer: smaller number indicates higher priority. Higher priority routes are handled first. Defaults to 0.
description	An arbitrary string.
expression	A filter expression like `match_recipient('.*@gmail.com')`
action	Route action. This action is executed when the expression evaluates to True. Example: `forward("alice@example.com")` You can pass multiple `action` parameters.

```
PUT /routes/<id>
```

Updates a given route by ID. All parameters are optional: this API call only updates the specified fields leaving others unchanged.

Parameter	Description
id	ID of the route
priority	Integer: smaller number indicates higher priority. Higher priority routes are handled first.
description	An arbitrary string.
expression	A filter expression like `match_recipient('.*@gmail.com')`
action	Route action. This action is executed when the expression evaluates to True. Example: `forward("alice@example.com")` You can pass multiple `action` parameters.

```
DELETE /routes/<id>
```

Deletes a route based on the id.

Parameter	Description
id	ID of the route

4.10.3 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()'
```

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode createRoute() throws UnirestException {

        HttpResponse <JsonNode> request = Unirest.post("https://api.mailgun.net/v3/
↪routes")
            .basicAuth("api", API_KEY)
            .field("priority", "0")
            .field("description", "sample route")
            .field("expression", "match_recipient('.*@YOUR_DOMAIN_NAME')")
            .field("action", "forward('http://myhost.com/messages/')")
```

(continues on next page)

(continued from previous page)

```
            .field("action", "stop()")
            .asJson();

        return request.getBody();
    }
}
```

```
# Include the Autoloader (see "Libraries" for install instructions)
require 'vendor/autoload.php';
use Mailgun\Mailgun;

# Instantiate the client.
$mgClient = Mailgun::create('PRIVATE_API_KEY', 'https://API_HOSTNAME');

# Define your expression, actions, and description
$expression  = 'match_recipient(".*@mg.example.com")';
$actions     = array('forward("my_address@example.com")', 'stop()');
$description = 'Catch All and Forward';

# Issue the call to the client.
$result = $mgClient->routes()->create($expression, $actions, $description);
```

```
def create_route():
    return requests.post(
        "https://api.mailgun.net/v3/routes",
        auth=("api", "YOUR_API_KEY"),
        data={"priority": 0,
              "description": "Sample route",
              "expression": "match_recipient('.*@YOUR_DOMAIN_NAME')",
              "action": ["forward('http://myhost.com/messages/')", "stop()"]})
```

```
def create_route
  data = {}
  data[:priority] = 0
  data[:description] = "Sample route"
  data[:expression] = "match_recipient('.*@YOUR_DOMAIN_NAME')"
  data[:action] = []
  data[:action] << "forward('http://myhost.com/messages/')"
  data[:action] << "stop()"
  RestClient.post "https://api:YOUR_API_KEY"\
  "@api.mailgun.net/v3/routes", data
end
```

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class CreateRouteChunk
{

    public static void Main (string[] args)
    {
        Console.WriteLine (CreateRoute ().Content.ToString ());
    }
```

(continues on next page)

(continued from previous page)

```
    public static IRestResponse CreateRoute ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.Resource = "routes";
        request.AddParameter ("priority", 0);
        request.AddParameter ("description", "Sample route");
        request.AddParameter ("expression", "match_recipient('.*@YOUR_DOMAIN_NAME')");
        request.AddParameter ("action",
                              "forward('http://myhost.com/messages/')");
        request.AddParameter ("action", "stop()");
        request.Method = Method.POST;
        return client.Execute (request);
    }

}
```

```
import (
    "context"
    "github.com/mailgun/mailgun-go/v3"
    "time"
)

func CreateRoute(domain, apiKey string) (mailgun.Route, error) {
    mg := mailgun.NewMailgun(domain, apiKey)

    ctx, cancel := context.WithTimeout(context.Background(), time.Second*30)
    defer cancel()

    return mg.CreateRoute(ctx, mailgun.Route{
        Priority:    1,
        Description: "Sample Route",
        Expression:  "match_recipient(\".*@YOUR_DOMAIN_NAME\")",
        Actions: []string{
            "forward(\"http://example.com/messages/\")",
            "stop()",
        },
    })
}
```

```
var DOMAIN = 'YOUR_DOMAIN_NAME';
var mailgun = require('mailgun-js')({ apiKey: "YOUR_API_KEY", domain: DOMAIN });

mailgun.post('/routes', {"priority": 0, "description": 'Sample route', "expression":
↪'match_recipient(".*@YOUR_DOMAIN_NAME")', "action": 'forward("http://myhost.com/
↪messages/")', "action": 'stop()'}, function (error, body) {
  console.log(body);
});
```

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"
  }
}
```

Listing routes:

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

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode getRoutes() throws UnirestException {

        HttpResponse <JsonNode> request = Unirest.get("https://api.mailgun.net/v3/
↪routes")
            .basicAuth("api", API_KEY)
            .queryString("skip", "0")
            .queryString("limit", "5")
            .asJson();

        return request.getBody();
    }
}
```

```
# Include the Autoloader (see "Libraries" for install instructions)
require 'vendor/autoload.php';
use Mailgun\Mailgun;

# Instantiate the client.
$mgClient = Mailgun::create('PRIVATE_API_KEY', 'https://API_HOSTNAME');

# Issue the call to the client.
$result = $mgClient->routes()->index();
```

```
def get_routes():
    return requests.get(
```

(continues on next page)

(continued from previous page)

```
        "https://api.mailgun.net/v3/routes",
        auth=("api", "YOUR_API_KEY"),
        params={"skip": 1,
                "limit": 1})
```

```
def get_routes
  RestClient.get "https://api:YOUR_API_KEY"\
  "@api.mailgun.net/v3/routes", :params => {
    :skip => 1,
    :limit => 1
  }
end
```

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class GetRoutesChunk
{

    public static void Main (string[] args)
    {
        Console.WriteLine (GetRoutes ().Content.ToString ());
    }

    public static IRestResponse GetRoutes ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.Resource = "routes";
        request.AddParameter ("skip", 1);
        request.AddParameter ("limit", 1);
        return client.Execute (request);
    }

}
```

```
import (
    "context"
    "github.com/mailgun/mailgun-go/v3"
    "time"
)

func ListRoutes(domain, apiKey string) ([]mailgun.Route, error) {
    mg := mailgun.NewMailgun(domain, apiKey)
    it := mg.ListRoutes(nil)

    ctx, cancel := context.WithTimeout(context.Background(), time.Second*30)
    defer cancel()
```

(continues on next page)

(continued from previous page)

```
    var page, result []mailgun.Route
    for it.Next(ctx, &page) {
        result = append(result, page...)
    }

    if it.Err() != nil {
        return nil, it.Err()
    }
    return result, nil
}
```

```
var DOMAIN = 'YOUR_DOMAIN_NAME';
var mailgun = require('mailgun-js')({ apiKey: "YOUR_API_KEY", domain: DOMAIN });

mailgun.get('/routes', {"skip": 0, "limit": 5}, function (error, body) {
  console.log(body);
});
```

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
```

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode getSingleRoute() throws UnirestException {

        HttpResponse <JsonNode> request = Unirest.get("https://api.mailgun.net/v3/
↪routes/YOUR_ROUTE_ID")
            .basicAuth("api", API_KEY)
```

(continues on next page)

(continued from previous page)

```
            .asJson();

        return request.getBody();
    }
}
```

```
# Include the Autoloader (see "Libraries" for install instructions)
require 'vendor/autoload.php';
use Mailgun\Mailgun;

# Instantiate the client.
$mgClient = Mailgun::create('PRIVATE_API_KEY', 'https://API_HOSTNAME');
$route_id = '5d9fde0fd8b861ec16cf2549'

# Issue the call to the client.
$result = $mgClient->routes()->show($route_id);
```

```
def get_route():
    return requests.get(
        "https://api.mailgun.net/v3/routes/4e97c1b2ba8a48567f007fb6",
        auth=("api", "YOUR_API_KEY"))
```

```
def get_route
  RestClient.
    get("https://api:YOUR_API_KEY"\
        "@api.mailgun.net/v3/routes/"\
        "4e97c1b2ba8a48567f007fb6"){|response, request, result| response }
end
```

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class GetRouteChunk
{

    public static void Main (string[] args)
    {
        Console.WriteLine (GetRoute ().Content.ToString ());
    }

    public static IRestResponse GetRoute ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.Resource = "routes/{id}";
        request.AddUrlSegment ("id", "4e97c1b2ba8a48567f007fb6");
        return client.Execute (request);
    }
```

(continues on next page)

(continued from previous page)

```
}
```

```
import (
    "context"
    "github.com/mailgun/mailgun-go/v3"
    "time"
)

func GetRoute(domain, apiKey string) (mailgun.Route, error) {
    mg := mailgun.NewMailgun(domain, apiKey)

    ctx, cancel := context.WithTimeout(context.Background(), time.Second*30)
    defer cancel()

    return mg.GetRoute(ctx, "route_id")
}
```

```
var DOMAIN = 'YOUR_DOMAIN_NAME';
var mailgun = require('mailgun-js')({ apiKey: "YOUR_API_KEY", domain: DOMAIN });

mailgun.get('/routes/your_route_id', function (error, body) {
  console.log(body);
});
```

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"
  }
}
```

Sample payload for a `store()` webhook:

```
Content-Type: multipart/alternative; boundary="001a114490d2c5be3d05433e6d03"
Date: Fri, 9 Dec 2016 13:04:51 -0600
From: Excited User <user@samples.mailgun.com>
Message-Id: <CABPem2N_Ucj3wRRZnLVpVF_fRjkTBXHZReZC3zY-hHsRa=T51g@samples.mailgun.com>
Mime-Version: 1.0
Subject: Message Routes
To: hook@sandboxdb91ab935a414789809f96c91229a0ee.mailgun.org
X-Envelope-From: <user@samples.mailgun.com>
X-Mailgun-Incoming: Yes
X-Originating-Ip: [2001:xxx:xxxx:xxx::beef:93]
body-html: <div dir="ltr">Testing Mailgun&#39;s forwarded and stored message routes :)
↪</div>
body-plain: Testing Mailgun's forwarded and stored message routes :)
domain: sandboxdb91ab935a414789809f96c91229a0ee.mailgun.org
```

(continues on next page)

(continued from previous page)

```
from: Excited User <user@samples.mailgun.com>
message-headers: [["X-Mailgun-Incoming", "Yes"], ["X-Envelope-From", "<user@samples.
↪mailgun.com>"], ["Mime-Version", "1.0"], ["X-Originating-Ip",
↪"[2001:xxx:xxxx:xxx::beef:93]"], ["From", "Excited User <user@samples.mailgun.com>
↪"], ["Date", "Fri, 9 Dec 2016 13:04:51 -0600"], ["Message-Id", "<CABPem2N_
↪Ucj3wRRZnLVpVF_fRjkTBXHZReZC3zY-hHsRa=T51g@samples.mailgun.com>"], ["Subject",
↪"Message Routes"], ["To", "hook@sandboxdb91ab935a414789809f96c91229a0ee.mailgun.org
↪"], ["Content-Type", "multipart/alternative; boundary=\
↪"001a114490d2c5be3d05433e6d03\""]]
message-url: https://si.api.mailgun.net/v3/domains/
↪sandboxdb91ab935a414789809f96c91229a0ee.mailgun.org/messages/
↪eyJwIjpmYWxzZSwiayI6IjFlOTZmNTkyLTAyOWItNDJkYi1iNjM5LTgzNTgwYzMxYjNhOCIsInMiOiIyMmNkYTRkZWFhIiwiYy
recipient: hook@sandboxdb91ab935a414789809f96c91229a0ee.mailgun.org
sender: user@samples.mailgun.com
signature: 6ed72df4b5f00af436fff03730dc8bda31bf5800fdf431d1da5c0009a639d57e
stripped-html: <div dir="ltr">Testing Mailgun&#39;s forwarded and stored message␣
↪routes :)</div>
stripped-signature:
stripped-text: Testing Mailgun's forwarded and stored message routes :)
subject: Message Routes
timestamp: 1481310293
token: f2a24f20007696fb23fd66ff0f59f17fac3f885324caaaec50
```

Sample payload for a `forward()` webhook:

```
Content-Type: multipart/alternative; boundary="001a114490d2c5be3d05433e6d03"
Date: Fri, 9 Dec 2016 13:04:51 -0600
From: Excited User <user@samples.mailgun.com>
Message-Id: <CABPem2N_Ucj3wRRZnLVpVF_fRjkTBXHZReZC3zY-hHsRa=T51g@samples.mailgun.com>
Mime-Version: 1.0
Subject: Message Routes
To: hook@sandboxdb91ab935a414789809f96c91229a0ee.mailgun.org
X-Envelope-From: <user@samples.mailgun.com>
X-Mailgun-Incoming: Yes
X-Originating-Ip: [2001:xxx:xxxx:xxx::beef:93]
body-html: <div dir="ltr">Testing Mailgun&#39;s forwarded and stored message routes :)
↪</div>
body-plain: Testing Mailgun's forwarded and stored message routes :)
from: Excited User <user@samples.mailgun.com>
message-headers: [["X-Mailgun-Incoming", "Yes"], ["X-Envelope-From", "<user@samples.
↪mailgun.com>"], ["Mime-Version", "1.0"], ["X-Originating-Ip",
↪"[2001:xxx:xxxx:xxx::beef:93]"], ["From", "Excited User <user@samples.mailgun.com>
↪"], ["Date", "Fri, 9 Dec 2016 13:04:51 -0600"], ["Message-Id", "<CABPem2N_
↪Ucj3wRRZnLVpVF_fRjkTBXHZReZC3zY-hHsRa=T51g@samples.mailgun.com>"], ["Subject",
↪"Message Routes"], ["To", "hook@sandboxdb91ab935a414789809f96c91229a0ee.mailgun.org
↪"], ["Content-Type", "multipart/alternative; boundary=\
↪"001a114490d2c5be3d05433e6d03\""]]
recipient: hook@sandboxdb91ab935a414789809f96c91229a0ee.mailgun.org
sender: user@samples.mailgun.com
signature: 17436304dd4dd094e9b8c3addb975acc6297718da468c2900dac4a43787c97596
stripped-html: <div dir="ltr">Testing Mailgun&#39;s forwarded and stored message␣
↪routes :)</div>
stripped-signature:
stripped-text: Testing Mailgun's forwarded and stored message routes :)
subject: Message Routes
timestamp: 1481310293
token: a71d0000ed34da6768198da96f9daaf8fb98adbccfdbd2fdaf
```

4.11 Webhooks

This API allows you to create, access, and delete webhooks programmatically.

The webhooks API endpoint is availble at:

```
v3/domains/<domain>/webhooks
```

Supported webhooks, and their documentation, are listed below:

Webhook Name	Documentation
clicked	*Tracking Clicks*
complained	*Tracking Spam Complaints*
delivered	*Tracking Deliveries*
opened	*Tracking Opens*
permanent_fail	*Tracking Failures*
temporary_fail	*Tracking Failures*
unsubscribed	*Tracking Unsubscribes*

```
GET /domains/<domain>/webhooks
```

Returns a list of webhooks set for the specified domain.

Parameter	Description
domain	Name of the domain

```
GET /domains/<domain>/webhooks/<webhookname>
```

Returns details about a the webhook specified in the URL.

Parameter	Description
domain	Name of the domain
id	Name of the webhook. (See above for supported webhooks)

```
POST /domains/<domain>/webhooks
```

Creates a new webhook.

Note: When adding a Clicked or Opened webhook, ensure that you also have tracking enabled.

Parameter	Description
domain	Name of the domain
id	Name of the webhook. (See above for supported webhooks)
url	URL for the webhook event. May be repeated up to 3 times.

```
PUT /domains/<domain>/webhooks/<webhookname>
```

Updates an existing webhook.

Parameter	Description
domain	Name of the domain
webhookname	Name of the webhook. (See above for supported webhooks)
url	URL for the webhook event. May be repeated up to 3 times.

```
DELETE /domains/<domain>/webhooks/<webhookname>
```

Deletes an existing webhook.

Note: Mailgun imposes a rate limit for the Webhook API endpoint. Users may issue no more than 300 requests per minute, per account. See the resultant rate limit response below.

Parameter	Description
domain	Name of the domain
webhookname	Name of the webhook. (See above for supported webhooks)

4.11.1 Examples

Return a list of webhooks set for the specified domain.

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

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode getWebhooks() throws UnirestException {

        HttpResponse <JsonNode> request = Unirest.get("https://api.mailgun.net/v3/
↪domains/" + YOUR_DOMAIN_NAME + "/webhooks")
            .basicAuth("api", API_KEY)
            .asJson();

        return request.getBody();
    }
}
```

```
# Include the Autoloader (see "Libraries" for install instructions)
require 'vendor/autoload.php';
use Mailgun\Mailgun;

# Instantiate the client.
$mgClient = Mailgun::create('PRIVATE_API_KEY', 'https://API_HOSTNAME');
$domain   = 'YOUR_DOMAIN_NAME';
```

(continues on next page)

(continued from previous page)

```
# Issue the call to the client.
$result = $mgClient->webhooks()->index($domain)
```

```
def get_bounces():
    return requests.get(
        "https://api.mailgun.net/v3/domains/YOUR_DOMAIN_NAME/webhooks",
        auth=("api", "YOUR_API_KEY"))
```

```
def get_webhooks
  RestClient.get "https://api:YOUR_API_KEY"\
                 "@api.mailgun.net/v3/domains/YOUR_DOMAIN_NAME/webhooks"
end
```

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class GetWebhooksChunk
{

    public static void Main (string[] args)
    {
        Console.WriteLine (GetWebhooks ().Content.ToString ());
    }

    public static IRestResponse GetWebhooks ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.AddParameter ("domain", "YOUR_DOMAIN_NAME", ParameterType.UrlSegment);
        request.Resource = "domains/{domain}/webhooks";
        return client.Execute (request);
    }

}
```

```
import (
    "context"
    "github.com/mailgun/mailgun-go/v3"
    "time"
)

func ListWebhooks(domain, apiKey string) (map[string]string, error) {
    mg := mailgun.NewMailgun(domain, apiKey)

    ctx, cancel := context.WithTimeout(context.Background(), time.Second*30)
    defer cancel()

    return mg.ListWebhooks(ctx)
}
```

```
var DOMAIN = 'YOUR_DOMAIN_NAME';
var mailgun = require('mailgun-js')({ apiKey: "YOUR_API_KEY", domain: DOMAIN });

mailgun.get(`/domain/${DOMAIN}/webhooks`, function (error, body) {
  console.log(body);
});
```

Sample response:

```
{
  "webhooks": {
    "opened": {
      "urls": [
                "https://your_domain.com/v1/opened",
                "https://your_domain.com/v2/opened"
              ]
    },
    "clicked": {
      "urls": [ "https://your_domain.com/v1/clicked" ]
    }
  }
}
```

Return a webhook for a specific event for the defined domain.

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

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode getWebhookEvent() throws UnirestException {

        HttpResponse <JsonNode> request = Unirest.get("https://api.mailgun.net/v3/
↪domains/" + YOUR_DOMAIN_NAME + "/webhooks/clicked")
            .basicAuth("api", API_KEY)
            .asJson();

        return request.getBody();
    }
}
```

```
# Include the Autoloader (see "Libraries" for install instructions)
require 'vendor/autoload.php';
use Mailgun\Mailgun;

# Instantiate the client.
$mgClient = Mailgun::create('PRIVATE_API_KEY', 'https://API_HOSTNAME');
$domain = 'YOUR_DOMAIN_NAME';
$webhook = 'delivered';
```

(continues on next page)

(continued from previous page)

```
# Issue the call to the client.
$result = $mgClient->webhooks()->show($domain, $webhook)
```

```
def get_domain():
    return requests.get(
        "https://api.mailgun.net/v3/domains/YOUR_DOMAIN_NAME/webhooks/clicked",
        auth=("api", "YOUR_API_KEY"))
```

```
def get_domain
  RestClient.get("https://api:YOUR_API_KEY"\
                 "@api.mailgun.net/v3/domains/YOUR_DOMAIN_NAME/webhooks/clicked"\
                 ){|response, request, result| response }
end
```

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class GetWebhookChunk
{

    public static void Main (string[] args)
    {
        Console.WriteLine (GetWebhook ().Content.ToString ());
    }

    public static IRestResponse GetWebhook ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.AddParameter ("domain", "YOUR_DOMAIN_NAME", ParameterType.UrlSegment);
        request.Resource = "/domains/{domain}/webhooks/clicked";
        return client.Execute (request);
    }

}
```

```
import (
    "context"
    "github.com/mailgun/mailgun-go/v3"
    "time"
)

func GetWebhook(domain, apiKey string) (string, error) {
    mg := mailgun.NewMailgun(domain, apiKey)

    ctx, cancel := context.WithTimeout(context.Background(), time.Second*30)
    defer cancel()

    return mg.GetWebhook(ctx, "clicked")
```

(continues on next page)

(continued from previous page)

```
}
```

```
var DOMAIN = 'YOUR_DOMAIN_NAME';
var mailgun = require('mailgun-js')({ apiKey: "YOUR_API_KEY", domain: DOMAIN });

mailgun.get(`/domains/${DOMAIN}/webhooks/clicked`, function (error, body) {
  console.log(body);
});
```

Sample response:

```
{
  "webhook": {
    "urls": [ "https://your_domain.com/v1/clicked" ]
  }
}
```

Create a new webhook.

```
curl -s --user 'api:YOUR_API_KEY' \
   https://api.mailgun.net/v3/domains/YOUR_DOMAIN_NAME/webhooks \
   -F id='clicked' \
   -F url='https://your_domain.com/v1/clicked'
   -F url='https://your_domain.com/v2/clicked'
   -F url='https://your_partner_domain.com/v1/clicked'
```

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode addWebhook() throws UnirestException {

        HttpResponse <JsonNode> request = Unirest.post("https://api.mailgun.net/v3/
↪domains/" + YOUR_DOMAIN_NAME + "/webhooks")
            .basicAuth("api", API_KEY)
            .field("id","click")
            .field("url", "https://your_domain.com/v1/clicked")
            .field("url", "https://your_domain.com/v2/clicked")
            .field("url", "https://your_partner_domain.com/v1/clicked")
            .asJson();

        return request.getBody();
    }
}
```

```
# Include the Autoloader (see "Libraries" for install instructions)
require 'vendor/autoload.php';
use Mailgun\Mailgun;

# Instantiate the client.
```

(continues on next page)

(continued from previous page)

```
$mgClient = Mailgun::create('PRIVATE_API_KEY', 'https://API_HOSTNAME');
$domain   = 'YOUR_DOMAIN_NAME';
$webhook  = 'delivered';
$destination_url = 'https://my.webhook.url/delivered'

# Issue the call to the client.
$result = $mgClient->webhooks()->create($domain, $webhook, $destination_url);
```

```
def add_webhook():
    return requests.post(
        "https://api.mailgun.net/v3/domains/YOUR_DOMAIN_NAME/webhooks",
        auth=("api", "YOUR_API_KEY"),
        data={
          'id':'clicked',
          'url':[ 'https://your_domain.com/v1/clicked',
          'https://your_domain.com/v2/clicked',
          'https://your_partner_domain.com/v1/clicked'
          ]
        })
```

```
def add_webhook
  RestClient.post("https://api:YOUR_API_KEY"\
                  "@api.mailgun.net/v3/domains/YOUR_DOMAIN_NAME/webhooks",
                  :id => 'clicked',
                  :url => ['https://your_domain.com/v1/clicked',
                           'https://your_domain.com/v2/clicked',
                           'https://your_partner_domain.com/v1/clicked'])
end
```

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class AddWebhookChunk
{

    public static void Main (string[] args)
    {
        Console.WriteLine (AddWebhook ().Content.ToString ());
    }

    public static IRestResponse AddWebhook ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3/");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.Resource = "domains/YOUR_DOMAIN_NAME/webhooks";
        request.AddParameter ("id", "clicked");
        request.AddParameter ("url", "https://your_domain.com/v1/clicked");
        request.AddParameter ("url", "https://your_domain.com/v2/clicked");
        request.AddParameter ("url", "https://your_partner_domain.com/v1/clicked");
```

(continues on next page)

(continued from previous page)

```
        request.Method = Method.POST;
        return client.Execute(request);
    }


}
```

```
import (
    "context"
    "github.com/mailgun/mailgun-go/v3"
    "time"
)

func CreateWebhook(domain, apiKey string) error {
    mg := mailgun.NewMailgun(domain, apiKey)

    ctx, cancel := context.WithTimeout(context.Background(), time.Second*30)
    defer cancel()

    return mg.CreateWebhook(ctx, "clicked", []string{"https://your_domain.com/v1/
↪clicked"})
}
```

```
var DOMAIN = 'YOUR_DOMAIN_NAME';
var mailgun = require('mailgun-js')({ apiKey: "YOUR_API_KEY", domain: DOMAIN });
var urls = ['https://your_domain.com/v1/clicked', 'https://your_domain.com/v2/clicked
↪', 'https://your_parner_domain.com/v1/clicked']

mailgun.post(`/domains/${DOMAIN}/webhooks`, {"id": 'clicked', "url": urls}, function␣
↪(error, body) {
  console.log(body);
});
```

Sample response:

```
{
  "message": "Webhook has been created",
  "webhook": {
    "urls": [
              "https://your_domain.com/v1/clicked",
              "https://your_domain.com/v2/clicked",
              "https://your_partner_domain.com/v1/clicked"
            ]
  }
}
```

Update an existing webhook.

```
curl -s --user 'api:YOUR_API_KEY' -X PUT \
    https://api.mailgun.net/v3/domains/YOUR_DOMAIN_NAME/webhooks/clicked \
    -F url='https://your_domain,com/v1/clicked'
```

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;
```

(continues on next page)

(continued from previous page)

```
public class MGSample {

    // ...

    public static JsonNode updateWebhook() throws UnirestException {

        HttpResponse <JsonNode> request = Unirest.put("https://api.mailgun.net/v3/
↪domains/" + YOUR_DOMAIN_NAME + "/webhooks/clicked")
            .basicAuth("api", API_KEY)
            .field("url", "https://your_domain.com/clicked")
            .asJson();

        return request.getBody();
    }
}
```

```
# Include the Autoloader (see "Libraries" for install instructions)
require 'vendor/autoload.php';
use Mailgun\Mailgun;

# Instantiate the client.
$mgClient = Mailgun::create('PRIVATE_API_KEY', 'https://API_HOSTNAME');
$domain = 'YOUR_DOMAIN_NAME';
$webhook = 'delivered';
$destination_url = 'https://my.webhook.url/delivered'

# Issue the call to the client.
$result = $mgClient->webhooks()->update($domain, $webhook, $destination_url);
```

```
def update_webhook():
    return requests.put(
        ("https://api.mailgun.net/v3/domains/YOUR_DOMAIN_NAME/webhooks/clicked"),
        auth=('api', 'YOUR_API_KEY'),
        data={'url': 'https://your_domain.com/clicked'})
```

```
def update_webhook
  RestClient.put("https://api:YOUR_API_KEY" \
                 "@api.mailgun.net/v3/domains/YOUR_DOMAIN_NAME/webhooks/clicked",
                 :url => 'https://your_domain.com/clicked')
end
```

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class UpdateWebhookChunk
{

    public static void Main (string[] args)
    {
        Console.WriteLine (UpdateWebhook ().Content.ToString ());
    }
```

(continues on next page)

(continued from previous page)

```
    public static IRestResponse UpdateWebhook()
    {
        RestClient client = new RestClient();
        client.BaseUrl = new Uri("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator("api",
                                       "YOUR_API_KEY");
        RestRequest request = new RestRequest();
        request.Resource = "/domains/YOUR_DOMAIN_NAME/webhooks/clicked";
        request.AddParameter("url", "https://your_domain.com/clicked");
        request.Method = Method.PUT;
        return client.Execute(request);
    }

}
```

```
import (
    "context"
    "github.com/mailgun/mailgun-go/v3"
    "time"
)

func UpdateWebhook(domain, apiKey string) error {
    mg := mailgun.NewMailgun(domain, apiKey)

    ctx, cancel := context.WithTimeout(context.Background(), time.Second*30)
    defer cancel()

    return mg.UpdateWebhook(ctx, "clicked", []string{"https://your_domain.com/clicked
↪"})
}
```

```
var DOMAIN = 'YOUR_DOMAIN_NAME';
var mailgun = require('mailgun-js')({ apiKey: "YOUR_API_KEY", domain: DOMAIN });

mailgun.put(`/domain/${DOMAIN}/webhooks/clicked`, {"url": 'https://your_domain.com/v1/
↪clicked'}, function (error, body) {
  console.log(body);
});
```

Sample response:

```
{
  "message": "Webhook has been updated",
  "webhook": {
    "urls": [ "https://your_domain.com/v1/clicked" ]
  }
}
```

Delete a webhook.

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

```
import com.mashape.unirest.http.HttpResponse;
```

(continues on next page)

(continued from previous page)

```
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode deleteWebhook() throws UnirestException {

        HttpResponse <JsonNode> request = Unirest.delete("https://api.mailgun.net/v3/
↪domains/" + YOUR_DOMAIN_NAME + "/webhooks/clicked")
            .basicAuth("api", API_KEY)
            .asJson();

        return request.getBody();
    }
}
```

```
# Include the Autoloader (see "Libraries" for install instructions)
require 'vendor/autoload.php';
use Mailgun\Mailgun;

# Instantiate the client.
$mgClient = Mailgun::create('PRIVATE_API_KEY', 'https://API_HOSTNAME');
$domain   = 'YOUR_DOMAIN_NAME';
$webhook  = 'delivered';

# Issue the call to the client.
$result = $mgClient->webhooks()->delete($domain, $webhook);
```

```
def delete_domain():
    return requests.delete(
        "https://api.mailgun.net/v3/domains/YOUR_DOMAIN_NAME/webhooks/clicked",
        auth=("api", "YOUR_API_KEY"))
```

```
def delete_domain
  RestClient.delete "https://api:YOUR_API_KEY"\
  "@api.mailgun.net/v3/domains/YOUR_DOMAIN_NAME/webhooks/clicked"
end
```

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class DeleteWebhookChunk
{

    public static void Main (string[] args)
    {
        Console.WriteLine (DeleteWebhook ().Content.ToString ());
    }

    public static IRestResponse DeleteWebhook ()
```

(continues on next page)

(continued from previous page)

```
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.Resource = "/domains/{name}/webhooks/clicked";
        request.AddUrlSegment ("name", "YOUR_DOMAIN_NAME");
        request.Method = Method.DELETE;
        return client.Execute (request);
    }

}
```

```
import (
    "context"
    "github.com/mailgun/mailgun-go/v3"
    "time"
)

func DeleteWebhook(domain, apiKey string) error {
    mg := mailgun.NewMailgun(domain, apiKey)

    ctx, cancel := context.WithTimeout(context.Background(), time.Second*30)
    defer cancel()

    return mg.DeleteWebhook(ctx, "clicked")
}
```

```
var DOMAIN = 'YOUR_DOMAIN_NAME';
var mailgun = require('mailgun-js')({ apiKey: "YOUR_API_KEY", domain: DOMAIN });

mailgun.delete(`/domains/${DOMAIN}/webhooks/clicked`, function (error, body) {
  console.log(body);
});
```

Sample response:

```
{
  "message": "Webhook has been deleted",
  "webhook": {
    "urls": [
              "https://your_domain.com/v1/clicked",
              "https://your_domain.com/v2/clicked",
              "https://your_partner_domain.com/v1/clicked"
            ]
  }
}
```

Rate Limit Response:

```
{
        "retry-seconds": 60
}
```

4.12 Mailing Lists

You can programmatically create mailing lists using Mailgun Mailing List API.

The Mailing List API endpoint is available at:

```
v3/lists
```

A mailing list is a group of members (recipients) which itself has an email address, like developers@mailgun.net. This address becomes an ID for this mailing list.

When you send a message to developers@mailgun.net, all members of the list will receive a copy of it.

```
GET /lists/pages
```

Paginate over mailing lists under your account

Parameter	Description
limit	Maximum number of records to return *(optional: 100 by default)*

```
GET /lists/<address>
```

Returns a single mailing list by a given address.

```
POST /lists
```

Creates a new mailing list.

Parameter	Description
address	A valid email address for the mailing list, e.g. `developers@mailgun.net`, or `Developers <devs@mg.net>`
name	Mailing list name, e.g. `Developers` *(optional)*
description	A description *(optional)*
access_level	List access level, one of: readonly (default), members, everyone
reply_preference	Set where replies should go: `list` (default) | `sender` *(optional)*

```
PUT /lists/<address>
```

Update mailing list properties, such as address, description or name

Parameter	Description
address	New mailing list address, e.g. `devs@mg.net` *(optional)*
name	New name, e.g. `My newsletter` *(optional)*
description	Description string *(optional)*
access_level	List access level, one of: readonly (default), members, everyone
reply_preference	Set where replies should go: `list` (default) | `sender` *(optional)*

```
DELETE /lists/<address>
```

Deletes a mailing list.

```
POST /lists/<address>/validate
```

Validate all the members of the mailing list.

```
GET /lists/<address>/validate
```

Retrieve current status of the mailing list validation job.

```
DELETE /lists/<address>/validate
```

Cancel an active mailing list validation job.

```
GET /lists/<address>/members/pages
```

Paginate over list members in the given mailing list

Parameter	Description
subscribed	`yes` to lists subscribed, `no` for unsubscribed. list all if not set
limit	Maximum number of records to return *(optional: 100 by default)*

```
GET /lists/<address>/members/<member_address>
```

Retrieves a mailing list member.

```
POST /lists/<address>/members
```

Adds a member to the mailing list.

Parameter	Description
address	Valid email address specification, e.g. `Alice <alice@example.com>` or just `alice@example.com`
name	Optional member name
vars	JSON-encoded dictionary string with arbitrary parameters, e.g. `{"gender":"female","age":27}`
subscribed	`yes` to add as subscribed *(default)*, `no` as unsubscribed
upsert	**`yes` to update member if present, `no` to** raise error in case of a duplicate member *(default)*

```
PUT /lists/<address>/members/<member_address>
```

Updates a mailing list member with given properties. Won't touch the property if it's not passed in.

Parameter	Description
address	Valid email address specification, e.g. `Alice <alice@example.com>` or just `alice@example.com`
name	Recipient name, e.g. `Alice`
vars	JSON-encoded dictionary string with arbitrary parameters, e.g. `{"gender":"female", "age":27}`
subscribed	`no` to set unsubscribed, `yes` as subscribed

```
POST /lists/<address>/members.json
```

Adds multiple members, up to 1,000 per call, to a Mailing List.

Parameter	Description
members	JSON-encoded array. Elements can be either addresses, e.g. `["bob@example.com", "alice@example.com"]`, or JSON objects, e.g. `[{"address": "bob@example.com", "name": "Bob", "subscribed": false}, {"address": "alice@example.com", "name": "Alice"}]`. Custom variables can be provided, see examples.
upsert	`yes` to update existing members, `no` (default) to ignore duplicates

```
DELETE /lists/<address>/members/<member_address>
```

Delete a mailing list member.

4.12.1 Access Levels

Mailing lists have three different access levels. These levels define how users can interact with the list.

Access Level	Description
read-only	Only authenticated users can post to this list. It is used for mass announcements and newsletters. This is the default access level.
members	Subscribed members of the list can communicate with each other.
everyone	Everyone can post to this list. Recommended turning spam filtering on when using this mode.

4.12.2 Examples

Create a mailing list:

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

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode createMailingList() throws UnirestException {

        HttpResponse <JsonNode> request = Unirest.post("https://api.mailgun.net/v3/
↪lists")
```

(continues on next page)

(continued from previous page)

```
            .basicAuth("api", API_KEY)
            .field("address", "LIST@YOUR_DOMAIN_NAME")
            .field("description", "LIST_DESCRIPTION")
            .asJson();

        return request.getBody();
    }
}
```

```
# Include the Autoloader (see "Libraries" for install instructions)
require 'vendor/autoload.php';
use Mailgun\Mailgun;

# Instantiate the client.
$mgClient = Mailgun::create('PRIVATE_API_KEY', 'https://API_HOSTNAME');
$mailing_list      = 'LIST@YOUR_DOMAIN_NAME';
$list_name         = 'Mailgun Subscribers';
$list_description  = 'News and service updates';
$access_level      = 'readonly';

# Issue the call to the client.
$result = $mgClient->mailingList()->create($mailing_list, $list_name, $list_
↪description, $access_level);
```

```
def create_mailing_list():
    return requests.post(
        "https://api.mailgun.net/v3/lists",
        auth=('api', 'YOUR_API_KEY'),
        data={'address': 'LIST@YOUR_DOMAIN_NAME',
              'description': "Mailgun developers list"})
```

```
def create_mailing_list
  RestClient.post("https://api:YOUR_API_KEY" \
                  "@api.mailgun.net/v3/lists",
                  :address => 'LIST@YOUR_DOMAIN_NAME',
                  :description => "Mailgun developers list")
end
```

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class CreateMailingListChunk
{

    public static void Main (string[] args)
    {
        Console.WriteLine (CreateMailingList ().Content.ToString ());
    }

    public static IRestResponse CreateMailingList ()
    {

        RestClient client = new RestClient ();
```

(continues on next page)

(continued from previous page)

```
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.Resource = "lists";
        request.AddParameter ("address", "LIST@YOUR_DOMAIN_NAME");
        request.AddParameter ("description", "Mailgun developers list");
        request.Method = Method.POST;
        return client.Execute (request);
    }

}
```

```
import (
    "context"
    "github.com/mailgun/mailgun-go/v3"
    "time"
)

func CreateMailingList(domain, apiKey string) (mailgun.MailingList, error) {
    mg := mailgun.NewMailgun(domain, apiKey)

    ctx, cancel := context.WithTimeout(context.Background(), time.Second*30)
    defer cancel()

    return mg.CreateMailingList(ctx, mailgun.MailingList{
        Address:     "list@example.com",
        Name:        "dev",
        Description: "Mailgun developers list.",
        AccessLevel: mailgun.AccessLevelMembers,
    })
}
```

```
var DOMAIN = 'YOUR_DOMAIN_NAME';
var mailgun = require('mailgun-js')({ apiKey: "YOUR_API_KEY", domain: DOMAIN });

mailgun.post('/lists', {"address": `list_name@${DOMAIN}`, "description": "list_
↪description"}, function (error, body) {
  console.log(body);
});
```

Sample response:

```
{
  "message": "Mailing list has been created",
  "list": {
      "created_at": "Tue, 06 Mar 2012 05:44:45 GMT",
      "address": "dev@samples.mailgun.org",
      "members_count": 0,
      "description": "Mailgun developers list",
      "name": ""
  }
}
```

Get a page of mailing lists:

```
curl -s --user 'api:YOUR_API_KEY' -G \
    https://api.mailgun.net/v3/lists/pages
```

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode mailingLists() throws UnirestException {

        HttpResponse <JsonNode> request = Unirest.get("https://api.mailgun.net/v3/
↪lists/pages")
            .basicAuth("api", API_KEY)
            .asJson();

        return request.getBody();
    }
}
```

```
# Include the Autoloader (see "Libraries" for install instructions)
require 'vendor/autoload.php';
use Mailgun\Mailgun;

# Instantiate the client.
$mgClient = Mailgun::create('PRIVATE_API_KEY', 'https://API_HOSTNAME');

# Issue the call to the client.
$response = $mgClient->mailingList()->pages();
```

```
def list_members():
    return requests.get(
        "https://api.mailgun.net/v3/lists/pages",
        auth=('api', 'YOUR_API_KEY'))
```

```
def list_members
  RestClient.get("https://api:YOUR_API_KEY" \
                 "@api.mailgun.net/v3/lists/pages")
end
```

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class GetMailingListsChunk
{

    public static void Main (string[] args)
    {
        Console.WriteLine (GetMailingLists ().Content.ToString ());
    }
```

(continues on next page)

(continued from previous page)

```
    public static IRestResponse GetMailingLists()
    {
        RestClient client = new RestClient();
        client.BaseUrl = new Uri("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator("api",
                                       "YOUR_API_KEY");
        RestRequest request = new RestRequest();
        request.Resource = "lists/pages";
        return client.Execute(request);
    }

}
```

```
import (
    "context"
    "github.com/mailgun/mailgun-go/v3"
    "time"
)

func ListMailingLists(domain, apiKey string) ([]mailgun.MailingList, error) {
    mg := mailgun.NewMailgun(domain, apiKey)
    it := mg.ListMailingLists(nil)

    ctx, cancel := context.WithTimeout(context.Background(), time.Second*30)
    defer cancel()

    var page, result []mailgun.MailingList
    for it.Next(ctx, &page) {
        result = append(result, page...)
    }

    if it.Err() != nil {
        return nil, it.Err()
    }
    return result, nil
}
```

```
var DOMAIN = 'YOUR_DOMAIN_NAME';
var mailgun = require('mailgun-js')({ apiKey: "YOUR_API_KEY", domain: DOMAIN });

mailgun.get('/lists/pages', function (error, body) {
  console.log(body);
});
```

Sample response:

```
{
  "items": [
    {
      "access_level": "everyone",
      "address": "dev@samples.mailgun.org",
      "created_at": "Tue, 06 Mar 2012 05:44:45 GMT",
      "description": "Mailgun developers list",
      "members_count": 1,
```

(continues on next page)

(continued from previous page)

```
      "name": ""
    },
    {
      "access_level": "readonly",
      "address": "bar@example.com",
      "created_at": "Wed, 06 Mar 2013 11:39:51 GMT",
      "description": "",
      "members_count": 2,
      "name": ""
    }
  ],
  "paging": {
    "first": "https://url_to_next_page",
    "last": "https://url_to_last_page",
    "next": "https://url_to_next_page",
    "previous": "https://url_to_previous_page"
  }
}
```

Add a mailing list member:

```
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}'
```

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode addListMember() throws UnirestException {

        HttpResponse <JsonNode> request = Unirest.post("https://api.mailgun.net/v3/
↪lists/{list}@{domain}/members")
            .basicAuth("api", API_KEY)
            .field("subscribed", true)
            .field("address", "bob@example.com")
            .field("name", "Bob Bar")
            .field("description", "developer")
            .field("vars", "{\"age\": 26}")
            .asJson();

        return request.getBody();
    }
}
```

```
# Include the Autoloader (see "Libraries" for install instructions)
require 'vendor/autoload.php';
```

(continues on next page)

(continued from previous page)

```php
use Mailgun\Mailgun;

# Instantiate the client.
$mgClient = Mailgun::create('PRIVATE_API_KEY', 'https://API_HOSTNAME');
$mailing_list = 'LIST@YOUR_DOMAIN_NAME';
$address ='bob@example.com';
$name = 'Bob';
$vars = array("id" => "123456");

# Issue the call to the client.
$result = $mgClient->mailingList()->member()->create(
                                                $mailing_list,
                                                $address,
                                                $name,
                                                $vars);
```

```python
def add_list_member():
    return requests.post(
        "https://api.mailgun.net/v3/lists/LIST@YOUR_DOMAIN_NAME/members",
        auth=('api', 'YOUR_API_KEY'),
        data={'subscribed': True,
              'address': 'bar@example.com',
              'name': 'Bob Bar',
              'description': 'Developer',
              'vars': '{"age": 26}'})
```

```ruby
def add_list_member
  RestClient.post("https://api:YOUR_API_KEY" \
                  "@api.mailgun.net/v3/lists/LIST@YOUR_DOMAIN_NAME/members",
                  :subscribed => true,
                  :address => 'bar@example.com',
                  :name => 'Bob Bar',
                  :description => 'Developer',
                  :vars => '{"age": 26}')
end
```

```csharp
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class AddListMemberChunk
{

    public static void Main (string[] args)
    {
        Console.WriteLine (AddListMember ().Content.ToString ());
    }

    public static IRestResponse AddListMember ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
```

(continues on next page)

(continued from previous page)

```
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.Resource = "lists/{list}/members";
        request.AddParameter ("list", "LIST@YOUR_DOMAIN_NAME",
                              ParameterType.UrlSegment);
        request.AddParameter ("address", "bar@example.com");
        request.AddParameter ("subscribed", true);
        request.AddParameter ("name", "Bob Bar");
        request.AddParameter ("description", "Developer");
        request.AddParameter ("vars", "{\"age\": 26}");
        request.Method = Method.POST;
        return client.Execute (request);
    }

}
```

```
import (
    "context"
    "github.com/mailgun/mailgun-go/v3"
    "time"
)

func AddListMember(domain, apiKey string) error {
    mg := mailgun.NewMailgun(domain, apiKey)

    memberJoe := mailgun.Member{
        Address:    "joe@example.com",
        Name:       "Joe Example",
        Subscribed: mailgun.Subscribed,
    }

    ctx, cancel := context.WithTimeout(context.Background(), time.Second*30)
    defer cancel()

    return mg.CreateMember(ctx, true, "mailingList@example.com", memberJoe)
}
```

```
var DOMAIN = 'YOUR_DOMAIN_NAME';
var mailgun = require('mailgun-js')({ apiKey: "YOUR_API_KEY", domain: DOMAIN });

var list = mailgun.lists(`mylist@${DOMAIN}`);

var bob = {
  subscribed: true,
  address: 'bob@example.com',
  name: 'Bob Barr',
  vars: {age: 34}
};

list.members().create(bob, function (error, data) {
  console.log(data);
});
```

Sample response:

```
{
  "member": {
      "vars": {
          "age": 26
      },
      "name": "Bob Bar",
      "subscribed": true,
      "address": "bar@example.com"
  },
  "message": "Mailing list member has been created"
}
```

Add multiple mailing list members (limit 1,000 per call):

```
curl -s --user 'api:YOUR_API_KEY' \
   https://api.mailgun.net/v3/lists/LIST@YOUR_DOMAIN_NAME/members.json \
   -F upsert=true \
   -F members='[{"address": "Alice <alice@example.com>", "vars": {"age": 26}},{"name
↪": "Bob", "address": "bob@example.com", "vars": {"age": 34}}]'
```

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode addListMembers() throws UnirestException {

        HttpResponse <JsonNode> request = Unirest.post("https://api.mailgun.net/v3/
↪lists/{list}@{domain}/members.json")
            .basicAuth("api", API_KEY)
            .field("upsert", true)
            .field("members", "[{\"address\": \"Alice <alice@example.com>\", \"vars\
↪": {\"age\": 26}},{\"name\": \"Bob\", \"address\": \"bob@example.com\", \"vars\": {\
↪"age\": 34}}]")
            .asJson();

        return request.getBody();
    }
}
```

```
# Include the Autoloader (see "Libraries" for install instructions)
require 'vendor/autoload.php';
use Mailgun\Mailgun;

# Instantiate the client.
$mgClient = Mailgun::create('PRIVATE_API_KEY', 'https://API_HOSTNAME');
$mailing_list = 'LIST@YOUR_DOMAIN_NAME';
$members = array(
    array(
        'address'    => 'bob@example.com',
        'name'       => 'Bob',
        'vars'       => array("id" => "123456")
    )
```

(continues on next page)

(continued from previous page)

```
);

# Issue the call to the client.
$result = $mgClient->mailingList()->member()->createMultiple($mailing_list, $members);
```

```
def add_list_member():
    return requests.post(
        "https://api.mailgun.net/v3/lists/LIST@YOUR_DOMAIN_NAME/members.json",
        auth=('api', 'YOUR_API_KEY'),
        data={'upsert': True,
              'members': '[{"address": "Alice <alice@example.com>", "vars": {"age":␣
↪26}},{"name": "Bob", "address": "bob@example.com", "vars": {"age": 34}}]'})
```

```
def add_list_member
  RestClient.post("https://api:YOUR_API_KEY" \
                  "@api.mailgun.net/v3/lists/LIST@YOUR_DOMAIN_NAME/members.json",
                  :upsert => true,
                  :members => '[{"address": "Alice <alice@example.com>", "vars": {"age
↪": 26}},{"name": "Bob", "address": "bob@example.com", "vars": {"age": 34}}]')
end
```

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class AddListMembersChunk
{

    public static void Main (string[] args)
    {
        Console.WriteLine (AddListMembers ().Content.ToString ());
    }

    public static IRestResponse AddListMembers ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.Resource = "lists/{list}/members.json";
        request.AddParameter ("list", "LIST@YOUR_DOMAIN_NAME",
                              ParameterType.UrlSegment);
        request.AddParameter ("members",
                              "[{\"address\":\"Alice<alice@example.com>\",\"vars\":{\
↪"age\":26}},{\"name\":\"Bob\",\"address\":\"bob@example.com\",\"vars\":{\"age\":34}}
↪]");
        request.AddParameter ("upsert", true);
        request.Method = Method.POST;
        return client.Execute (request);
    }

}
```

```go
import (
    "context"
    "github.com/mailgun/mailgun-go/v3"
    "time"
)

func AddListMembers(domain, apiKey string) error {
    mg := mailgun.NewMailgun(domain, apiKey)

    ctx, cancel := context.WithTimeout(context.Background(), time.Second*30)
    defer cancel()

    return mg.CreateMemberList(ctx, nil, "mailgunList@example.com", []interface{}{
        mailgun.Member{
            Address:    "alice@example.com",
            Name:       "Alice's debugging account",
            Subscribed: mailgun.Unsubscribed,
        },
        mailgun.Member{
            Address:    "Bob Cool <bob@example.com>",
            Name:       "Bob's Cool Account",
            Subscribed: mailgun.Subscribed,
        },
        mailgun.Member{
            Address: "joe.hamradio@example.com",
            // Charlette is a ham radio packet BBS user.
            // We attach her packet BBS e-mail address as an arbitrary var here.
            Vars: map[string]interface{}{
                "packet-email": "KW9ABC @ BOGUS-4.#NCA.CA.USA.NOAM",
            },
        },
    })
}
```

```node
var DOMAIN = 'YOUR_DOMAIN_NAME';
var mailgun = require('mailgun-js')({ apiKey: "YOUR_API_KEY", domain: DOMAIN });

var members = [
  {
    address: 'Alice <alice@example.com>',
    vars: { age: 26 }
  },
  {
    name: 'Bob',
    address: 'bob@example.com',
    vars: { age: 34 }
  }
];

mailgun.lists(`mylist@${DOMAIN}`).members().add({ members: members, subscribed: true }
↪, function (error, body) {
  console.log(body);
});
```

Sample response:

```
{
  "message": "Mailing list has been updated",
  "list": {
    "members_count": 7,
    "description": "My updated test mailing list",
    "created_at": "Wed, 06 Mar 2013 11:39:51 GMT",
    "access_level": "readonly",
    "address": "dev@samples.mailgun.org",
    "name": "Test List Updated"
  }
}
```

You can also update an existing member:

```
curl -s --user 'api:YOUR_API_KEY' -X PUT \
    https://api.mailgun.net/v3/lists/LIST@YOUR_DOMAIN_NAME/members/bar@example.com \
    -F subscribed=False \
    -F name='Foo Bar'
```

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode updateMembers() throws UnirestException {

        HttpResponse <JsonNode> request = Unirest.put("https://api.mailgun.net/v3/
↪lists/LIST_NAME@YOUR_DOMAIN_NAME/members/alice@example.com")
            .basicAuth("api", API_KEY)
            .field("subscribed", false)
            .field("name", "Alice")
            .asJson();

        return request.getBody();
    }
}
```

```
# Include the Autoloader (see "Libraries" for install instructions)
require 'vendor/autoload.php';
use Mailgun\Mailgun;

# Instantiate the client.
$mgClient = Mailgun::create('PRIVATE_API_KEY', 'https://API_HOSTNAME');
$mailing_list = 'LIST@YOUR_DOMAIN_NAME';
$recipient    = 'bob@example.com';
$params = array(
    'name'       => 'Bob',
    'subscribed' => 'yes',
    'vars'       => '{"age": 30  "pet": "cat" }'
);

# Issue the call to the client.
$result = $mgClient->mailingList()->member()->update($mailing_list, $recipient,
↪$params);
```

(continues on next page)

(continued from previous page)

```
def update_member():
    return requests.put(
        ("https://api.mailgun.net/v3/lists/LIST@YOUR_DOMAIN_NAME/members"
         "/bar@example.com"),
        auth=('api', 'YOUR_API_KEY'),
        data={'subscribed': False,
              'name': 'Foo Bar'})
```

```
def update_member
  RestClient.put("https://api:YOUR_API_KEY" \
                 "@api.mailgun.net/v3/lists/LIST@YOUR_DOMAIN_NAME/members" \
                 "/bar@example.com",
                 :subscribed => false,
                 :name => 'Foo Bar')
end
```

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class UpdateListMemberChunk
{

    public static void Main (string[] args)
    {
        Console.WriteLine (UpdateListMember ().Content.ToString ());
    }

    public static IRestResponse UpdateListMember ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.Resource = "lists/{list}/members/{member}";
        request.AddParameter ("list", "LIST@YOUR_DOMAIN_NAME",
                              ParameterType.UrlSegment);
        request.AddParameter ("member", "bar@example.com",
                              ParameterType.UrlSegment);
        request.AddParameter ("subscribed", false);
        request.AddParameter ("name", "Foo Bar");
        request.Method = Method.PUT;
        return client.Execute (request);
    }

}
```

```
import (
    "context"
    "github.com/mailgun/mailgun-go/v3"
    "time"
```

(continues on next page)

(continued from previous page)

```
}

func UpdateMember(domain, apiKey string) error {
    mg := mailgun.NewMailgun(domain, apiKey)

    ctx, cancel := context.WithTimeout(context.Background(), time.Second*30)
    defer cancel()

    _, err := mg.UpdateMember(ctx, "bar@example.com", "list@example.com", mailgun.
↪Member{
        Name: "Foo Bar",
        Subscribed: mailgun.Unsubscribed,
    })
    return err
}
```

```
var DOMAIN = 'YOUR_DOMAIN_NAME';
var mailgun = require('mailgun-js')({ apiKey: "YOUR_API_KEY", domain: DOMAIN });

var list = mailgun.lists(`mylist@${DOMAIN}`);

list.members('bob@example.com').update({ "name": 'Bob Bar' }, function (error, data) {
  console.log(data);
});
```

Sample response:

```
{
  "member": {
      "vars": {
          "age": 26
      },
      "name": "Foo Bar",
      "subscribed": false,
      "address": "bar@example.com"
  },
  "message": "Mailing list member has been updated"
}
```

Listing members:

```
curl -s --user 'api:YOUR_API_KEY' -G \
    https://api.mailgun.net/v3/lists/LIST@YOUR_DOMAIN_NAME/members/pages
```

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode listMembers() throws UnirestException {

        HttpResponse <JsonNode> request = Unirest.get("https://api.mailgun.net/v3/
↪lists/{list_name}@{domain}/members/pages")
```

(continues on next page)

(continued from previous page)

```
            .basicAuth("api", API_KEY)
            .asJson();

        return request.getBody();
    }
}
```

```
# Include the Autoloader (see "Libraries" for install instructions)
require 'vendor/autoload.php';
use Mailgun\Mailgun;

# Instantiate the client.
$mgClient    = Mailgun::create('PRIVATE_API_KEY', 'https://API_HOSTNAME');
$mailing_list = 'LIST@YOUR_DOMAIN_NAME';

# Issue the call to the client.
$result =  $mgClient->mailingList()->member()->index($mailing_list);
```

```
def list_members():
    return requests.get(
        "https://api.mailgun.net/v3/lists/LIST@YOUR_DOMAIN_NAME/members/pages",
        auth=('api', 'YOUR_API_KEY'))
```

```
def list_members
  RestClient.get("https://api:YOUR_API_KEY" \
                 "@api.mailgun.net/v3/lists/LIST@YOUR_DOMAIN_NAME/members/pages")
end
```

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class GetListMembersChunk
{

    public static void Main (string[] args)
    {
        Console.WriteLine (GetListMembers ().Content.ToString ());
    }

    public static IRestResponse GetListMembers ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.Resource = "lists/{list}/members/pages";
        request.AddParameter ("list", "LIST@YOUR_DOMAIN_NAME",
                              ParameterType.UrlSegment);
        return client.Execute (request);
    }
```

(continues on next page)

(continued from previous page)

```
}
```

```
import (
    "context"
    "github.com/mailgun/mailgun-go/v3"
    "time"
)

func GetMembers(domain, apiKey string) ([]mailgun.Member, error) {
    mg := mailgun.NewMailgun(domain, apiKey)
    it := mg.ListMembers("list@example.com", nil)

    ctx, cancel := context.WithTimeout(context.Background(), time.Second*30)
    defer cancel()

    var page, result []mailgun.Member
    for it.Next(ctx, &page) {
        result = append(result, page...)
    }

    if it.Err() != nil {
        return nil, it.Err()
    }
    return result, nil
}
```

```
var DOMAIN = 'YOUR_DOMAIN_NAME';
var mailgun = require('mailgun-js')({ apiKey: "YOUR_API_KEY", domain: DOMAIN });

var list = mailgun.lists(`mylist@${DOMAIN}`);

list.members().list(function (err, members) {
  console.log(members);
});
```

Sample response:

```
{
  "items": [
      {
          "vars": {
              "age": 26
          },
          "name": "Foo Bar",
          "subscribed": false,
          "address": "bar@example.com"
      }
  ],
  "paging": {
    "first": "https://url_to_first_page",
    "last": "https://url_to_last_page",
    "next": "http://url_to_next_page",
    "previous": "http://url_to_previous_page"
  }
}
```

Remove a member:

```
curl -s --user 'api:YOUR_API_KEY' -X DELETE \
    https://api.mailgun.net/v3/lists/LIST@YOUR_DOMAIN_NAME/members/bar@example.com
```

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode removeMembers() throws UnirestException {

        HttpResponse <JsonNode> request = Unirest.delete("https://api.mailgun.net/v3/
↪lists/YoungJustice@example.com/members/karen@example.com")
            .basicAuth("api", API_KEY)
            .asJson();

        return request.getBody();
    }
}
```

```
# Include the Autoloader (see "Libraries" for install instructions)
require 'vendor/autoload.php';
use Mailgun\Mailgun;

# Instantiate the client.
$mgClient = Mailgun::create('PRIVATE_API_KEY', 'https://API_HOSTNAME');
$mailing_list = 'LIST@YOUR_DOMAIN_NAME';
$recipient    = 'bob@example.com';

# Issue the call to the client.
$result = $mgClient->mailingList()->member()->delete($mailing_list, $recipient);
```

```
def remove_member():
    return requests.delete(
        ("https://api.mailgun.net/v3/lists/LIST@YOUR_DOMAIN_NAME/members"
         "/bar@example.com"),
        auth=('api', 'YOUR_API_KEY'))
```

```
def remove_member
  RestClient.delete("https://api:YOUR_API_KEY" \
                    "@api.mailgun.net/v3/lists/LIST@YOUR_DOMAIN_NAME/members" \
                    "/bar@example.com")
end
```

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class RemoveListMemberChunk
{
```

(continues on next page)

(continued from previous page)

```
    public static void Main (string[] args)
    {
        Console.WriteLine (RemoveListMember ().Content.ToString ());
    }

    public static IRestResponse RemoveListMember ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.Resource = "lists/{list}/members/{member}";
        request.AddParameter ("list", "LIST@YOUR_DOMAIN_NAME",
                              ParameterType.UrlSegment);
        request.AddParameter ("member", "bar@example.com",
                              ParameterType.UrlSegment);
        request.Method = Method.DELETE;
        return client.Execute (request);
    }

}
```

```
import (
    "context"
    "github.com/mailgun/mailgun-go/v3"
    "time"
)

func DeleteListMember(domain, apiKey string) error {
    mg := mailgun.NewMailgun(domain, apiKey)

    ctx, cancel := context.WithTimeout(context.Background(), time.Second*30)
    defer cancel()

    return mg.DeleteMember(ctx, "joe@example.com", "list@example.com")
}
```

```
var DOMAIN = 'YOUR_DOMAIN_NAME';
var mailgun = require('mailgun-js')({ apiKey: "YOUR_API_KEY", domain: DOMAIN });

var list = mailgun.lists(`mylist@${DOMAIN}`);

list.members('bob@example.com').delete(function (err, body) {
  console.log(body);
});
```

Sample response:

```
{
  "member": {
      "address": "bar@example.com"
  },
  "message": "Mailing list member has been deleted"
}
```

Remove mailing list:

```
curl -s --user 'api:YOUR_API_KEY' -X DELETE \
    https://api.mailgun.net/v3/lists/LIST@YOUR_DOMAIN_NAME
```

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode removeMailingList() throws UnirestException {

        HttpResponse <JsonNode> request = Unirest.delete("https://api.mailgun.net/v3/
↪lists/YoungJustice@example.com")
            .basicAuth("api", API_KEY)
            .asJson();

        return request.getBody();
    }
}
```

```
# Include the Autoloader (see "Libraries" for install instructions)
require 'vendor/autoload.php';
use Mailgun\Mailgun;

# Instantiate the client.
$mgClient = Mailgun::create('PRIVATE_API_KEY', 'https://API_HOSTNAME');
$mailing_list = 'LIST@YOUR_DOMAIN_NAME';

# Issue the call to the client.
$result = $mgClient->mailingList()->delete($mailing_list);
```

```
def remove_list():
    return requests.delete(
        "https://api.mailgun.net/v3/lists/LIST@YOUR_DOMAIN_NAME",
        auth=('api', 'YOUR_API_KEY'))
```

```
def remove_list
  RestClient.delete("https://api:YOUR_API_KEY" \
                    "@api.mailgun.net/v3/lists/LIST@YOUR_DOMAIN_NAME")
end
```

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class RemoveMailingListChunk
{

    public static void Main (string[] args)
    {
```

(continues on next page)

(continued from previous page)

```
        Console.WriteLine (RemoveMailingList ().Content.ToString ());
    }

    public static IRestResponse RemoveMailingList ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.Resource = "lists/{list}";
        request.AddParameter ("list", "LIST@YOUR_DOMAIN_NAME",
                              ParameterType.UrlSegment);
        request.Method = Method.DELETE;
        return client.Execute (request);
    }

}
```

```
import (
    "context"
    "github.com/mailgun/mailgun-go/v3"
    "time"
)

func DeleteMailingList(domain, apiKey string) error {
    mg := mailgun.NewMailgun(domain, apiKey)

    ctx, cancel := context.WithTimeout(context.Background(), time.Second*30)
    defer cancel()

    return mg.DeleteMailingList(ctx, "list@example.com")
}
```

```
var DOMAIN = 'YOUR_DOMAIN_NAME';
var mailgun = require('mailgun-js')({ apiKey: "YOUR_API_KEY", domain: DOMAIN });

var list = mailgun.lists(`mylist@${DOMAIN}`);

list.delete(function (err, body) {
 console.log(body);
});
```

Sample response:

```
{
  "message": "Mailing list has been deleted",
  "address": "dev@samples.mailgun.org"
}
```

Run mailing list validation:

```
curl -s --user 'api:YOUR_API_KEY' \
    https://api.mailgun.net/v3/lists/LIST@YOUR_DOMAIN_NAME/validate \
```

```java
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode validateMailingList() throws UnirestException {

        HttpResponse <JsonNode> request = Unirest.post("https://api.mailgun.net/v3/
↪lists/{list}@{domain}/validate")
            .basicAuth("api", API_KEY)
            .asJson();

        return request.getBody();
    }
}
```

```php
# Currently, the PHP SDK does not support Mailing List validations.
# Consider using the following php curl function.
function upload_bulk_validation() {
  $ch = curl_init();

  curl_setopt($ch, CURLOPT_HTTPAUTH, CURLAUTH_BASIC);
  curl_setopt($ch, CURLOPT_USERPWD, 'api:PRIVATE_API_KEY');
  curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);

  curl_setopt($ch, CURLOPT_CUSTOMREQUEST, 'POST');
  curl_setopt($ch, CURLOPT_URL, 'https://api.mailgun.net/v3/lists/LIST@YOUR_DOMAIN_
↪NAME/validate');

  $result = curl_exec($ch);
  curl_close($ch);

  return $result;
}
```

```python
def validate_mailing_list():
    return requests.post(
        "https://api.mailgun.net/v3/lists/LIST@YOUR_DOMAIN_NAME/validate",
        auth=('api', 'YOUR_API_KEY'))
```

```ruby
def validate_mailing_list
  RestClient.post("https://api:YOUR_API_KEY" \
                  "@api.mailgun.net/v3/lists/LIST@YOUR_DOMAIN_NAME/validate")
end
```

```dotnet
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class ValidateMailingListChunk
{
```

(continues on next page)

(continued from previous page)

```
    public static void Main (string[] args)
    {
        Console.WriteLine (ValidateMailingList ().Content.ToString ());
    }

    public static IRestResponse ValidateMailingList ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.Resource = "lists/{list}/validate";
        request.AddParameter ("list", "LIST@YOUR_DOMAIN_NAME",
                              ParameterType.UrlSegment);
        request.Method = Method.POST;
        return client.Execute (request);
    }

}
```

Sample response:

```
{
    "id": "listname@yourdomain.com",
    "message": "The validation job was submitted."
}
```

Get mailing list validation status:

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

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode getMailingListValidation() throws UnirestException {

        HttpResponse<JsonNode> request = Unirest.post("https://api.mailgun.net/v3/
↪lists/{list}@{domain}/validate")
            .basicAuth("api", API_KEY)
            .asJson();

        return request.getBody();
    }
}
```

```
# Currently, the PHP SDK does not support Mailing List Validations.
# Consider using the following php curl function.
function get_mailing_list_validation() {
  $ch = curl_init();

  curl_setopt($ch, CURLOPT_HTTPAUTH, CURLAUTH_BASIC);
  curl_setopt($ch, CURLOPT_USERPWD, 'api:PRIVATE_API_KEY');
  curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);

  curl_setopt($ch, CURLOPT_CUSTOMREQUEST, 'GET');
  curl_setopt($ch, CURLOPT_URL, 'https://api.mailgun.net/v3/lists/LIST@YOUR_DOMAIN_
↪NAME/validate');

  $result = curl_exec($ch);
  curl_close($ch);

  return $result;
}
```

```
def get_mailing_list_validation_status():
    return requests.get(
        "https://api.mailgun.net/v3/lists/LIST@YOUR_DOMAIN_NAME/validate",
        auth=('api', 'YOUR_API_KEY'))
```

```
def get_mailing_list_validation_status
  RestClient.get("https://api:YOUR_API_KEY"\
                 "@api.mailgun.net/v3/lists/LIST@YOUR_DOMAIN_NAME/validate"\
                 {|response, request, result| response }
end
```

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class GetMailingListValidationChunk
{

    public static void Main (string[] args)
    {
        Console.WriteLine (GetMailingListValidation ().Content.ToString ());
    }

    public static IRestResponse GetMailingListValidation ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.AddParameter ("list", "LIST@YOUR_DOMAIN_NAME", ParameterType.
↪UrlSegment);
        request.Resource = "/lists/{list}/validate";
        return client.Execute (request);
    }
```

(continues on next page)

(continued from previous page)

```
}
```

Sample response:

```
{
  "created_at": "Tue, 26 Feb 2019 21:30:03 GMT",
  "download_url": {
    "csv": "<download_link>",
    "json": "<download_link>"
  },
  "id": "listname@mydomain.sandbox.mailgun.org",
  "quantity": 207665,
  "records_processed": 207665,
  "status": "uploaded",
  "summary": {
    "result": {
      "deliverable": 184199,
      "do_not_send": 5647,
      "undeliverable": 12116,
      "unknown": 5613
    },
    "risk": {
      "high": 17763,
      "low": 142547,
      "medium": 41652,
      "unknown": 5613
    }
  }
}
```

Field Explanation:

Parameter	Type	Description
created_at	string	Date/Time that the request was initiated
download_url	array	*csv* and *json* representation of the download link for the results of the list validation
id	string	list name given when the list was initially created
quantity	integer	number of total items in the list to be validated
records_processed	integer	de-duplicated total of validated email addresses
status	string	current state of the list validation request
summary	array	nested count results for *deliverable*, *do_not_send*, *undeliverable* and *unknown* statuses
risk	array	nested count results for *high*, *low*, *medium* or *unknown* risk assessment results

Cancel mailing list validation:

```
curl -s --user 'api:YOUR_API_KEY' -X DELETE \
    https://api.mailgun.net/v3/lists/LIST@YOUR_DOMAIN_NAME/members/bar@example.com
```

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
```

(continues on next page)

(continued from previous page)

```
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode cancelMailingListValidation() throws UnirestException {

        HttpResponse<JsonNode> request = Unirest.delete("https://api.mailgun.net/v3/
↪lists/YoungJustice@example.com/validate")
            .basicAuth("api", API_KEY)
            .asJson();

        return request.getBody();
    }
}
```

```
# Currently, the PHP SDK does not support Mailing List validations.
# Consider using the following php curl function.
function delete_mailing_list_validation() {
  $ch = curl_init();

  curl_setopt($ch, CURLOPT_HTTPAUTH, CURLAUTH_BASIC);
  curl_setopt($ch, CURLOPT_USERPWD, 'api:PRIVATE_API_KEY');
  curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);

  curl_setopt($ch, CURLOPT_CUSTOMREQUEST, 'DELETE');
  curl_setopt($ch, CURLOPT_URL, 'https://api.mailgun.net/v3/lists/LIST@YOUR_DOMAIN_
↪NAME/validate');

  $result = curl_exec($ch);
  curl_close($ch);

  return $result;
}
```

```
def cancel_mailing_list_validation():
    return requests.delete(
        ("https://api.mailgun.net/v3/lists/LIST@YOUR_DOMAIN_NAME/validate"),
        auth=('api', 'YOUR_API_KEY'))
```

```
def cancel_mailing_list_validation
  RestClient.delete("https://api:YOUR_API_KEY" \
                    "@api.mailgun.net/v3/lists/LIST@YOUR_DOMAIN_NAME/validate" \
                    "/bar@example.com")
end
```

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class DeleteMailingListValidationChunk
{
```

(continues on next page)

(continued from previous page)

```
    public static void Main (string[] args)
    {
        Console.WriteLine (CancelMailingListValidation ().Content.ToString ());
    }

    public static IRestResponse CancelMailingListValidation()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.Resource = "lists/{list}/validate}";
        request.AddParameter ("list", "LIST@YOUR_DOMAIN_NAME",
                              ParameterType.UrlSegment);
        request.Method = Method.DELETE;
        return client.Execute (request);
    }


}
```

Sample response:

```
{
    "message": "Validation job canceled."
}
```

4.13 Templates

This API allows you to store predefined templates and use them to send messages using *sending API*.

The Templates API endpoint is available at:

```
v3/<domain>/templates
```

The API has following limitations:

- 100 templates per domains
- 10 versions per template
- 100Kb max template size

4.13.1 Template API

Store new template

```
POST /<domain>/templates
```

This API stores a new template, storing its name, description and, optionally, the template content. If the content is provided a new version is automatically created and becomes the active version.

Parameter	Description
name	Name of the template being created. The name can contain alpha characters, digits and next symbols: .-_~
description	Description of the template being created
template	(Optional) Content of the template
tag	(Optional) Initial tag of the created version. If *template* parameter is provided and *tag* is missing default value *initial* is used. The tag can contain alpha characters, digits and next symbols: .-_~
engine	(Optional) The template engine to use to render the template. Valid only if template parameter is provided. Currently the API supports only one engine: **handlebars**
comment	(Optional) Version comment. Valid only if new version is being created (template parameter is provided)

Example of storing template metadata only:

```
curl -s --user 'api:YOUR_API_KEY' -X POST \
  https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/templates \
  -F name='template.name' \
  -F description='template description'
```

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode createTemplate() throws UnirestException {

        HttpResponse <JsonNode> request = Unirest.post("https://api.mailgun.net/v3/"␣
↪+ YOUR_DOMAIN_NAME + "/templates")
            .basicAuth("api", API_KEY)
            .field("name", "template.name")
            .field("description", "template description")
            .asJson();

        return request.getBody();
    }
}
```

```
# Currently, the PHP SDK does not support the Templates endpoint.
# Consider using the following php curl function.
function create_template() {
  $params = array(
    'name' => 'template.name',
    'description' => 'template description',
  );

  $ch = curl_init();
```

(continues on next page)

(continued from previous page)

```
  curl_setopt($ch, CURLOPT_HTTPAUTH, CURLAUTH_BASIC);
  curl_setopt($ch, CURLOPT_USERPWD, 'api:PRIVATE_API_KEY');
  curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);

  curl_setopt($ch, CURLOPT_CUSTOMREQUEST, 'POST');
  curl_setopt($ch, CURLOPT_URL, 'https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/templates
↪');
  curl_setopt($ch, CURLOPT_POSTFIELDS, $params);

  $result = curl_exec($ch);
  curl_close($ch);

  return $result;
}
```

```
def store_template():
    return requests.post(
        "https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/templates",
        auth=("api", "YOUR_API_KEY"),
        data={'name': 'template.name',
              'description': 'template description'})
```

```
def store_template
  RestClient.post "https://api:YOUR_API_KEY"\
  "@api.mailgun.net/v3/YOUR_DOMAIN_NAME/templates",
  :name=> 'template.name',
  :description: => 'template description'
end
```

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class StoreTemplatesChunk
{

    public static void Main (string[] args)
    {
        Console.WriteLine (StoreTemplate ().Content.ToString ());
    }

    public static IRestResponse StoreTemplate ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.Resource = "{domain}/templates";
        request.AddParameter ("domain", "YOUR_DOMAIN_NAME", ParameterType.UrlSegment);
        request.AddParameter ("name", "template.name");
        request.AddParameter ("description", "template description")
```

(continues on next page)

(continued from previous page)

```
        request.Method = Method.POST;
        return client.Execute (request);
    }

}
```

```
func CreateTemplate(domain, apiKey string) error {
    mg := mailgun.NewMailgun(domain, apiKey)

    ctx, cancel := context.WithTimeout(context.Background(), time.Second*30)
    defer cancel()

    return mg.CreateTemplate(ctx, &mailgun.Template{
        Name: "template.name",
        Version: mailgun.TemplateVersion{
            Template: `'<div class="entry"> <h1>{{.title}}</h1> <div class="body"> {{.
↪body}} </div> </div>'`,
            Engine:   mailgun.TemplateEngineGo,
            Tag:      "v1",
        },
    })
}
```

```
var DOMAIN = 'YOUR_DOMAIN_NAME';
var mailgun = require('mailgun-js')({ apiKey: "YOUR_API_KEY", domain: DOMAIN });

mailgun.post(`/${DOMAIN}/templates`, {"name" : "template.name",
                                      "description": "template description"},
                                      function (error, body) {
                                          console.log(body);
                                      });
```

Sample response:

```
{
  "template": {
      "createdAt": "Wed, 29 Aug 2018 23:31:13 UTC",
      "description": "template description",
      "name": "template.name",
  },
  "message": "template has been stored"
}
```

Example of storing template with a version:

```
curl -s --user 'api:YOUR_API_KEY' -X POST \
  https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/templates \
  -F name='template.name' \
  -F description='template description' \
  -F template='{{fname}} {{lname}}' \
  -F engine='handlebars'
  -F comment='version comment'
```

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
```

(continues on next page)

(continued from previous page)

```
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode storeVersion() throws UnirestException {

        HttpResponse <JsonNode> request = Unirest.post("https://api.mailgun.net/v3/"␣
↪+ YOUR_DOMAIN_NAME + "/templates")
            .basicAuth("api", API_KEY)
            .field("name", "template.name")
            .field("description", "template description")
            .field("template", "{{fname}} {{lname}}")
            .field("engine", "handlebars")
            .field("commnt", "version comment")
            .asJson();

        return request.getBody();
    }
}
```

```
# Currently, the PHP SDK does not support the Templates endpoint.
# Consider using the following php curl function.
function create_template_version() {
  $params = array(
    'name' => 'template.name',
    'description' => 'template description',
    'template' => '{{fname}} {{lname}}',
    'engine' => 'handlebars',
    'comment' => 'version comment'
  );

  $ch = curl_init();

  curl_setopt($ch, CURLOPT_HTTPAUTH, CURLAUTH_BASIC);
  curl_setopt($ch, CURLOPT_USERPWD, 'api:PRIVATE_API_KEY');
  curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);

  curl_setopt($ch, CURLOPT_CUSTOMREQUEST, 'POST');
  curl_setopt($ch, CURLOPT_URL, 'https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/templates
↪');
  curl_setopt($ch, CURLOPT_POSTFIELDS, $params);

  $result = curl_exec($ch);
  curl_close($ch);

  return $result;
}
```

```
def store_template():
    return requests.post(
        "https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/templates",
        auth=("api", "YOUR_API_KEY"),
        data={'name': 'template.name',
```

(continues on next page)

(continued from previous page)

```
            'description': 'template description',
            'template': '{{fname}} {{lname}}',
            'engine': 'handlebars',
            'comment': 'version comment'})
```

```
def store_template
  RestClient.post "https://api:YOUR_API_KEY"\
  "@api.mailgun.net/v3/YOUR_DOMAIN_NAME/templates",
  :name => 'template.name',
  :description => 'template description',
  :template => '{{fname}} {{lname}}',
  :engine => 'handlebars',
  :comment => 'version comment'
end
```

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class StoreTemplatesChunk
{

    public static void Main (string[] args)
    {
        Console.WriteLine (StoreTemplate ().Content.ToString ());
    }

    public static IRestResponse StoreTemplate ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.Resource = "{domain}/templates";
        request.AddParameter ("domain", "YOUR_DOMAIN_NAME", ParameterType.UrlSegment);
        request.AddParameter ("name", "template.name");
        request.AddParameter ("description", "template description");
        request.AddParameter ("template", "{{fname}} {{lname}}");
        request.AddParameter ("engine", "handlebars");
        request.AddParameter ("comment", "version comment");
        request.Method = Method.POST;
        return client.Execute (request);
    }

}
```

```
func AddTemplateVersion(domain, apiKey string) error {
    mg := mailgun.NewMailgun(domain, apiKey)

    ctx, cancel := context.WithTimeout(context.Background(), time.Second*30)
    defer cancel()
```

(continues on next page)

(continued from previous page)

```
    return mg.AddTemplateVersion(ctx, "template.name", &mailgun.TemplateVersion{
        Template: `'<div class="entry"> <h1>{{.title}}</h1> <div class="body"> {{.
↪body}} </div> </div>'`,
        Engine:   mailgun.TemplateEngineGo,
        Tag:      "v2",
        Active:   true,
    })
}
```

```
var DOMAIN = 'YOUR_DOMAIN_NAME';
var mailgun = require('mailgun-js')({ apiKey: "YOUR_API_KEY", domain: DOMAIN });

mailgun.post(`/${DOMAIN}/templates`, {"name" : "template.name",
                                      "description": "template description",
                                      "template": "{{fname}} {{lname}}",
                                      "engine": "handlebars"},
                                      function (error, body) {
                                              console.log(body);
                                      });
```

Sample response:

```
{
  "template": {
      "createdAt": "Wed, 29 Aug 2018 23:31:13 UTC",
      "description": "template description",
      "name": "template.name",
      "version": {
          "createdAt": "Wed, 29 Aug 2018 23:31:14 UTC",
          "engine": "handlebars",
          "tag": "initial",
          "comment": "version comment"
      }
  },
  "message": "template has been stored"
}
```

Get template

```
GET /<domain>/templates/<name>
```

Returns metadata information about a stored template specified in url. If the *active* flag is provided the content of active version of the template is returned.

Parameter	Description
active	(Optional) If this flag is set to **yes** the active version of the template is included into a response.

Example:

```
curl -s --user 'api:YOUR_API_KEY' \
  https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/templates/TEMPLATE_NAME
```

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode getTemplate() throws UnirestException {

        HttpResponse <JsonNode> request = Unirest.get("https://api.mailgun.net/v3/
↪YOUR_DOMAIN_NAME/templates/TEMPLATE_NAME")
            .basicAuth("api", API_KEY)
            .asJson();

        return request.getBody();
    }
}
```

```
# Currently, the PHP SDK does not support the Templates endpoint.
# Consider using the following php curl function.
function get_template() {

  $ch = curl_init();

  curl_setopt($ch, CURLOPT_HTTPAUTH, CURLAUTH_BASIC);
  curl_setopt($ch, CURLOPT_USERPWD, 'api:PRIVATE_API_KEY');
  curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);

  curl_setopt($ch, CURLOPT_CUSTOMREQUEST, 'GET');
  curl_setopt($ch, CURLOPT_URL, 'https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/
↪templates/TEMPLATE_NAME');

  $result = curl_exec($ch);
  curl_close($ch);

  return $result;
}
```

```
def get_template():
    return requests.get(
        "https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/templates/TEMPLATE_NAME",
        auth=("api", "YOUR_API_KEY"))
```

```
def get_template
  RestClient.
    get("https://api:YOUR_API_KEY"\
        "@api.mailgun.net/v3/YOUR_DOMAIN_NAME/templates/TEMPLATE_NAME") {|response,␣
↪request, result| response }
end
```

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;
```

(continues on next page)

(continued from previous page)

```
public class GetTemplatesChunk
{

    public static void Main (string[] args)
    {
        Console.WriteLine (GetTemplate ().Content.ToString ());
    }

    public static IRestResponse GetTemplate ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.Resource = "/{domain}/templates/{name}";
        request.AddUrlSegment ("domain", "YOUR_DOMAIN_NAME");
        request.AddUrlSegment ("name", "TEMPLATE_NAME");
        return client.Execute (request);
    }

}
```

```
func GetTemplate(domain, apiKey string) (mailgun.Template, error) {
    mg := mailgun.NewMailgun(domain, apiKey)

    ctx, cancel := context.WithTimeout(context.Background(), time.Second*30)
    defer cancel()

    return mg.GetTemplate(ctx, "my-template")
}
```

```
var DOMAIN = 'YOUR_DOMAIN_NAME';
var mailgun = require('mailgun-js')({ apiKey: "YOUR_API_KEY", domain: DOMAIN });

mailgun.get('/domains/${DOMAIN}/templates/TEMPLATE_NAME', function (error, body) {
  console.log(body);
});
```

Sample response:

```
{
  "template": {
      "createdAt": "Wed, 29 Aug 2018 23:31:13 UTC",
      "description": "template description",
      "name": "template.name"
  }
}
```

Example of retrieving active version of a template:

```
curl -s --user 'api:YOUR_API_KEY' -X GET \
  https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/templates/TEMPLATE_NAME \
  -F active='yes'
```

```java
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode getTemplate() throws UnirestException {

        HttpResponse <JsonNode> request = Unirest.get("https://api.mailgun.net/v3/
↪YOUR_DOMAIN_NAME/templates/TEMPLATE_NAME")
            .basicAuth("api", API_KEY)
            .queryString("active", "yes")
            .asJson();

        return request.getBody();
    }
}
```

```php
# Currently, the PHP SDK does not support the Templates endpoint.
# Consider using the following php curl function.
function get_active_template() {

  $params = array(
      'active' => 'yes'
  );

  $ch = curl_init();

  curl_setopt($ch, CURLOPT_HTTPAUTH, CURLAUTH_BASIC);
  curl_setopt($ch, CURLOPT_USERPWD, 'api:PRIVATE_API_KEY');
  curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);

  curl_setopt($ch, CURLOPT_CUSTOMREQUEST, 'GET');
  curl_setopt($ch, CURLOPT_URL, 'https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/
↪templates/TEMPLATE_NAME');
  curl_setopt($ch, CURLOPT_POSTFIELDS, $params);

  $result = curl_exec($ch);
  curl_close($ch);

  return $result;
}
```

```python
def get_template():
    return requests.get(
        "https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/templates/TEMPLATE_NAME",
        auth=("api", "YOUR_API_KEY"),
        params={"active": "yes"})
```

```ruby
def get_template
   RestClient.get "https://api:YOUR_API_KEY"\
   "@api.mailgun.net/v3/YOUR_DOMAIN_NAME/templates/TEMPLATE_NAME", :params => {
       :active => "yes"
```

(continues on next page)

(continued from previous page)

```
    }
end
```

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class GetTemplatesChunk
{

    public static void Main (string[] args)
    {
        Console.WriteLine (GetTemplate ().Content.ToString ());
    }

    public static IRestResponse GetTemplate ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.Resource = "/{domain}/templates/{name}";
        request.AddUrlSegment ("domain", "YOUR_DOMAIN_NAME");
        request.AddUrlSegment ("name", "TEMPLATE_NAME");
        request.AddParameter ("active", "yes");
        return client.Execute (request);
    }

}
```

```
func ListActiveTemplates(domain, apiKey string) ([]mailgun.Template, error) {
    mg := mailgun.NewMailgun(domain, apiKey)
    it := mg.ListTemplates(&mailgun.ListTemplateOptions{Active: true})

    ctx, cancel := context.WithTimeout(context.Background(), time.Second*30)
    defer cancel()

    var page, result []mailgun.Template
    for it.Next(ctx, &page) {
        result = append(result, page...)
    }

    if it.Err() != nil {
        return nil, it.Err()
    }
    return result, nil
}
```

```
var DOMAIN = 'YOUR_DOMAIN_NAME';
var mailgun = require('mailgun-js')({ apiKey: "YOUR_API_KEY", domain: DOMAIN });

mailgun.get('/${DOMAIN}/templates/TEMPLATE_NAME', {"active", "yes"}, function (error,␣
↪body) {
```

(continues on next page)

(continued from previous page)

```
  console.log(body);
});
```

Sample response:

```
{
  "template": {
      "createdAt": "Wed, 29 Aug 2018 23:31:13 UTC",
      "description": "template description",
      "name": "template.name",
      "version": {
          "createdAt": "Wed, 29 Aug 2018 23:31:15 UTC",
          "engine": "handlebars",
          "tag": "v0",
          "template": "{{fname}} {{lname}}",
          "comment": "version comment"
      }
  }
}
```

Update template

```
PUT /<domain>/templates/<name>
```

Update metadata information of a template specified in url

Parameter	Description
description	Updated description of the template

Example:

```
curl -s --user 'api:YOUR_API_KEY' -X PUT \
    https://api.mailgun.net/v3/domains/YOUR_DOMAIN_NAME/templates/TEMPLATE_NAME \
    -F description = 'new template description'
```

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode updateTemplate() throws UnirestException {

        HttpResponse<JsonNode> request = Unirest.put("https://api.mailgun.net/v3/" +␣
↪YOUR_DOMAIN_NAME + "/templates" +TEMPLATE_NAME)
            .basicAuth("api", API_KEY)
            .field("description", "new template description")
            .asJson();

        return request.getBody();
```

(continues on next page)

(continued from previous page)

```
    }
}
```

```
# Currently, the PHP SDK does not support the Templates endpoint.
# Consider using the following php curl function.
function update_template() {
  $params = array(
    'description' => 'new template description'
  );

  $ch = curl_init();

  curl_setopt($ch, CURLOPT_HTTPAUTH, CURLAUTH_BASIC);
  curl_setopt($ch, CURLOPT_USERPWD, 'api:PRIVATE_API_KEY');
  curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);

  curl_setopt($ch, CURLOPT_CUSTOMREQUEST, 'PUT');
  curl_setopt($ch, CURLOPT_URL, 'https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/
↪templates/TEMPLATE_NAME');
  curl_setopt($ch, CURLOPT_POSTFIELDS, $params);

  $result = curl_exec($ch);
  curl_close($ch);

  return $result;
}
```

```
def update_template():
    return requests.put(
        ("https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/templates/TEMPLATE_NAME"),
        auth=('api', 'YOUR_API_KEY'),
        data={'description': 'new template description'})
```

```
def update_template
  RestClient.put("https://api:YOUR_API_KEY" \
                 "@api.mailgun.net/v3/YOUR_DOMAIN_NAME/templates/TEMPLATE_NAME",
                 :description => 'new template description')
end
```

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class UpdateTemplateChunk
{

    public static void Main (string[] args)
    {
        Console.WriteLine (UpdateTemplate ().Content.ToString ());
    }

    public static IRestResponse UpdateTemplate ()
    {
        RestClient client = new RestClient ();
```

(continues on next page)

(continued from previous page)

```
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.Resource = "/YOUR_DOMAIN_NAME/template/TEMPLATE_NAME";
        request.AddParameter ("description", "new template description");
        request.Method = Method.PUT;
        return client.Execute (request);
    }

}
```

```
import (
    "context"
    "github.com/mailgun/mailgun-go/v3"
    "time"
)

func UpdateTemplate(domain, apiKey string) error {
    mg := mailgun.NewMailgun(domain, apiKey)

    ctx, cancel := context.WithTimeout(context.Background(), time.Second*30)
    defer cancel()

    return mg.UpdateTemplate(ctx, &mailgun.Template{
        Name:        "TEMPLATE_NAME",
        Description: "Add a description to the template",
    })
}
```

```
var DOMAIN = 'YOUR_DOMAIN_NAME';
var mailgun = require('mailgun-js')({ apiKey: "YOUR_API_KEY", domain: DOMAIN });

mailgun.put(`/${DOMAIN}/templates/TEMPLATE_NAME`, {"description": "new template␣
↪descripton"},
                                                  function (error, body) {
                                                      console.log(body);
                                                  });
```

Sample response:

```
{
  "message": "template has been updated",
  "template": {
      "createdAt": "Wed, 29 Aug 2018 23:31:15 UTC",
      "description": "new template description",
      "name": "template.name"
  }
}
```

Delete template

```
DELETE /<domain>/templates/<name>
```

Delete a template specified in url. This method deletes all versions of the specified template

Example:

```
curl -s --user 'api:YOUR_API_KEY' -X DELETE \
    https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/templates/TEMPLATE_NAME
```

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode deleteTemplate() throws UnirestException {

        HttpResponse<JsonNode> request = Unirest.delete("https://api.mailgun.net/v3/
↪"+ YOUR_DOMAIN_NAME +"/templates/TEMPLATE_NAME")
            .basicAuth("api", API_KEY)
            .asJson();

        return request.getBody();
    }
}
```

```
# Currently, the PHP SDK does not support the Templates endpoint.
# Consider using the following php curl function.
function delete_template() {

  $ch = curl_init();

  curl_setopt($ch, CURLOPT_HTTPAUTH, CURLAUTH_BASIC);
  curl_setopt($ch, CURLOPT_USERPWD, 'api:PRIVATE_API_KEY');
  curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);

  curl_setopt($ch, CURLOPT_CUSTOMREQUEST, 'DELETE');
  curl_setopt($ch, CURLOPT_URL, 'https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/
↪templates/TEMPLATE_NAME');

  $result = curl_exec($ch);
  curl_close($ch);

  return $result;
}
```

```
def delete_template
  RestClient.delete "https://api:YOUR_API_KEY"\
  "@api.mailgun.net/v3/YOUR_DOMAIN_NAME/templates/TEMPLATE_NAME"
end
```

```
using System;
using System.IO;
```

(continues on next page)

(continued from previous page)

```
using RestSharp;
using RestSharp.Authenticators;

public class DeleteTemplate
{

    public static void Main (string[] args)
    {
        Console.WriteLine (DeleteTemplate ().Content.ToString ());
    }

    public static IRestResponse DeleteTemplate ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.AddParameter ("domain", "YOUR_DOMAIN_NAME", ParameterType.UrlSegment);
        request.Resource = "{domain}/templates/{name}";
        request.AddUrlSegment ("name", "TEMPLATE_NAME");
        request.Method = Method.DELETE;
        return client.Execute (request);
    }

}
```

```
func DeleteTemplate(domain, apiKey string) error {
    mg := mailgun.NewMailgun(domain, apiKey)

    ctx, cancel := context.WithTimeout(context.Background(), time.Second*30)
    defer cancel()

    return mg.DeleteTemplate(ctx, "TEMPLATE_NAME")
}
```

```
var DOMAIN = 'YOUR_DOMAIN_NAME';
var TEMPLATE_ID = 'YOUR_TEMPLATE_ID';
var mailgun = require('mailgun-js')({ apiKey: "YOUR_API_KEY", domain: DOMAIN });

mailgun.delete(`/${DOMAIN}/templates/${TEMPLATE_NAME}`, function (error, body) {
  console.log(body);
});
```

Sample response:

```
{
  "template": {
      "name": "template.name"
  },
  "message": "template has been deleted"
}
```

View all templates in domain

```
GET /<domain>/templates
```

Returns a list of stored templates for the domain

Parameter	Description
page	Name of a page to retrive. `first`, `last`, `next`, `prev`
limit	Number of records to retrive. Default value is 10
p	Pivot is used to retrieve records in chronological order

Example:

```
curl -s --user 'api:YOUR_API_KEY' -G \
  https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/templates \
  -d limit=10
```

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode listTemplates() throws UnirestException {

        HttpResponse <JsonNode> request = Unirest.get("https://api.mailgun.net/v3/" +␣
↪YOUR_DOMAIN_NAME + "/templates")
            .basicAuth("api", API_KEY)
            .queryString("limit","10")
            .asJson();

        return request.getBody();
    }
}
```

```
# Currently, the PHP SDK does not support the Templates endpoint.
# Consider using the following php curl function.
function get_templates() {

  $ch = curl_init();

  curl_setopt($ch, CURLOPT_HTTPAUTH, CURLAUTH_BASIC);
  curl_setopt($ch, CURLOPT_USERPWD, 'api:PRIVATE_API_KEY');
  curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);

  curl_setopt($ch, CURLOPT_CUSTOMREQUEST, 'GET');
  curl_setopt($ch, CURLOPT_URL, 'https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/templates
↪');

  $result = curl_exec($ch);
  curl_close($ch);
```

(continues on next page)

(continued from previous page)

```
  return $result;
}
```

```
def list_templates():
    return requests.get(
        "https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/templates",
        auth=("api", "YOUR_API_KEY"),
        params={"limit": 10})
```

```
def list_templates
  RestClient.get "https://api:YOUR_API_KEY"\
  "@api.mailgun.net/v3/YOUR_DOMAIN_NAME/templates", :params => {
    :limit => 10
  }
end
```

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class ListTemplatesChunk
{

    public static void Main (string[] args)
    {
        Console.WriteLine (ListTemplates ().Content.ToString ());
    }

    public static IRestResponse ListTemplates ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.AddParameter ("domain", "YOUR_DOMAIN_NAME", ParameterType.UrlSegment);
        request.AddParameter ("limit", 10);
        request.Resource = "/{domain}/templates";
        return client.Execute (request);
    }

}
```

```
import (
    "context"
    "github.com/mailgun/mailgun-go/v3"
    "time"
)

func ListTemplates(domain, apiKey string) ([]mailgun.Template, error) {
    mg := mailgun.NewMailgun(domain, apiKey)
    it := mg.ListTemplates(nil)
```

(continues on next page)

(continued from previous page)

```
    ctx, cancel := context.WithTimeout(context.Background(), time.Second*30)
    defer cancel()

    var page, result []mailgun.Template
    for it.Next(ctx, &page) {
        result = append(result, page...)
    }

    if it.Err() != nil {
        return nil, it.Err()
    }
    return result, nil
}
```

```
var DOMAIN = 'YOUR_DOMAIN_NAME';
var mailgun = require('mailgun-js')({ apiKey: "YOUR_API_KEY", domain: DOMAIN });

mailgun.get('/${DOMAIN}/templates', {"limit": 5}, function (error, body) {
  console.log(body);
});
```

Sample response:

```
{
  "items": [
      {
          "createdAt": "Wed, 29 Aug 2018 23:31:15 UTC",
          "description": "Template description",
          "name": "template.0",
      },
      {
          "createdAt": "Wed, 29 Aug 2018 23:31:18 UTC",
          "description": "Template description",
          "name": "template.1"
      },
      {
          "createdAt": "Wed, 29 Aug 2018 23:31:21 UTC",
          "description": "Template description",
          "name": "template.2"
      }
  ],
  "paging": {
      "first": "https://api.mailgin.net/v3/YOUR_DOMAIN_NAME/templates?limit=10",
      "last": "https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/templates?page=last&
↪limit=10",
      "next": "https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/templates?page=next&
↪p=template.2&limit=10",
      "prev": "https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/templates?page=prev&
↪p=template.0&limit=10"
  }
}
```

Delete all template in domain

```
DELETE /<domain>/templates
```

Delete all stored templates for the domain

Example:

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

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode deleteTemplates() throws UnirestException {

        HttpResponse<JsonNode> request = Unirest.delete("https://api.mailgun.net/v3/
↪"+ YOUR_DOMAIN_NAME +"/templates")
            .basicAuth("api", API_KEY)
            .asJson();

        return request.getBody();
    }
}
```

```
# Currently, the PHP SDK does not support the Templates endpoint.
# Consider using the following php curl function.
function delete_all_templates() {

  $ch = curl_init();

  curl_setopt($ch, CURLOPT_HTTPAUTH, CURLAUTH_BASIC);
  curl_setopt($ch, CURLOPT_USERPWD, 'api:PRIVATE_API_KEY');
  curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);

  curl_setopt($ch, CURLOPT_CUSTOMREQUEST, 'DELETE');
  curl_setopt($ch, CURLOPT_URL, 'https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/templates
↪');

  $result = curl_exec($ch);
  curl_close($ch);

  return $result;
}
```

```
def delete_templates():
    return requests.delete(
        "https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/templates",
        auth=("api", "YOUR_API_KEY"))
```

```
def delete_templates
  RestClient.delete "https://api:YOUR_API_KEY"\
  "@api.mailgun.net/v3/YOUR_DOMAIN_NAME/templates"
end
```

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class DeleteTemplatesChunk
{

    public static void Main (string[] args)
    {
        Console.WriteLine (DeleteTemplates ().Content.ToString ());
    }

    public static IRestResponse DeleteTemplates ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.AddParameter ("domain", "YOUR_DOMAIN_NAME", ParameterType.UrlSegment);
        request.Resource = "{domain}/templates";
        request.Method = Method.DELETE;
        return client.Execute (request);
    }

}
```

```
// Not implemented
```

```
var DOMAIN = 'YOUR_DOMAIN_NAME';
var mailgun = require('mailgun-js')({ apiKey: "YOUR_API_KEY", domain: DOMAIN });

mailgun.delete(`/${DOMAIN}/templates`, function (error, body) {
  console.log(body);
});
```

Sample response:

```
{
  "message": "templates have been deleted"
}
```

Create new version

```
POST /<domain>/templates/<template>/versions
```

Create a new version of the template. If the template doesn't contain any versions yet the first version becomes active

Parameter	Description
template	Content of the template
tag	Tag of the version is being created
engine	(Optional) Template engine. Only one engine are supported right now - `handlebars`.
comment	(Optional) The comment of the stored version
active	(Optional) If this flag is set to `yes` this version becomes active

Example:

```
curl -s --user 'api:YOUR_API_KEY' -X POST \
  https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/templates/TEMPLATE_NAME/versions \
  -F tag='v0' \
  -F template='{{fname}} {{lname}}' \
  -F engine='handlebars'
```

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

     // ...

    public static JsonNode storeTemplateVersion() throws UnirestException {

        HttpResponse <JsonNode> request = Unirest.post("https://api.mailgun.net/v3/"␣
↪+ YOUR_DOMAIN_NAME + "/templates/TEMPLATE_NAME/versions")
            .basicAuth("api", API_KEY)
            .field("tag", "v0")
            .field("template", "{{fname}} {{lname}}")
            .field("engine", "handlebars")
            .asJson();

        return request.getBody();
    }
}
```

```
# Currently, the PHP SDK does not support the Templates endpoint.
# Consider using the following php curl function.
function create_template_version() {
  $params = array(
    'tag' => 'v0',
    'template' => '{{fname}} {{lname}}',
    'engine' => 'handlebars'
  );

  $ch = curl_init();

  curl_setopt($ch, CURLOPT_HTTPAUTH, CURLAUTH_BASIC);
  curl_setopt($ch, CURLOPT_USERPWD, 'api:PRIVATE_API_KEY');
  curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);

  curl_setopt($ch, CURLOPT_CUSTOMREQUEST, 'POST');
  curl_setopt($ch, CURLOPT_URL, 'https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/
↪templates/TEMPLATE_NAME/versions');
```

(continues on next page)

(continued from previous page)

```
  curl_setopt($ch, CURLOPT_POSTFIELDS, $params);

  $result = curl_exec($ch);
  curl_close($ch);

  return $result;
}
```

```
def store_template_version():
    return requests.post(
        "https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/templates/TEMPLATE_NAME/versions
↪",
        auth=("api", "YOUR_API_KEY"),
        data={'tag': 'v0',
              'template': '{{fname}} {{lname}}',
              'engine': 'handlebars'})
```

```
def store_template_version
  RestClient.post "https://api:YOUR_API_KEY"\
  "@api.mailgun.net/v3/YOUR_DOMAIN_NAME/templates/TEMPLATE_NAME/versions",
  :tag => 'v0',
  :template => '{{fname}} {{lname}}',
  :engine => 'handlebars'
end
```

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class StoreTemplateVersionChunk
{

    public static void Main (string[] args)
    {
        Console.WriteLine (StoreTemplateVersion ().Content.ToString ());
    }

    public static IRestResponse StoreTemplateVersion ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.Resource = "{domain}/templates/{name}/versions";
        request.AddParameter ("domain", "YOUR_DOMAIN_NAME", ParameterType.UrlSegment);
        request.AddParameter ("name", "TEMPLATE_NAME", ParameterType.UrlSegment);
        request.AddParameter ("tag", "v0");
        request.AddParameter ("template", "{{fname}} {{lname}}");
        request.AddParameter ("engine", "handlebars");
        request.Method = Method.POST;
        return client.Execute (request);
    }
```

(continues on next page)

(continued from previous page)

```
}
```

```
import (
    "context"
    "github.com/mailgun/mailgun-go/v3"
    "time"
)

func AddTemplateVersion(domain, apiKey string) error {
    mg := mailgun.NewMailgun(domain, apiKey)

    ctx, cancel := context.WithTimeout(context.Background(), time.Second*30)
    defer cancel()

    return mg.AddTemplateVersion(ctx, "TEMPLATE_NAME", &mailgun.TemplateVersion{
        Template: `'<div class="entry"> <h1>{{.title}}</h1> <div class="body"> {{.
↪body}} </div> </div>'`,
        Engine:   mailgun.TemplateEngineGo,
        Tag:      "v2",
        Active:   true,
    })
}
```

```
var DOMAIN = 'YOUR_DOMAIN_NAME';
var mailgun = require('mailgun-js')({ apiKey: "YOUR_API_KEY", domain: DOMAIN });

mailgun.post(`/${DOMAIN}/templates/TEMPLATE_NAME/versions`, {"tag": "v0",
                                                             "template" : "{{fname}} {
↪{lname}}",
                                                             "engine": "handlebars"},
                                                               function (error, body)
↪{
                                                                    console.log(body);
                                                               });
```

Sample response:

```
{
  "template": {
      "createdAt": "Wed, 29 Aug 2018 23:31:11 UTC",
      "description": "template description",
      "name": "template.name",
      "version": {
          "createdAt": "Wed, 29 Aug 2018 23:31:21 UTC",
          "engine": "handlebars",
          "tag": "v1.0.0",
          "comment": "version comment"
      }
  },
  "message": "new version of the template has been stored"
}
```

Get version

```
GET /<domain>/templates/<name>/versions/<tag>
```

Retrieve information and content of specifed version of the template

Example:

```
curl -s --user 'api:YOUR_API_KEY' -G \
  https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/templates/TEMPLATE_NAME/versions/
↪VERSION_TAG
```

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode getTemplateVersion() throws UnirestException {

        HttpResponse <JsonNode> request = Unirest.get("https://api.mailgun.net/v3/" +␣
↪YOUR_DOMAIN_NAME + "/templates/TEMPLATE_NAME/versions/VERSION_TAG")
            .basicAuth("api", API_KEY)
            .asJson();

        return request.getBody();
    }
}
```

```
# Currently, the PHP SDK does not support the Templates endpoint.
# Consider using the following php curl function.
function get_template_version() {

  $ch = curl_init();

  curl_setopt($ch, CURLOPT_HTTPAUTH, CURLAUTH_BASIC);
  curl_setopt($ch, CURLOPT_USERPWD, 'api:PRIVATE_API_KEY');
  curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);

  curl_setopt($ch, CURLOPT_CUSTOMREQUEST, 'GET');
  curl_setopt($ch, CURLOPT_URL, 'https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/
↪templates/TEMPLATE_NAME/versions/VERSION_TAG');
  curl_setopt($ch, CURLOPT_POSTFIELDS, $params);

  $result = curl_exec($ch);
  curl_close($ch);

  return $result;
}
```

```
def get_template_version():
    return requests.get(
        "https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/templates/TEMPLATE_NAME/versions/
↪VERSION_TAG",
```

(continues on next page)

(continued from previous page)

```
        auth=("api", "YOUR_API_KEY"))
```

```
def get_template_version
  RestClient.get "https://api:YOUR_API_KEY"\
  "@api.mailgun.net/v3/YOUR_DOMAIN_NAME/templates/TEMPLATE_NAME/versions/VERSION_TAG"
end
```

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class GetTemplateVersionChunk
{

    public static void Main (string[] args)
    {
        Console.WriteLine (GetTemplateVersion ().Content.ToString ());
    }

    public static IRestResponse GetTemplateVersion ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.Resource = "/{domain}/templates/{name}/versions/{tag}";
        request.AddParameter ("domain", "YOUR_DOMAIN_NAME", ParameterType.UrlSegment);
        request.AddParameter ("name", "TEMPLATE_NAME", ParameterType.UrlSegment);
        request.AddParameter ("tag", "VERSION_TAG", ParameterType.UrlSegment);

        return client.Execute (request);
    }


}
```

```
import (
    "context"
    "github.com/mailgun/mailgun-go/v3"
    "time"
)

func GetTemplateVersion(domain, apiKey string) (mailgun.TemplateVersion, error) {
    mg := mailgun.NewMailgun(domain, apiKey)

    ctx, cancel := context.WithTimeout(context.Background(), time.Second*30)
    defer cancel()

    // Get the template version tagged as 'VERSION_TAG'
    return mg.GetTemplateVersion(ctx, "TEMPLATE_NAME", "VERSION_TAG")
}
```

```
var DOMAIN = 'YOUR_DOMAIN_NAME';
```

(continues on next page)

(continued from previous page)

```
var mailgun = require('mailgun-js')({ apiKey: "YOUR_API_KEY", domain: DOMAIN });

mailgun.get('/${DOMAIN}/templates/TEMPLATE_NAME/versions/VERSION_TAG', function␣
↪(error, body) {
  console.log(body);
});
```

Sample response:

```
{
  "template": {
      "createdAt": "Wed, 29 Aug 2018 23:31:11 UTC",
      "description": "template description",
      "name": "template.name",
      "version": {
          "createdAt": "Wed, 29 Aug 2018 23:31:21 UTC",
          "engine": "handlebars",
          "tag": "v1.1.0",
          "comment": "version comment",
          "template": "{{fname}} {{lname}}"
      }
  }
}
```

Update version

```
PUT /<domain>/templates/<name>/versions/<tag>
```

Update information or content of the specific version of the template

Parameter	Description
template	(Optional) The new content of the version
comment	(Optional) New comment for the provided version
active	(Optional) If this flag is set to `yes` this version becomes active

Example:

```
curl -s --user 'api:YOUR_API_KEY' -X PUT \
  https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/templates/TEMPLATE_NAME/versions/
↪VERSION_TAG \
  -F template='{{fname}} {{lname}}' \
  -F comment='Updated version comment' \
  -F active='yes'
```

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...
```

(continues on next page)

(continued from previous page)

```
    public static JsonNode UpdateVersion() throws UnirestException {

        HttpResponse <JsonNode> request = Unirest.put("https://api.mailgun.net/v3/" +␣
↪YOUR_DOMAIN_NAME + "/templates/TEMPLATE_NAME/versions/VERSION_TAG")
            .basicAuth("api", API_KEY)
            .field("template", "{{fname}} {{lname}}")
            .field("comment", "Updated version comment")
            .field("active", "yes")
            .asJson();

        return request.getBody();
    }
}
```

```
# Currently, the PHP SDK does not support the Templates endpoint.
# Consider using the following php curl function.
function update_template_version() {
  $params =   array(
    'template' => '{{fname}} {{lname}}',
    'comment' => 'Updated version comment',
    'active' => 'yes'
  );

  $ch = curl_init();

  curl_setopt($ch, CURLOPT_HTTPAUTH, CURLAUTH_BASIC);
  curl_setopt($ch, CURLOPT_USERPWD, 'api:PRIVATE_API_KEY');
  curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);

  curl_setopt($ch, CURLOPT_CUSTOMREQUEST, 'PUT');
  curl_setopt($ch, CURLOPT_URL, 'https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/
↪templates/TEMPLATE_NAME/versions/VERSION_TAG');
  curl_setopt($ch, CURLOPT_POSTFIELDS, $params);

  $result = curl_exec($ch);
  curl_close($ch);

  return $result;
}
```

```
def update_version():
    return requests.put(
        "https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/templates/TEMPLATE_NAME/versions/
↪VERSION_TAG",
        auth=("api", "YOUR_API_KEY"),
        data={'template': '{{fname}} {{lname}}',
              'comment': 'Updated version comment',
              'active': 'yes'})
```

```
def update_version
  RestClient.put "https://api:YOUR_API_KEY"\
  "@api.mailgun.net/v3/YOUR_DOMAIN_NAME/templates/TEMPLATE_NAME/versions/VERSION_TAG",
  :template => '{{fname}} {{lname}}',
  :comment => 'Updated version comment',
  :active => 'yes'
end
```

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class UpdateVersionChunk
{

    public static void Main (string[] args)
    {
        Console.WriteLine (UpdateVersion ().Content.ToString ());
    }

    public static IRestResponse UpdateVersion ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.Resource = "{domain}/templates/{name}/versions/{tag}";
        request.AddParameter ("domain", "YOUR_DOMAIN_NAME", ParameterType.UrlSegment);
        request.AddParameter ("name", "TEMPLATE_NAME", ParameterType.UrlSegment);
        request.AddParameter ("tag", "VERSION_TAG", ParameterType.UrlSegment);
        request.AddParameter ("template", "{{fname}} {{lname}}");
        request.AddParameter ("comment", "Updated version comment");
        request.AddParameter ("active", "yes");
        request.Method = Method.PUT;
        return client.Execute (request);
    }

}
```

```
import (
    "context"
    "github.com/mailgun/mailgun-go/v3"
    "time"
)

func UpdateTemplateVersion(domain, apiKey string) error {
    mg := mailgun.NewMailgun(domain, apiKey)

    ctx, cancel := context.WithTimeout(context.Background(), time.Second*30)
    defer cancel()

    return mg.UpdateTemplateVersion(ctx, "TEMPLATE_NAME", &mailgun.TemplateVersion{
        Comment: "Add a comment to the template and make it 'active'",
        Tag:     "VERSION_TAG",
        Active:  true,
    })
}
```

```
var DOMAIN = 'YOUR_DOMAIN_NAME';
var mailgun = require('mailgun-js')({ apiKey: "YOUR_API_KEY", domain: DOMAIN });

mailgun.put(`/${DOMAIN}/templates/TEMPLATE_NAME/versions/VERSION_TAG`,
```

(continues on next page)

(continued from previous page)

```
            {"template" : "{{fname}} {{lname}}",
             "comment": "Updated version comment",
             "active": "yes"},
             function (error, body) {
                 console.log(body);
             });
```

Sample response:

```
{
  "template": {
      "name": "template.name",
      "version": {
        "tag": "v1.2.0"
      }
   "message": "version has been udpated"
}
```

Delete version

```
DELETE /<domain>/templates/<template>/versions/<version>
```

Delete a specific version of the template

Example:

```
curl -s --user 'api:YOUR_API_KEY' -X DELETE -G \
   https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/templates/TEMPLATE_NAME/versions/
↪VERSION_TAG
```

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode deleteTemplateVersion() throws UnirestException {

        HttpResponse<JsonNode> request = Unirest.delete(
                                "https://api.mailgun.net/v3/" + YOUR_DOMAIN_NAME + "/
↪templates/TEMPLATE_NAME/versions/VERSION_TAG")
            .basicAuth("api", API_KEY)
            .asJson();

        return request.getBody();
    }
}
```

```
# Currently, the PHP SDK does not support the Templates endpoint.
# Consider using the following php curl function.
function delete_template_version() {
```

(continues on next page)

(continued from previous page)

```
  $ch = curl_init();

  curl_setopt($ch, CURLOPT_HTTPAUTH, CURLAUTH_BASIC);
  curl_setopt($ch, CURLOPT_USERPWD, 'api:PRIVATE_API_KEY');
  curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);

  curl_setopt($ch, CURLOPT_CUSTOMREQUEST, 'DELETE');
  curl_setopt($ch, CURLOPT_URL, 'https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/
↪templates/TEMPLATE_NAME/versions/VERSION_TAG');
  curl_setopt($ch, CURLOPT_POSTFIELDS, $params);

  $result = curl_exec($ch);
  curl_close($ch);

  return $result;
}
```

```
def delete_template_version():
    return requests.delete(
        "https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/templates/TEMPLATE_NAME/versions/
↪VERSION_TAG",
        auth=("api", "YOUR_API_KEY"))
```

```
def delete_template_version
  RestClient.get "https://api:YOUR_API_KEY"\
  "@api.mailgun.net/v3/YOUR_DOMAIN_NAME/templates/TEMPLATE_NAME/versions/VERSION_TAG"

end
```

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class DeleteTemplateVersionChunk
{

    public static void Main (string[] args)
    {
        Console.WriteLine (DeleteTemplateVersion ().Content.ToString ());
    }

    public static IRestResponse DeleteTemplateVersion ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();

        request.Resource = "/{domain}/templates/{name}/versions/{tag}";
        request.AddParameter ("domain", "YOUR_DOMAIN_NAME", ParameterType.UrlSegment);
        request.AddParameter ("name", "TEMPLATE_NAME", ParameterType.UrlSegment);
        request.AddParameter ("tag", "VERSION_TAG", ParameterType.UrlSegment);
```

(continues on next page)

(continued from previous page)

```
        request.Method = Method.DELETE;
        return client.Execute(request);
    }

}
```

```
import (
    "context"
    "github.com/mailgun/mailgun-go/v3"
    "time"
)

func DeleteTemplateVersion(domain, apiKey string) error {
    mg := mailgun.NewMailgun(domain, apiKey)

    ctx, cancel := context.WithTimeout(context.Background(), time.Second*30)
    defer cancel()

    // Delete the template version tagged as 'VERSION_TAG'
    return mg.DeleteTemplateVersion(ctx, "TEMPLATE_NAME", "VERSION_TAG")
}
```

```
var DOMAIN = 'YOUR_DOMAIN_NAME';
var mailgun = require('mailgun-js')({ apiKey: "YOUR_API_KEY", domain: DOMAIN });

mailgun.delete('/${DOMAIN}/templates/TEMPLATE_NAME/versions/VERSION_TAG', function␣
↪(error, body) {
  console.log(body);
});
```

Sample response:

```
{
  "template": {
      "name": "template.name",
      "version": {
        "tag": "v1.3.0"
      }
   "message": "version has been deleted"
}
```

View all versions in template

```
GET /<domain>/templates/<template>/versions
```

Returns a list of stored versions of the template

Parameter	Description
page	Name of a page to retrive. `first`, `last`, `next`, `prev`
limit	Number of records to retrive. Default value is 10
p	Pivot is used to retrieve records in chronological order

Example:

```
curl -s --user 'api:YOUR_API_KEY' -G \
  https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/templates/TEMPLATE_NAME/versions
  -d limit=10
```

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode listTemplateVersions() throws UnirestException {

        HttpResponse <JsonNode> request = Unirest.get("https://api.mailgun.net/v3/" +␣
↪YOUR_DOMAIN_NAME + "/templates/TEMPLATE_NAME/versions")
            .basicAuth("api", API_KEY)
            .asJson();

        return request.getBody();
    }
}
```

```
# Currently, the PHP SDK does not support the Templates endpoint.
# Consider using the following php curl function.
function get_all_versions() {
  $ch = curl_init();

  curl_setopt($ch, CURLOPT_HTTPAUTH, CURLAUTH_BASIC);
  curl_setopt($ch, CURLOPT_USERPWD, 'api:PRIVATE_API_KEY');
  curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);

  curl_setopt($ch, CURLOPT_CUSTOMREQUEST, 'GET');
  curl_setopt($ch, CURLOPT_URL, 'https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/
↪templates/TEMPLATE_NAME/versions');

  $result = curl_exec($ch);
  curl_close($ch);

  return $result;
}
```

```
def list_template_versions():
    return requests.get(
        "https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/templates/TEMPLATE_NAME/versions
↪",
        auth=("api", "YOUR_API_KEY"))
```

```
def list_template_version
  RestClient.get "https://api:YOUR_API_KEY"\
  "@api.mailgun.net/v3/YOUR_DOMAIN_NAME/templates/TEMPLATE_NAME/versions"
end
```

```
using System;
```

(continues on next page)

(continued from previous page)

```
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class ListTemplateVersionsChunk
{

    public static void Main (string[] args)
    {
        Console.WriteLine (ListTemplateVersions ().Content.ToString ());
    }

    public static IRestResponse GetTemplateVersion ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.AddParameter ("domain", "YOUR_DOMAIN_NAME", ParameterType.UrlSegment);
        request.AddParameter ("name", "TEMPLATE_NAME", ParameterType.UrlSegment);
        request.Resource = "/{domain}/templates/{name}/versions";
        return client.Execute (request);
    }

}
```

```
import (
    "context"
    "github.com/mailgun/mailgun-go/v3"
    "time"
)

func ListTemplateVersions(domain, apiKey string) ([]mailgun.TemplateVersion, error) {
    mg := mailgun.NewMailgun(domain, apiKey)
    it := mg.ListTemplateVersions("TEMPLATE_NAME", nil)

    ctx, cancel := context.WithTimeout(context.Background(), time.Second*30)
    defer cancel()

    var page, result []mailgun.TemplateVersion
    for it.Next(ctx, &page) {
        result = append(result, page...)
    }

    if it.Err() != nil {
        return nil, it.Err()
    }
    return result, nil
}
```

```
var DOMAIN = 'YOUR_DOMAIN_NAME';
var mailgun = require('mailgun-js')({ apiKey: "YOUR_API_KEY", domain: DOMAIN });

mailgun.get('/${DOMAIN}/templates/TEMPLATE_NAME/versions', function (error, body) {
```

(continues on next page)

(continued from previous page)

```
  console.log(body);
});
```

Sample response:

```
{
  "template": {
      "createdAt": "Wed, 29 Aug 2018 23:31:11 UTC",
      "description": "Template description",
      "name": "template.name",
      "versions": [
          {
              "createdAt": "Wed, 29 Aug 2018 23:31:21 UTC",
              "engine": "handlebars",
              "tag": "v0",
              "comment": "Version comment"
          },
          {
              "createdAt": "Wed, 29 Aug 2018 23:31:31 UTC",
              "engine": "handlebars",
              "tag": "v1",
              "comment": "Version comment"
          },
          {
              "createdAt": "Wed, 29 Aug 2018 23:31:41 UTC",
              "engine": "handlebars",
              "tag": "v2",
              "comment": "Version comment"
          }
      ]
  },
  "paging": {
      "first": "https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/templates/
↪z4ujt7CiEeik0RJbspqxaQ/versions?limit=10",
      "last": "https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/templates/
↪z4ujt7CiEeik0RJbspqxaQ/versions?page=last&limit=10",
      "next": "https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/templates/
↪z4ujt7CiEeik0RJbspqxaQ/versions?page=next&p=v2&limit=10",
      "prev": "https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/templates/
↪z4ujt7CiEeik0RJbspqxaQ/versions?page=prev&p=v0&limit=10"
  }
}
```

4.14 Email Validation

This API endpoint is an email address validation service.

We will validate the given address based on:

- Mailbox detection
- Syntax checks (RFC defined grammar)
- DNS validation
- Spell checks

- Email Service Provider (ESP) specific local-part grammar (if available).

The Email Validation API endpoint is available at:

```
v4/address/validate
```

Pricing details for Mailgun's email validation service can be found on our pricing page.

Mailgun's email validation service is intended to validate email addresses submitted through forms like newsletters, online registrations and shopping carts. Refer to our Acceptable Use Policy (AUP) for more information about how to use the service appropriately.

Note: A previous version of the API is described here api-email-validation

4.14.1 Single Validation

Note: This feature is now available via the EU API.

```
GET /v4/address/validate
```

Given an arbitrary address, validates address based off defined checks.

Parameter	Description
address	An email address to validate. (Maximum: 512 characters)

```
POST /v4/address/validate
```

Given an arbitrary address, validates address based off defined checks.

Form-Data	Description
address	An email address to validate. (Maximum: 512 characters)

Request Examples

Validate a single email address using the GET method.

```
curl --user 'api:PRIVATE_API_KEY' -G \
    https://api.mailgun.net/v4/address/validate \
    --data-urlencode address='foo@mailgun.net'
```

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...
```

(continues on next page)

(continued from previous page)

```
    public static JsonNode validateEmail() throws UnirestException {

        HttpResponse <JsonNode> request = Unirest.get("https://api.mailgun.net/v4/
↪address/validate")
            .basicAuth("api", PRIVATE_API_KEY)
            .queryString("address", "foo@mailgun.com")
            .asJson();

        return request.getBody();
    }
}
```

```
# Currently, the PHP SDK does not support the v4 Validations endpoint.
# Consider using the following php curl function.
function get_validate() {
  $params = array(
      "address" => "bob@example.com"
  );
  $ch = curl_init();

  curl_setopt($ch, CURLOPT_HTTPAUTH, CURLAUTH_BASIC);
  curl_setopt($ch, CURLOPT_USERPWD, 'api:PRIVATE_API_KEY');
  curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);

  curl_setopt($ch, CURLOPT_CUSTOMREQUEST, 'GET');
  curl_setopt($ch, CURLOPT_POSTFIELDS, $params);
  curl_setopt($ch, CURLOPT_URL, 'https://api.mailgun.net/v4/address/validate');
  $result = curl_exec($ch);
  curl_close($ch);

  return $result;
}
```

```
def get_validate():
    return requests.get(
        "https://api.mailgun.net/v4/address/validate",
        auth=("api", "PRIVATE_API_KEY"),
        params={"address": "foo@mailgun.net"})
```

```
def get_validate
  RestClient.get "https://api:PRIVATE_API_KEY"\
  "@api.mailgun.net/v4/address/validate",
  {params: {address: "foo@mailgun.net"}}
end
```

```
import (
   "encoding/json"
   "net/http"
)

type ValidationResponse struct {
   Address       string   `json:"address"`
   IsDisposable  bool     `json:"is_disposable_address"`
   IsRoleAddress bool     `json:"is_role_address"`
   Reason        []string `json:"reason"`
```

(continues on next page)

(continued from previous page)

```
    Result          string    `json:"result"`
    Risk            string    `json:"risk"`
}


func validateAddress(email string) (vr ValidationResponse, err error) {

    // creating HTTP request and returning response

    client := &http.Client{}
    req, _ := http.NewRequest("GET", "https://api.mailgun.net/v4/address/validate", ␣
↪nil)
    req.SetBasicAuth("api", apiKey)
    param := req.URL.Query()
    param.Add("address", email)
    req.URL.RawQuery = param.Encode()
    response, err := client.Do(req)

    if err != nil {
        return
    }

    // decoding into validation response struct
    err = json.NewDecoder(response.Body).Decode(&vr)
    return
}
```

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class GetValidateChunk
{

    public static void Main (string[] args)
    {
        Console.WriteLine (GetValidate ().Content.ToString ());
    }

    public static IRestResponse GetValidate ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v4");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "PRIVATE_API_KEY");
        RestRequest request = new RestRequest ();
        request.Resource = "/address/validate";
        request.AddParameter ("address", "foo@mailgun.net");
        return client.Execute (request);
    }

}
```

Validate a single email address using the POST method.

```
curl --user 'api:PRIVATE_API_KEY' -X POST \
    https://api.mailgun.net/v4/address/validate \
    -F address='foo@mailgun.net'
```

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode validateEmail() throws UnirestException {

        HttpResponse <JsonNode> request = Unirest.post("https://api.mailgun.net/v4/
↪address/validate")
            .basicAuth("api", PRIVATE_API_KEY)
            .field("address", "foo@mailgun.com")
            .asJson();

        return request.getBody();
    }
}
```

```
# Currently, the PHP SDK does not support the v4 Validations endpoint.
# Consider using the following php curl function.
function post_validate() {
  $params = array(
      "address" => "bob@example.com"
  );
  $ch = curl_init();

  curl_setopt($ch, CURLOPT_HTTPAUTH, CURLAUTH_BASIC);
  curl_setopt($ch, CURLOPT_USERPWD, 'api:PRIVATE_API_KEY');
  curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);

  curl_setopt($ch, CURLOPT_CUSTOMREQUEST, 'POST');
  curl_setopt($ch, CURLOPT_POSTFIELDS, $params);
  curl_setopt($ch, CURLOPT_URL, 'https://api.mailgun.net/v4/address/validate');
  $result = curl_exec($ch);
  curl_close($ch);

  return $result;
}
```

```
def post_validate():
    return requests.post(
        "https://api.mailgun.net/v4/address/validate",
        auth=("api", "PRIVATE_API_KEY"),
        data={"address": "foo@mailgun.net"})
```

```
def post_validate
  RestClient.post "https://api:PRIVATE_API_KEY"\
  "@api.mailgun.net/v4/address/validate",
  {:address => "foo@mailgun.net", :multipart => true}
```

(continues on next page)

(continued from previous page)

```
end
```

```
import (
        "bytes"
        "encoding/json"
        "mime/multipart"
        "net/http"
)

type ValidationResponse struct {
        Address       string   `json:"address"`
        IsDisposable  bool     `json:"is_disposable_address"`
        IsRoleAddress bool     `json:"is_role_address"`
        Reason        []string `json:"reason"`
        Result        string   `json:"result"`
        Risk          string   `json:"risk"`
}

func validateAddress(email string) (vr ValidationResponse, err error) {

        // creating HTTP request and returning response
        body := &bytes.Buffer{}
        writer := multipart.NewWriter(body)
        address, _ := writer.CreateFormField("address")
        _, _ = address.Write([]byte(email))
        writer.Close()

        client := &http.Client{}
        req, _ := http.NewRequest("POST", "https://api.mailgun.net/v4/address/validate
↪", body)
        req.Header.Set("Content-Type", writer.FormDataContentType())
        req.SetBasicAuth("api", "api_key_here")
        response, err := client.Do(req)

        if err != nil {
                return
    }

        // decoding into validation response struct
        err = json.NewDecoder(response.Body).Decode(&vr)
        return
}
```

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class GetValidateChunk
{

    public static void Main (string[] args)
    {
        Console.WriteLine (PostValidate ().Content.ToString ());
    }
```

(continues on next page)

(continued from previous page)

```
    public static IRestResponse PostValidate()
    {
        RestClient client = new RestClient();
        client.BaseUrl = new Uri("https://api.mailgun.net/v4");
        client.Authenticator =
            new HttpBasicAuthenticator("api",
                                       "PRIVATE_API_KEY");
        RestRequest request = new RestRequest();
        request.Resource = "/address/validate";;
        request.AddParameter("address", "foo@mailgun.net");
        request.Method = Method.POST
        return client.Execute(request);
    }
}
```

Example of a failed mailbox verification result.

```
{
    "address": "nonexistentemail@realdomain.com",
    "is_disposable_address": false,
    "is_role_address": false,
    "reason": [mailbox_does_not_exist],
    "result": "undeliverable",
    "risk": "high"
}
```

Example of successful mailbox verification result.

```
{
    "address": "existingemail@realdomain.com",
    "is_disposable_address": false,
    "is_role_address": false,
    "reason": [],
    "result": "deliverable",
    "risk": "low"
}
```

Warning: For advanced users only

The provider_lookup query parameter provides users with the control to allow or prevent Mailgun from reaching out to the mailbox provider.

```
GET /v4/address/validate?address=test123@test.com&provider_lookup=true
```

```
POST /v4/address/validate?provider_lookup=true
```

'true' (default state) - A provider lookup will be performed if Mailgun's internal analysis is insufficient.

'false' - A provider lookup will not be performed. If Mailgun does not have information on the recipient address, the API will return the following response:

```
{
    "address": "address@domain.com",
    "is_disposable_address": false,
```

(continues on next page)

(continued from previous page)

```
    "is_role_address": false,
    "reason": ["no_data"],
    "result": "unknown",
    "risk": "unknown"
}
```

Field Explanation

Parameter	Type	Description
address	string	Email address being validated
did_you_mean	string	(Optional) Null if nothing, however if a potential typo is made to the domain, the closest suggestion is provided
is_disposable_address	boolean	If the domain is in a list of disposable email addresses, this will be appropriately categorized
is_role_address	boolean	Checks the mailbox portion of the email if it matches a specific role type ('admin', 'sales', 'webmaster')
reason	array	List of potential reasons why a specific validation may be unsuccessful.
result	string	Either `deliverable`, `undeliverable`, `catch_all` or `unknown`. Please see the Result Types section below for details on each result type.
risk	string	`high`, `medium`, `low`, or `unknown` Depending on the evaluation of all aspects of the given email.
root_address	string	(Optional) If the address is an alias; this will contain the root email address with alias parts removed.

Reason Explanation

Reason	Description
unknown_provider	The MX provider is an unknown provider.
no_mx / No MX host found	The recipient domain does not have a valid MX host. *Note: this reason will be consolidated to only "no_mx" in the future.*
high_risk_domain	Information obtained about the domain indicates it is high risk to send email to.
subdomain_mailer	The recipient domain is identified to be a subdomain and is not on our exception list. Subdomains are considered to be high risk as many spammers and malicious actors utilize them.
immature_domain	The domain is newly created based on the WHOIS information.
tld_risk	The domain has a top-level-domain (TLD) that has been identified as high risk.
mailbox_does_not_exist	The mailbox is undeliverable or does not exist.
mailbox_is_disposable_address	The mailbox has been identified to be a disposable address. Disposable address are temporary, generally one time use, addresses.
mailbox_is_role_address	The mailbox is a role based address (ex. support@..., marketing@...).
catch_all	The validity of the recipient address cannot be determined as the provider accepts any and all email regardless of whether or not the recipient's mailbox exists.
long_term_disposable	The mailbox has been identified as a long term disposable address. Long term disposable addresses can be quickly and easily deactivated by users, but they will not expire without user intervention.

Result Types

Reason	Description
deliverable	The recipient address is considered to be valid and should accept email.
undeliverable	The recipient address is considered to be invalid and will result in a bounce if sent to.
do not send	The recipient address is considered to be highly risky and will negatively impact sending reputation if sent to.
catch_all	The validity of the recipient address cannot be determined as the provider accepts any and all email regardless of whether or not the recipient's mailbox exists.
unknown	The validity of the recipient address cannot be determined for a variety of potential reasons. Please refer to the associated 'reason' array returned in the response.

4.14.2 Bulk Validation

Note: Bulk Validation allows for the validation of a list of email addresses. Given a list name and an uploaded file of email addresses, a backend processing job will be run to validate the list. Once the validations have all been completed, the results will be provided with download links.

Note: It's important to upload as *multi-part/form-data* where the file is defined by *file*.

Currently only raw *csv* and *gzip* are supported.

The column header for emails needs to be either *email* or *email_address*

Warning: Lists must comply to either UTF-8 or ASCII encoding and not have a '@' in the name.

```
GET /v4/address/validate/bulk
```

Get list of all bulk validation jobs.

```
POST /v4/address/validate/bulk/<list_id>
```

Create a bulk validation job.

```
GET /v4/address/validate/bulk/<list_id>
```

Check the current status of a bulk validation job.

```
DELETE /v4/address/validate/bulk/<list_id>
```

Cancel current running bulk validation job.

Get the status of a bulk validation job:

```
curl -s --user 'api:YOUR_API_KEY' -G \
    https://api.mailgun.net/v4/address/validate/bulk/LIST_NAME
```

```java
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode getMailingListValidation() throws UnirestException {

        HttpResponse<JsonNode> request = Unirest.get("https://api.mailgun.net/v4/
↪address/validate/bulk/{list}")
            .basicAuth("api", API_KEY)
            .asJson();

        return request.getBody();
    }
}
```

```php
# Currently, the PHP SDK does not support the v4 Validations endpoint.
# Consider using the following php curl function.
function get_bulk_validation() {
  $ch = curl_init();

  curl_setopt($ch, CURLOPT_HTTPAUTH, CURLAUTH_BASIC);
  curl_setopt($ch, CURLOPT_USERPWD, 'api:PRIVATE_API_KEY');
  curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);

  curl_setopt($ch, CURLOPT_CUSTOMREQUEST, 'GET');
  curl_setopt($ch, CURLOPT_URL, 'https://api.mailgun.net/v4/address/validate/bulk/
↪LIST_NAME');
  $result = curl_exec($ch);
  curl_close($ch);

  return $result;
}
```

```python
def get_mailing_list_validation_status():
    return requests.get(
        "https://api.mailgun.net/v4/address/validate/bulk/LIST_NAME",
        auth=('api', 'YOUR_API_KEY'))
```

```ruby
def get_mailing_list_validation_status
  RestClient.get("https://api:YOUR_API_KEY"\
                 "@api.mailgun.net/v4/address/validate/bulk/LIST_NAME"\
                 ){|response, request, result| response }
end
```

```c#
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class GetBulkValidationChunk
{
```

(continues on next page)

(continued from previous page)

```
    public static void Main (string[] args)
    {
        Console.WriteLine (GetBulkValidation ().Content.ToString ());
    }

    public static IRestResponse GetBulkValidation ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v4");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.AddParameter ("list", "LIST_NAME", ParameterType.UrlSegment);
        request.Resource = "/address/validate/bulk/{list}";
        return client.Execute (request);
    }

}
```

Sample Response:

```
{
  "created_at": "Tue, 26 Feb 2019 21:30:03 GMT",
  "download_url": {
    "csv": "<download_link>",
    "json": "<download_link>"
  },
  "id": "bulk_validations_sandbox_mailgun_org",
  "quantity": 207665,
  "records_processed": 207665,
  "status": "uploaded",
  "summary": {
    "result": {
      "deliverable": 181854,
      "do_not_send": 5647,
      "undeliverable": 12116,
      "catch_all" : 2345,
      "unknown": 5613
    },
    "risk": {
      "high": 17763,
      "low": 142547,
      "medium": 41652,
      "unknown": 5613
    }
  }
}
```

Field Explanation:

Parameter	Type	Description
created_at	string	Date/Time that the request was initiated
download_url	array	*csv* and *json* representation of the download link for the results of the bulk validations
id	string	list_id name given when the list was initially created
quantity	integer	number of total items in the list to be validated
records_processed	integer	de-duplicated total of validated email addresses
status	string	current state of the list validation request. (`created`, `processing`, `completed`, `uploading`, `uploaded`, and `failed`)
summary	collection	summary of the validations in the list provided
result	array	nested results count. (`catch_all`, `deliverable`, `do_not_send`, `undeliverable`, and *unknown*)
risk	array	nested risk assessment count (`high`, `low`, `medium` or `unknown`)

Get a list of bulk validation jobs:

This request will return a list of validation jobs in descending order by time created.

```
curl -s --user 'api:YOUR_API_KEY' -G \
    https://api.mailgun.net/v4/address/validate/bulk
```

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode getMailingListValidation() throws UnirestException {

        HttpResponse<JsonNode> request = Unirest.get("https://api.mailgun.net/v4/
↪address/validate/bulk")
            .basicAuth("api", API_KEY)
            .queryString("limit", 2)
            .asJson();

        return request.getBody();
    }
}
```

```
# Currently, the PHP SDK does not support the v4 Validations endpoint.
# Consider using the following php curl function.
function get_bulk_validations() {
  $ch = curl_init();

  curl_setopt($ch, CURLOPT_HTTPAUTH, CURLAUTH_BASIC);
  curl_setopt($ch, CURLOPT_USERPWD, 'api:PRIVATE_API_KEY');
  curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);

  curl_setopt($ch, CURLOPT_CUSTOMREQUEST, 'GET');
```

(continues on next page)

(continued from previous page)

```
  curl_setopt($ch, CURLOPT_URL, 'https://api.mailgun.net/v4/address/validate/bulk?
↪limit=2');
  $result = curl_exec($ch);
  curl_close($ch);

  return $result;
}
```

```
def get_mailing_list_validation_status():
    return requests.get(
        "https://api.mailgun.net/v4/address/validate/bulk",
        auth=('api', 'YOUR_API_KEY'),
        params={"limit": 2})
```

```
def get_mailing_list_validation_status
  RestClient.get("https://api:YOUR_API_KEY"\
                 "@api.mailgun.net/v4/address/validate/bulk"\,
                 {params: {limit: 2}}
                 {|response, request, result| response }
end
```

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class GetBulkValidationsChunk
{

    public static void Main (string[] args)
    {
        Console.WriteLine (GetBulkValidations ().Content.ToString ());
    }

    public static IRestResponse GetBulkValidation ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v4");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.Resource = "/address/validate/bulk";
        request.AddParameter ("limit", 2);
        return client.Execute (request);
    }


}
```

Parameter	Type	Description
limit	integer	Number of entries to return. Default: 500.

Sample Response:

```
{
    "jobs":[
    {
        "created_at": "Tue, 26 Feb 2019 21:30:03 GMT",
        "download_url": {
            "csv": "<download_link>",
            "json": "<download_link>"
        }
        "id": "bulk_validations_sandbox2_mailgun_org",
        "quantity": 207665,
        "records_processed": 207665,
        "status": "uploaded",
        "summary": {
            "result": {
                "deliverable": 181854,
                "do_not_send": 5647,
                "undeliverable": 12116,
                "catch_all" : 2345,
                "unknown": 5613},
            "risk": {
                "high": 17763,
                "low": 142547,
                "medium": 41652,
                "unknown": 5613}
        }
    },
    {
        "created_at": "Tue, 23 Feb 2019 21:30:03 GMT",
        "download_url": {
            "csv": "<download_link>",
            "json": "<download_link>"
        }
        "id": "bulk_validations_sandbox_mailgun_org",
        "quantity": 207,
        "records_processed": 207,
        "status": "uploaded",
        "summary": {
            "result": {
                "deliverable": 181854,
                "do_not_send": 5647,
                "undeliverable": 12116,
                "catch_all" : 2345,
                "unknown": 5613},
            "risk": {
                "high": 17763,
                "low": 142547,
                "medium": 41652,
                "unknown": 5613}
        }
    }],
"total":3,
"paging": {
  "next":
      "https://url_to_next_page",
  "previous":
      "https://url_to_previous_page",
  "first":
```


(continues on next page)

(continued from previous page)

```
            "https://url_to_first_page",
        "last":
            "https://url_to_last_page"
    },
}
```

Results Fields Explanation:

Field	Description
Deliverable	The collection of validation jobs requested for.
Undeliverable	The total number of validation jobs.
Do Not Send	A collection of pagination links for traversing the validation jobs.
Catch All	The total number of domain associated with result is considered a catch_all domain

Create a bulk validation job:

```
curl -s --user 'api:YOUR_API_KEY' \
    https://api.mailgun.net/v4/address/validate/bulk/LIST_NAME \
    -F 'file=@/path/to/file' \
```

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode validateMailingList() throws UnirestException {

        HttpResponse<JsonNode> request = Unirest.post("https://api.mailgun.net/v4/
↪address/validate/bulk/LIST_NAME")
            .basicAuth("api", API_KEY)
            .field("file", new File("/path/to/file"))
            .asJson();

        return request.getBody();
    }
}
```

```
# Currently, the PHP SDK does not support the v4 Validations endpoint.
# Consider using the following php curl function.
function upload_bulk_validation() {
  $ch = curl_init();

  curl_setopt($ch, CURLOPT_HTTPAUTH, CURLAUTH_BASIC);
  curl_setopt($ch, CURLOPT_USERPWD, 'api:PRIVATE_API_KEY');
  curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);

  curl_setopt($ch, CURLOPT_CUSTOMREQUEST, 'POST');
  curl_setopt($ch, CURLOPT_URL, 'https://api.mailgun.net/v4/address/validate/bulk/
↪LIST_NAME');
  curl_setopt($ch, CURLOPT_POSTFIELDS, array(
```

(continues on next page)

(continued from previous page)

```
        'file'=> curl_file_create('subscribers.csv')
    );

    $result = curl_exec($ch);
    curl_close($ch);

    return $result;
}
```

```
def validate_mailing_list():
    return requests.post(
        "https://api.mailgun.net/v4/address/validate/bulk/LIST_NAME",
        files = {'file': open('/path/to/file','rb')},
        auth=('api', 'YOUR_API_KEY'))
```

```
def validate_mailing_list
  RestClient.post("https://api:YOUR_API_KEY" \
                  "@api.mailgun.net/v4/address/validate/bulk/LIST_NAME",
                  fields_hash.merge(:file => File.new('/path/to/file')))
end
```

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class BulkValidationChunk
{

    public static void Main (string[] args)
    {
        Console.WriteLine (BulkValidation ().Content.ToString ());
    }

    public static IRestResponse BulkValidation ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v4");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.Resource = "address/validate/bulk/{list}";
        request.AddParameter ("list", "LIST_NAME",
                              ParameterType.UrlSegment);
        request.Method = Method.POST;
        request.AddFile("file", @"/path/to/file");
        return client.Execute (request);
    }

}
```

Sample Response:

```
{
```

(continues on next page)

(continued from previous page)

```
 "id":"myemails"
 "message": "The validation job was submitted."
}
```

Cancel a bulk validation job:

```
curl -s --user 'api:YOUR_API_KEY' -X DELETE \
    https://api.mailgun.net/v4/address/validate/bulk/LIST_NAME
```

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode cancelMailingListValidation() throws UnirestException {
        url = "https://api.mailgun.net/v4/address/validate/bulk/LIST_NAME"
        HttpResponse <JsonNode> request = Unirest.delete(url)
            .basicAuth("api", API_KEY)
            .asJson();

        return request.getBody();
    }
}
```

```
# Currently, the PHP SDK does not support the v4 Validations endpoint.
# Consider using the following php curl function.
function delete_bulk_validation() {
  $ch = curl_init();

  curl_setopt($ch, CURLOPT_HTTPAUTH, CURLAUTH_BASIC);
  curl_setopt($ch, CURLOPT_USERPWD, 'api:PRIVATE_API_KEY');
  curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);

  curl_setopt($ch, CURLOPT_CUSTOMREQUEST, 'DELETE');
  curl_setopt($ch, CURLOPT_URL, 'https://api.mailgun.net/v4/address/validate/bulk/
↪LIST_NAME');
  $result = curl_exec($ch);
  curl_close($ch);

  return $result;
}
```

```
def cancel_bulk_validation():
    return requests.delete(
        ("https://api.mailgun.net/v4/address/validate/bulk/LIST_NAME"),
        auth=('api', 'YOUR_API_KEY'))
```

```
def cancel_bulk_validation
  RestClient.delete("https://api:YOUR_API_KEY" \
                    "@api.mailgun.net/v4/address/validate/bulk/LIST_NAME")
end
```

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class DeleteBulkValidationChunk
{

    public static void Main (string[] args)
    {
        Console.WriteLine (CancelBulkValidation ().Content.ToString ());
    }

    public static IRestResponse CancelBulkValidation()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v4");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.Resource = "address/validate/bulk/{list}";
        request.AddParameter ("list", "LIST_NAME",
                              ParameterType.UrlSegment);
        request.Method = Method.DELETE;
        return client.Execute (request);
    }


}
```

Sample Response:

```
{
 "message": "Validation job canceled."
}
```

4.14.3 Bulk Validation Preview

Note: Bulk validation preview performs a free analysis of a list of email addresses allowing you to make an informed decision to run a complete bulk validation or not. Given a preview name and an uploaded file of email addresses, a preliminary validation run will be preformed. The results of the preview will be, on average, an estimate of the deliverability and risk of the emails provide. This evaluation is based on a statistical sampling of the list provided.

Note: It's important to upload as *multi-part/form-data* where the file is defined by *file*.

Currently only raw *csv* and *gzip* are supported.

The column header for emails needs to be either *email* or *email_address*

Warning: Lists must comply to either UTF-8 or ASCII encoding and not have a '@' in the name.

```
GET /v4/address/validate/preview
```

Get list of all bulk validation previews.

```
POST /v4/address/validate/preview/<list_id>
```

Create a bulk validation preview.

```
GET /v4/address/validate/preview/<list_id>
```

Check the current status of a bulk validation preview.

```
DELETE /v4/address/validate/preview/<list_id>
```

Delete a bulk validation preview.

```
PUT /v4/address/validate/preview/<list_id>
```

Promote a bulk validation preview to a bulk validation job.

Get the results of a bulk validation preview:

```
curl -s --user 'api:YOUR_API_KEY' -G \
    https://api.mailgun.net/v4/address/validate/preview/LIST_NAME
```

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode getBulkPreview() throws UnirestException {

        HttpResponse<JsonNode> request = Unirest.get("https://api.mailgun.net/v4/
↪address/validate/preview/{list}")
            .basicAuth("api", API_KEY)
            .asJson();

        return request.getBody();
    }
}
```

```
# Currently, the PHP SDK does not support the v4 Validations endpoint.
# Consider using the following php curl function.
function get_bulk_preview() {
  $ch = curl_init();

  curl_setopt($ch, CURLOPT_HTTPAUTH, CURLAUTH_BASIC);
  curl_setopt($ch, CURLOPT_USERPWD, 'api:PRIVATE_API_KEY');
  curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);

  curl_setopt($ch, CURLOPT_CUSTOMREQUEST, 'GET');
  curl_setopt($ch, CURLOPT_URL, 'https://api.mailgun.net/v4/address/validate/preview/
↪LIST_NAME');
```

(continues on next page)

(continued from previous page)

```
  $result = curl_exec($ch);
  curl_close($ch);

  return $result;
}
```

```
def get_bulk_preview():
    return requests.get(
        "https://api.mailgun.net/v4/address/validate/preview/LIST_NAME",
        auth=('api', 'YOUR_API_KEY'))
```

```
def get_bulk_preview
  RestClient.get("https://api:YOUR_API_KEY"\
                 "@api.mailgun.net/v4/address/validate/preview/LIST_NAME"\
                 ){|response, request, result| response }
end
```

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class GetBulkValidationPreview
{

    public static void Main (string[] args)
    {
        Console.WriteLine (GetBulkPreview ().Content.ToString ());
    }

    public static IRestResponse GetBulkPreview ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v4");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.AddParameter ("list", "LIST_NAME", ParameterType.UrlSegment);
        request.Resource = "/address/validate/preview/{list}";
        return client.Execute (request);
    }

}
```

Sample Response:

```
{
  "preview": {
    "id": "test_500",
    "valid": true,
    "status": "preview_complete",
    "quantity": 8,
    "created_at": 1590080191,
    "summary": {
```

(continues on next page)

(continued from previous page)

```
      "result": {
        "deliverable": 37.5,
        "undeliverable": 23,
        "catch_all" : 2,
        "unknown": 37.5
      },
      "risk": {
        "high": 25,
        "low": 25,
        "medium": 12.5,
        "unknown": 37.5
      }
    }
  }
}
```

Field Explanation:

Field	Type	Description
id	string	list_id name given when the list was initially created
created_at	string	Date/Time that the request was initiated
quantity	integer	number of total items in the list to be previewed
status	string	current state of the list validation request. (`preview_processing`, `preview_complete`)
valid	bool	a boolean to represent if the list is valid
summary	collection	summary of the validations in the list provided
result	array	nested results averaged. (`deliverable`, `undeliverable`, `catch_all` and `unknown`)
risk	array	nested risk assessment count (`high`, `low`, `medium` or `unknown`)

Get a list of bulk validation previews:

This request will return a list of bulk validation previews.

```
curl -s --user 'api:YOUR_API_KEY' -G \
    https://api.mailgun.net/v4/address/validate/preview
```

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode getBulkPreviews() throws UnirestException {

        HttpResponse<JsonNode> request = Unirest.get("https://api.mailgun.net/v4/
↪address/validate/preview)
            .basicAuth("api", API_KEY)
            .asJson();
```

(continues on next page)

(continued from previous page)

```
        return request.getBody();
    }
}
```

```
# Currently, the PHP SDK does not support the v4 Validations endpoint.
# Consider using the following php curl function.
function get_bulk_validations() {
  $ch = curl_init();

  curl_setopt($ch, CURLOPT_HTTPAUTH, CURLAUTH_BASIC);
  curl_setopt($ch, CURLOPT_USERPWD, 'api:PRIVATE_API_KEY');
  curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);

  curl_setopt($ch, CURLOPT_CUSTOMREQUEST, 'GET');
  curl_setopt($ch, CURLOPT_URL, 'https://api.mailgun.net/v4/address/validate/preview
↪');
  $result = curl_exec($ch);
  curl_close($ch);

  return $result;
}
```

```
def get_bulk_previews():
    return requests.get(
        "https://api.mailgun.net/v4/address/validate/preview",
        auth=('api', 'YOUR_API_KEY'))
```

```
def get_bulk_previews
  RestClient.get("https://api:YOUR_API_KEY"\
                 "@api.mailgun.net/v4/address/validate/preview"\,
                 {|response, request, result| response })
end
```

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class GetBulkPreviews
{

    public static void Main (string[] args)
    {
        Console.WriteLine (GetPreviews ().Content.ToString ());
    }

    public static IRestResponse GetPreviews ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v4");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
```

(continues on next page)

(continued from previous page)

```
        request.Resource = "/address/validate/preview";
        return client.Execute(request);
    }

}
```

Sample Response:

```
{
  "previews": [
    {
      "id": "test_500",
      "valid": true,
      "status": "preview_complete",
      "quantity": 8,
      "created_at": 1590080191,
      "summary": {
        "result": {
          "deliverable": 37.5,
          "do_not_send": 0,
          "undeliverable": 23,
          "catch_all": 2,
          "unknown": 37.5
        },
        "risk": {
          "high": 25,
          "low": 25,
          "medium": 12.5,
          "unknown": 37.5
        }
      }
    },
    {
      "id": "test_501",
      "valid": true,
      "status": "preview_complete",
      "quantity": 8,
      "created_at": 1590155015,
      "summary": {
        "result": {
          "deliverable": 37.5,
          "do_not_send": 0,
          "undeliverable": 23,
          "catch_all": 2,
          "unknown": 37.5
        },
        "risk": {
          "high": 25,
          "low": 25,
          "medium": 12.5,
          "unknown": 37.5
        }
      }
    }
  ]
}
```

Response Fields Explanation:

Field	Type	Description
previews	collection	A collection of bulk validation previews.

Create a bulk validation preview:

```
curl -s --user 'api:YOUR_API_KEY' \
    https://api.mailgun.net/v4/address/validate/preview/LIST_NAME \
    -F 'file=@/path/to/file' \
```

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode createBulkPreview() throws UnirestException {

        HttpResponse <JsonNode> request = Unirest.post("https://api.mailgun.net/v4/
↪address/validate/preview/LIST_NAME")
            .basicAuth("api", API_KEY)
            .field("file", new File("/path/to/file"))
            .asJson();

        return request.getBody();
    }
}
```

```
# Currently, the PHP SDK does not support the v4 Validations endpoint.
# Consider using the following php curl function.
function create_bulk_preview() {
  $ch = curl_init();

  curl_setopt($ch, CURLOPT_HTTPAUTH, CURLAUTH_BASIC);
  curl_setopt($ch, CURLOPT_USERPWD, 'api:PRIVATE_API_KEY');
  curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);

  curl_setopt($ch, CURLOPT_CUSTOMREQUEST, 'POST');
  curl_setopt($ch, CURLOPT_URL, 'https://api.mailgun.net/v4/address/validate/preview/
↪LIST_NAME');
  curl_setopt($ch, CURLOPT_POSTFIELDS, array(
      'file'=> curl_file_create('subscribers.csv')
  ));

  $result = curl_exec($ch);
  curl_close($ch);

  return $result;
}
```

```
def create_bulk_preview():
    return requests.post(
```

(continues on next page)

(continued from previous page)

```
        "https://api.mailgun.net/v4/address/validate/preview/LIST_NAME",
        files = {'file': open('/path/to/file','rb')},
        auth=('api', 'YOUR_API_KEY'))
```

```
def create_bulk_preview
  RestClient.post("https://api:YOUR_API_KEY" \
                  "@api.mailgun.net/v4/address/validate/preview/LIST_NAME",
                  fields_hash.merge(:file => File.new('/path/to/file')))
end
```

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class BulkPreview
{

    public static void Main (string[] args)
    {
        Console.WriteLine (CreateBulkPreview ().Content.ToString ());
    }

    public static IRestResponse CreateBulkPreview ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v4");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.Resource = "address/validate/preview/{list}";
        request.AddParameter ("list", "LIST_NAME",
                              ParameterType.UrlSegment);
        request.Method = Method.POST;
        request.AddFile("file", @"/path/to/file");
        return client.Execute (request);
    }

}
```

Sample Response:

```
{
  "id": "test_501",
  "message": "The bulk preview was submitted."
}
```

Delete a bulk validation preview:

```
curl -s --user 'api:YOUR_API_KEY' -X DELETE \
    https://api.mailgun.net/v4/address/validate/preview/LIST_NAME
```

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
```

(continues on next page)

(continued from previous page)

```
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode cancelBulkPreview() throws UnirestException {
        url = "https://api.mailgun.net/v4/address/validate/preview/LIST_NAME"
        HttpResponse <JsonNode> request = Unirest.delete(url)
            .basicAuth("api", API_KEY)
            .asJson();

        return request.getBody();
    }
}
```

```
# Currently, the PHP SDK does not support the v4 Validations endpoint.
# Consider using the following php curl function.
function delete_bulk_preview() {
  $ch = curl_init();

  curl_setopt($ch, CURLOPT_HTTPAUTH, CURLAUTH_BASIC);
  curl_setopt($ch, CURLOPT_USERPWD, 'api:PRIVATE_API_KEY');
  curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);

  curl_setopt($ch, CURLOPT_CUSTOMREQUEST, 'DELETE');
  curl_setopt($ch, CURLOPT_URL, 'https://api.mailgun.net/v4/address/validate/preview/
↪LIST_NAME');
  $result = curl_exec($ch);
  curl_close($ch);

  return $result;
}
```

```
def delete_bulk_preview():
    return requests.delete(
        ("https://api.mailgun.net/v4/address/validate/preview/LIST_NAME"),
        auth=('api', 'YOUR_API_KEY'))
```

```
def delete_bulk_preview
  RestClient.delete("https://api:YOUR_API_KEY" \
                    "@api.mailgun.net/v4/address/validate/preview/LIST_NAME")
end
```

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class DeleteBulkPreview
{

    public static void Main (string[] args)
    {
```

(continues on next page)

(continued from previous page)

```
        Console.WriteLine (DeletePreview ().Content.ToString ());
    }

    public static IRestResponse DeletePreview()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v4");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.Resource = "address/validate/preview/{list}";
        request.AddParameter ("list", "LIST_NAME",
                              ParameterType.UrlSegment);
        request.Method = Method.DELETE;
        return client.Execute (request);
    }

}
```

Sample Response: A 204 will be returned upon successful deletion.

4.15 Inbox Placement

4.15.1 Start an inbox placement test

```
POST /v3/inbox/tests
```

Start an inbox placement test. The required form fields are as follows.

Field	Description
domain	The sending domain registered with mailgun to send the messages with.
subject	The subject associated with the message being tested.
html	The html that makes up the body of the message being tested.
from	The sending address associated with the sending of the message.

```
curl -X POST https://api.mailgun.net/v3/inbox/tests \
  -F 'domain=domain.com' \
  -F 'subject=testSubject' \
  -F 'from=user@sending_domain.com' \
  --form-string html='<html>HTML version of the body</html>' \
  --user 'api:<YOUR_API_KEY'>'
```

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...
```

(continues on next page)

(continued from previous page)

```
    public static JsonNode validateMailingList() throws UnirestException {

        HttpResponse <JsonNode> request = Unirest.post("https://api.mailgun.net/v3/
↪inbox/tests")
            .basicAuth("api", API_KEY)
            .field("domain", "domain.com")
            .field("subject", "testSubject")
            .field("from", "user@sending_domain.com")
            .field("html", "<html>HTML version of the body</html>")
            .asJson();

        return request.getBody();
    }
}
```

```
# Currently, the PHP SDK does not support the inbox placement endpoint.
# Consider using the following php curl function.
function create_inbox_placement_test() {
  $ch = curl_init();

  curl_setopt($ch, CURLOPT_HTTPAUTH, CURLAUTH_BASIC);
  curl_setopt($ch, CURLOPT_USERPWD, 'api:PRIVATE_API_KEY');
  curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);

  curl_setopt($ch, CURLOPT_CUSTOMREQUEST, 'POST');
  curl_setopt($ch, CURLOPT_URL, 'https://api.mailgun.net/v3/inbox/tests');
  curl_setopt($ch, CURLOPT_POSTFIELDS, array(
      'domain'=> 'domain.com',
      'from'=> 'user@sending_domain.com',
      'subject'=>'testSubject',
      'html'=>'<html>HTML version of the body</html>',
      )
  );

  $result = curl_exec($ch);
  curl_close($ch);

  return $result;
}
```

```
def create_inbox_placement_test():
    data = {'domain': 'domain.com',
            'from': 'user@sending_domain.com',
            'subject': 'testSubject',
            'html': '<html>HTML version of the body</html>' }
    return requests.post(
        "https://api.mailgun.net/v3/inbox/tests", data=data,
        auth=('api', 'YOUR_API_KEY'))
```

```
def validate_mailing_list
  data = {'domain'=> 'domain.com',
          'from'=> 'user@sending_domain.com',
          'subject'=> 'testSubject',
          'html'=> '<html>HTML version of the body</html>' }
```

(continues on next page)

(continued from previous page)

```
  RestClient.post("https://api:YOUR_API_KEY" \
                  "@api.mailgun.net/v3/inbox/tests",
                  fields_hash.merge(data))
end
```

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class CreateInboxPlacementTest
{

    public static void Main (string[] args)
    {
        Console.WriteLine (StartInboxPlacementTest ().Content.ToString ());
    }

    public static IRestResponse StartInboxPlacementTest ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.AddParameter ("domain", "YOUR_DOMAIN_NAME", ParameterType.UrlSegment);
        request.Resource = "inbox/tests";
        request.AddParameter ("from", "user@sending_domain.com'");
        request.AddParameter ("domain", "domain.com");
        request.AddParameter ("subject", "testSubject");
        request.AddParameter ("html", "<html>HTML version of the body</html>");
        request.Method = Method.POST;
        return client.Execute (request);
    }
}
```

Example response for creating an inbox placement test.

```
{
    "tid": "5e22167af8424f444ca6d8e2"
}
```

Field Explanation:

Name	Type	Description
tid	string	Unique identifier for an inbox placement test.

4.15.2 Get all inbox placement tests

This API endpoint is used for interfacing with the inbox placement service. Pricing details for Mailgun's inbox placement service can be found on our pricing page. Mailgun's inbox placement service is intended to be used for seed testing for emails. Refer to our Acceptable Use Policy (AUP) for more information about how to use the service appropriately.

```
GET /v3/inbox/tests
```

Retrieve a list of all the inbox placement tests and their results ran on the account.

```
curl -s --user 'api:YOUR_API_KEY' -G \
    https://api.mailgun.net/v3/inbox/tests
```

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode getInboxPlacementTests() throws UnirestException {

        HttpResponse<JsonNode> request = Unirest.get("https://api.mailgun.net/v3/
↪inbox/tests")
            .basicAuth("api", API_KEY)
            .asJson();

        return request.getBody();
    }
}
```

```
# Currently, the PHP SDK does not support the Inbox Placement endpoint.
# Consider using the following php curl function.
function get_inbox_placement_tests() {
  $ch = curl_init();

  curl_setopt($ch, CURLOPT_HTTPAUTH, CURLAUTH_BASIC);
  curl_setopt($ch, CURLOPT_USERPWD, 'api:PRIVATE_API_KEY');
  curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);

  curl_setopt($ch, CURLOPT_CUSTOMREQUEST, 'GET');
  curl_setopt($ch, CURLOPT_URL, 'https://api.mailgun.net/v3/inbox/tests');
  $result = curl_exec($ch);
  curl_close($ch);

  return $result;
}
```

```
def get_inbox_placement_tests():
    return requests.get(
        "https://api.mailgun.net/v3/inbox/tests",
        auth=('api', 'YOUR_API_KEY'))
```

```
def get_inbox_placement_tests
  RestClient.get("https://api:YOUR_API_KEY"\
                 "@api.mailgun.net/v3/inbox/tests"\
                 ){|response, request, result| response }
end
```

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class InboxPlacementTests
{

    public static void Main (string[] args)
    {
        Console.WriteLine (GetInboxPlacementTests ().Content.ToString ());
    }

    public static IRestResponse GetInboxPlacementTests ()
    {
        RestClient client = new RestClient ();
        client.BaseUrl = new Uri ("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator ("api",
                                        "YOUR_API_KEY");
        RestRequest request = new RestRequest ();
        request.Resource = "/inbox/tests";
        return client.Execute (request);
    }

}
```

Example response of getting a list of inbox placement tests.

```
{
  "paging": {
    "first": "https://api.mailgun.net/v3/inbox/tests?ascending=0",
    "last": "https://api.mailgun.net/v3/inbox/tests?ascending=1",
    "next": "https://api.mailgun.net/v3/inbox/tests?ascending=0&
↪cursor=5e22167af8424f444ca6d8ea",
    "previous": "https://api.mailgun.net/v3/inbox/tests?ascending=1&
↪cursor=5e22167af8424f444ca6d8e2"
  },
  "tests": [
    {
      "tid": "5e22167af8424f444ca6d8e2",
      "counts": {
        "inbox": 2,
        "junk": 1,
        "missing": 0
      },
      "domain": "ibp.voxcreator.com",
      "status": "completed",
      "seeds": [
        "joesmith915@o365.mailgun.email",
        "joesmith916@o365.mailgun.email",
        "janedoe@o365.mailgun.email"
      ],
      "start_time": "2020-01-17T20:18:02.093Z",
      "end_time": "2020-01-17T20:33:02.097Z",
      "summary": {
        "stats": {
```

(continues on next page)

(continued from previous page)


```
        "averages": {
          "mailgun_send": {
            "inbox": 95.48,
            "junk": 3.2,
            "missing": 1.32
          }
        }
      }
    },
    "subject": "This Service is Awesome!"
  },
  {
    "tid": "5e22167af8424f444ca6d8ea",
    "counts": {
      "inbox": 2,
      "junk": 1,
      "missing": 0
    },
    "domain": "ibp.voxcreator.com",
    "status": "completed",
    "seeds": [
      "joesmith915@o365.mailgun.email",
      "joesmith916@o365.mailgun.email",
      "janedoe@o365.mailgun.email"
    ],
    "start_time": "2020-01-17T20:18:02.093Z",
    "end_time": "2020-01-17T20:33:02.097Z",
    "summary": {
      "stats": {
        "averages": {
          "mailgun_send": {
            "inbox": 95.48,
            "junk": 3.2,
            "missing": 1.32
          }
        }
      }
    },
    "subject": "This Mail is Awesome!"
  }
],
"total": 2
}
```


Field Explanation:

Name	Type	Description
paging	object	Urls used to page through the list of inbox placement tests.
tests	array	List of inbox placement tests.
total	integer	Total number of inbox placement tests ran for the account

4.15.3 Retrieve individual test details

```
GET /v3/inbox/tests/<test_id>
```

Retrieve a single inbox placement test.

Parameter	Description
test_id	The unique identifier for the inbox placement test.

```
curl -s --user 'api:YOUR_API_KEY' -G \
    https://api.mailgun.net/v3/inbox/tests/TEST_ID
```

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode getInboxPlacementTest() throws UnirestException {

        HttpResponse<JsonNode> request = Unirest.get("https://api.mailgun.net/v3/
↪inbox/tests/{test_id}")
            .basicAuth("api", API_KEY)
            .asJson();

        return request.getBody();
    }
}
```

```
# Currently, the PHP SDK does not support the inbox placement endpoint.
# Consider using the following php curl function.
function get_inbox_placement_test() {
  $ch = curl_init();

  curl_setopt($ch, CURLOPT_HTTPAUTH, CURLAUTH_BASIC);
  curl_setopt($ch, CURLOPT_USERPWD, 'api:PRIVATE_API_KEY');
  curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);

  curl_setopt($ch, CURLOPT_CUSTOMREQUEST, 'GET');
  curl_setopt($ch, CURLOPT_URL, 'https://api.mailgun.net/v3/inbox/tests/TEST_ID');
  $result = curl_exec($ch);
  curl_close($ch);

  return $result;
}
```

```
def get_inbox_placement_test():
    return requests.get(
        "https://api.mailgun.net/v3/inbox/tests/TEST_ID",
        auth=('api', 'YOUR_API_KEY'))
```

```ruby
def get_inbox_placement_test
  RestClient.get("https://api:YOUR_API_KEY"\
                 "@api.mailgun.net/v3/inbox/tests/TEST_ID"\
                 ){|response, request, result| response }
end
```

```csharp
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class InboxPlacementTest
{

    public static void Main (string[] args)
    {
        Console.WriteLine (GetInboxPlacementTest().Content.ToString());
    }

    public static IRestResponse GetInboxPlacementTest()
    {
        RestClient client = new RestClient();
        client.BaseUrl = new Uri("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator("api",
                                       "YOUR_API_KEY");
        RestRequest request = new RestRequest();
        request.AddParameter("test_id", "TEST_ID", ParameterType.UrlSegment);
        request.Resource = "/inbox/tests/{test_id}";
        return client.Execute(request);
    }

}
```

Example response of getting an inbox placement test.

```
{
  "tid": "5e22167af8424f444ca6d8e2",
  "counts": {
    "inbox": 2,
    "junk": 1,
    "missing": 0
  },
  "domain": "inbox_placement.domain.com",
  "status": "completed",
  "seeds": [
    "joesmith915@o365.mailgun.email",
    "joesmith916@o365.mailgun.email",
    "janedoe@o365.mailgun.email"
  ],
  "start_time": "2020-01-17T20:18:02.093Z",
  "end_time": "2020-01-17T20:33:02.097Z",
  "summary": {
    "stats": {
      "averages": {
        "mailgun_send": {
          "inbox": 95.48,
```

(continues on next page)

(continued from previous page)

```
            "junk": 3.2,
            "missing": 1.32
        }
      }
    }
  },
  "render_url": "https://mg-inbox-placement.s3.amazonaws.com/export/
↪b156f44d9c27ee74422a3e38dd831343ec541938.png?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-
↪Credential=AKIAJR5LUWTPXYIVY4GA%2F20200118%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-
↪Date=20200118T225458Z&X-Amz-Expires=3600&X-Amz-SignedHeaders=host&X-Amz-
↪Signature=f255b140f0efa94507bb62542b7a1191faaac708588edcb0d5dfd88a777e0061",
  "subject": "This is an awesome API!"
}
```

Field Explanation:

Name	Type	Description
tid	string	Unique identifier for an inbox placement test.
counts	object	Total counts for the mailboxes where the messages landed across the seed address sent to.
domain	string	The sending domain used to send the messages to the seed addresses.
status	string	The current status for a test. e.g. (“running”, “completed”, “error”, “created”)
seeds	array	The seed addresses the test message was sent to.
start_time	string	The time in which the inbox placement test began.
end_time	string	The time in which the inbox placement test ended.
summary	object	A summarized view of the inbox placement test.
rendered_url	string	A link to a rendered version of the message that was sent to the seed addresses.
subject	string	The subject for the message that was sent to the seed addresses.

4.15.4 Delete an inbox placement test

```
DELETE /v3/inbox/tests/<test_id>
```

Delete a single inbox placement test.

Parameter	Description
test_id	The unique identifier for the inbox placement test.

```
curl -s --user 'api:YOUR_API_KEY' -X DELETE -G \
    https://api.mailgun.net/v3/inbox/tests/TEST_ID
```

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...
```

(continues on next page)

(continued from previous page)

```
    public static JsonNode deleteInboxPlacementTest() throws UnirestException {

        HttpResponse<JsonNode> request = Unirest.delete("https://api.mailgun.net/v3/
↪inbox/tests/{test_id}")
            .basicAuth("api", API_KEY)
            .asJson();

        return request.getBody();
    }
}
```

```
# Currently, the PHP SDK does not support the inbox placement endpoint.
# Consider using the following php curl function.
function delete_inbox_placement_test() {
  $ch = curl_init();

  curl_setopt($ch, CURLOPT_HTTPAUTH, CURLAUTH_BASIC);
  curl_setopt($ch, CURLOPT_USERPWD, 'api:PRIVATE_API_KEY');
  curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);

  curl_setopt($ch, CURLOPT_CUSTOMREQUEST, 'DELETE');
  curl_setopt($ch, CURLOPT_URL, 'https://api.mailgun.net/v3/inbox/tests/TEST_ID');
  $result = curl_exec($ch);
  curl_close($ch);

  return $result;
}
```

```
def delete_inbox_placement_test():
    return requests.delete(
        "https://api.mailgun.net/v3/inbox/tests/TEST_ID",
        auth=('api', 'YOUR_API_KEY'))
```

```
def delete_inbox_placement_test
  RestClient.delete("https://api:YOUR_API_KEY"\
                    "@api.mailgun.net/v3/inbox/tests/TEST_ID"\
                    ){|response, request, result| response }
end
```

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class InboxPlacementTest
{

    public static void Main (string[] args)
    {
        Console.WriteLine (GetInboxPlacementTest().Content.ToString());
    }

    public static IRestResponse DeleteInboxPlacementTest()
    {
        RestClient client = new RestClient();
```

(continues on next page)

(continued from previous page)

```
        client.BaseUrl = new Uri("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator("api",
                                       "YOUR_API_KEY");
        RestRequest request = new RestRequest(Method.DELETE);
        request.AddParameter("test_id", "TEST_ID", ParameterType.UrlSegment);
        request.Resource = "/inbox/tests/{test_id}";
        return client.Execute(request);
    }


}
```

Example response for deleting an inbox placement test.

```
{
    "message": "deleted"
}
```

4.15.5 Retrieve provider results (counters) for a test

```
GET /v3/inbox/tests/<test_id>/counters
```

Retrieve a provider breakdown of the inbox placement test's counters.

Parameter	Description
test_id	The unique identifier for the inbox placement test.

```
curl -s --user 'api:YOUR_API_KEY' -G \
    https://api.mailgun.net/v3/inbox/tests/TEST_ID/counters
```

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode getInboxPlacementTestCounters() throws UnirestException {

        HttpResponse<JsonNode> request = Unirest.get("https://api.mailgun.net/v3/
↪inbox/tests/{test_id}/counters")
            .basicAuth("api", API_KEY)
            .asJson();

        return request.getBody();
    }
}
```

```
# Currently, the PHP SDK does not support the inbox placement endpoint.
# Consider using the following php curl function.
```

(continues on next page)

(continued from previous page)

```php
function get_inbox_placement_test_counters() {
  $ch = curl_init();

  curl_setopt($ch, CURLOPT_HTTPAUTH, CURLAUTH_BASIC);
  curl_setopt($ch, CURLOPT_USERPWD, 'api:PRIVATE_API_KEY');
  curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);

  curl_setopt($ch, CURLOPT_CUSTOMREQUEST, 'GET');
  curl_setopt($ch, CURLOPT_URL, 'https://api.mailgun.net/v3/inbox/tests/TEST_ID/
↪counters');
  $result = curl_exec($ch);
  curl_close($ch);

  return $result;
}
```

```python
def get_inbox_placement_test_counters():
    return requests.get(
        "https://api.mailgun.net/v3/inbox/tests/TEST_ID/counters",
        auth=('api', 'YOUR_API_KEY'))
```

```ruby
def get_inbox_placement_test_counters
  RestClient.get("https://api:YOUR_API_KEY"\
                 "@api.mailgun.net/v3/inbox/tests/TEST_ID/counters"\
                 ){|response, request, result| response }
end
```

```c#
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class InboxPlacementTest
{

    public static void Main (string[] args)
    {
        Console.WriteLine (GetInboxPlacementTestCounters().Content.ToString());
    }

    public static IRestResponse GetInboxPlacementTestCounters()
    {
        RestClient client = new RestClient();
        client.BaseUrl = new Uri("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator("api",
                                       "YOUR_API_KEY");
        RestRequest request = new RestRequest();
        request.AddParameter("test_id", "TEST_ID", ParameterType.UrlSegment);
        request.Resource = "/inbox/tests/{test_id}/counters";
        return client.Execute(request);
    }
}
```

Example response for inbox placement counters.

```
{
  "counters": [
    {
      "inbox": 2,
      "junk": 1,
      "provider": "o365"
    },
  ]
}
```

Field Explanation:

Name	Type	Description
tid	string	Unique identifier for an inbox placement test.

4.15.6 Get all inbox placement test checks

```
GET /v3/inbox/tests/<test_id>/checks
```

Retrieve a list of all the checks sent for an inbox placement test.

Parameter	Description
test_id	The unique identifier for the inbox placement test.

```
curl -s --user 'api:YOUR_API_KEY' -G \
    https://api.mailgun.net/v3/inbox/tests/TEST_ID/checks
```

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode getInboxPlacementTestCounters() throws UnirestException {

        HttpResponse<JsonNode> request = Unirest.get("https://api.mailgun.net/v3/
↪inbox/tests/{test_id}/checks")
            .basicAuth("api", API_KEY)
            .asJson();

        return request.getBody();
    }
}
```

```
# Currently, the PHP SDK does not support the inbox placement endpoint.
# Consider using the following php curl function.
function get_inbox_placement_test_checks() {
  $ch = curl_init();
```

(continues on next page)

(continued from previous page)

```
  curl_setopt($ch, CURLOPT_HTTPAUTH, CURLAUTH_BASIC);
  curl_setopt($ch, CURLOPT_USERPWD, 'api:PRIVATE_API_KEY');
  curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);

  curl_setopt($ch, CURLOPT_CUSTOMREQUEST, 'GET');
  curl_setopt($ch, CURLOPT_URL, 'https://api.mailgun.net/v3/inbox/tests/TEST_ID/checks
↪');
  $result = curl_exec($ch);
  curl_close($ch);

  return $result;
}
```

```
def get_inbox_placement_test_checks():
    return requests.get(
        "https://api.mailgun.net/v3/inbox/tests/TEST_ID/checks",
        auth=('api', 'YOUR_API_KEY'))
```

```
def get_inbox_placement_test_checks
  RestClient.get("https://api:YOUR_API_KEY"\
                 "@api.mailgun.net/v3/inbox/tests/TEST_ID/checks"\
                 {|response, request, result| response }
end
```

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class InboxPlacementTest
{

    public static void Main (string[] args)
    {
        Console.WriteLine (GetInboxPlacementTestChecks().Content.ToString());
    }

    public static IRestResponse GetInboxPlacementTestChecks()
    {
        RestClient client = new RestClient();
        client.BaseUrl = new Uri("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator("api",
                                       "YOUR_API_KEY");
        RestRequest request = new RestRequest();
        request.AddParameter("test_id", "TEST_ID", ParameterType.UrlSegment);
        request.Resource = "/inbox/tests/{test_id}/checks";
        return client.Execute(request);
    }
}
```

Example response of getting a list of all checks for an inbox placement test.

```
{
  "checks": [
```

(continues on next page)

(continued from previous page)


```
    {
      "address": "aa_ext_test01mg@comcast.net",
      "provider": "comcast",
      "ip": "00.114.000.000",
      "folder": "inbox",
      "headers": "Return-Path: <bounce+b671ba.25af4b3-aa_ext_test01mg=comcast.
↪net@test.domain.com>\r\nDelivered-To: aa_ext_test01mg@comcast.net\r\nReceived: from␣
↪dovdir4-asa-04o.email.comcast.net ([96.114.154.247])\r\n\tby dovback4-asa-18o.email.
↪comcast.net with LMTP\r\n\tid iD97DYUWIl58dAAAt8rViA\r\n\t(envelope-from
↪<bounce+b671ba.25af4b3-aa_ext_test01mg=comcast.net@test.domain.com>)\r\n\tfor <aa_
↪ext_test01mg@comcast.net>; Fri, 17 Jan 2020 20:18:13 +0000\r\nReceived: from dovpxy-
↪asc-09o.email.comcast.net ([96.114.154.247])\r\n\tby dovdir4-asa-04o.email.comcast.
↪net with LMTP\r\n\tid 2AU9DYUWIl79cAAAp4A1CQ\r\n\t(envelope-from <bounce+b671ba.
↪25af4b3-aa_ext_test01mg=comcast.net@test.domain.com>)\r\n\tfor <aa_ext_
↪test01mg@comcast.net>; Fri, 17 Jan 2020 20:18:13 +0000\r\nReceived: from resimta-po-
↪40v.sys.comcast.net ([96.114.154.247])\r\n\t(using TLSv1.2 with cipher ECDHE-RSA-
↪AES256-GCM-SHA384 (256/256 bits))\r\n\tby dovpxy-asc-09o.email.comcast.net with␣
↪LMTP id IC0YMYMWIl5eOQAAquDo3w\r\n\t; Fri, 17 Jan 2020 20:18:13 +0000\r\nReceived:␣
↪from so254-22.mailgun.net ([00.114.000.000])\r\n\tby resimta-po-40v.sys.comcast.net␣
↪with ESMTP\r\n\tid sY4Iie8ttwt3TsY4KiR4U0; Fri, 17 Jan 2020 20:18:12 +0000\r\nX-CAA-
↪SPAM: F00000\r\nX-Xfinity-VAAS:␣
↪gggruggvucftvghtrhhoucdtuddrgedugedrtdekgdegudcutefuodetggdotefrodftvfcurfhrohhfihhlvgemucevohhmtgg
↪Xfinity-VMeta: sc=0.00;st=legit\r\nX-Xfinity-Message-Heuristics: IPv6:N;TLS=1;SPF=1;
↪DMARC=\r\nAuthentication-Results: resimta-po-40v.sys.comcast.net;\r\n\tdkim=pass␣
↪header.d=test.domain.com header.b=NA9s3uW5\r\nDKIM-Signature: a=rsa-sha256; v=1;␣
↪c=relaxed/relaxed; d=test.domain.com; q=dns/txt;\r\n s=k1; t=1579292292; h=Mime-
↪Version: Content-Type: Subject: From: To:\r\n Message-Id: Sender: Date: Content-
↪Transfer-Encoding;\r\n bh=bYuUQQIdEIrM6gyAxa1Xp4Nd0A2cWpdksHogCDsE+j8=;␣
↪b=NA9s3uW5ejKsh0a/lDlEEfoKhyh8OevkRYfDau6tqRIYw/82eEWHxSRQrbTdKjxOWSH4ZHw1\r\n␣
↪DpigSIhjF3Ub5BQdV64LtN8Bcd1ps/2exdIa21qiKewJDFQht9KoLURTCI5FY+03dywAIeM4\r\n yOp9o/
↪cuQTKJH2qM4iiDgRE0Gsg=\r\nX-Mailgun-Sending-Ip: 00.114.000.000\r\nX-Mailgun-Sid:␣
↪WyJjYTRiMyIsICJhYV9leHRfdGVzdDAxbWdAY29tY2FzdC5uZXQiLCAiMjVhZjRiMyJd\r\nContent-
↪Transfer-Encoding: quoted-printable\r\nReceived: by luna.mailgun.net with HTTP; Fri,
↪ 17 Jan 2020 20:18:11 +0000\r\nDate: Fri, 17 Jan 2020 20:18:11 +0000\r\nSender:␣
↪user@test.domain.com\r\nMessage-Id: <20200117201811.1.BDA3E43254369346@test.domain.
↪com>\r\nX-Mailgun-Seed-Test-Id: 5e22167af8424f444ca6d8e2\r\nTo: aa_ext_
↪test01mg@comcast.net\r\nFrom: user@test.domain.com\r\nSubject:␣
↪testSubject\r\nContent-Type: text/html; charset=\"ascii\"\r\nMime-Version: 1.0",
      "message_id": "<20200117201811.1.BDA3E43254369346@test.domain.com>",
      "time": "2020-01-17T20:18:08.8Z"
    },
    {
      "address": "aa_ext_test02mg@comcast.net",
      "provider": "comcast",
      "ip": "00.114.000.000",
      "folder": "inbox",
      "headers": "Return-Path: <bounce+1e9aa3.25af4b3-aa_ext_test02mg=comcast.
↪net@test.domain.com>\r\nDelivered-To: aa_ext_test02mg@comcast.net\r\nReceived: from␣
↪dovdir3-asa-01o.email.comcast.net ([96.114.154.247])\r\n\tby dovback3-asa-07o.email.
↪comcast.net with LMTP\r\n\tid uCAWG4gWIl6IDgAAVWOgEw\r\n\t(envelope-from
↪<bounce+1e9aa3.25af4b3-aa_ext_test02mg=comcast.net@test.domain.com>)\r\n\tfor <aa_
↪ext_test02mg@comcast.net>; Fri, 17 Jan 2020 20:18:16 +0000\r\nReceived: from dovpxy-
↪asc-01o.email.comcast.net ([96.114.154.247])\r\n\tby dovdir3-asa-01o.email.comcast.
↪net with LMTP\r\n\tid 0CrqGogWIl7xTAAAwP0GGg\r\n\t(envelope-from <bounce+1e9aa3.
↪25af4b3-aa_ext_test02mg=comcast.net@test.domain.com>)\r\n\tfor <aa_ext_
↪test02mg@comcast.net>; Fri, 17 Jan 2020 20:18:16 +0000\r\nReceived: from resimta-po-
↪40v.sys.comcast.net ([96.114.154.247])\r\n\t(using TLSv1.2 with cipher ECDHE-RSA-
↪AES256-GCM-SHA384 (256/256 bits))\r\n\tby dovpxy-asc-01o.email.comcast.net with␣
↪LMTP id OD4qI4UWIl5UQQAAyeh4YQ\r\n\t; Fri, 17 Jan 2020 20:18:16 +0000\r\nReceived:␣
↪from so254-22.mailgun.net ([00.114.000.000])\r\n\tby resimta-po-40v.sys.comcast.net␣
↪with ESMTP\r\n\tid sY4Iie8ttwt3TsY4MiR4WV; Fri, 17 Jan 2020 20:18:16 +0000\r\nX-CAA-
↪SPAM: F00000\r\nX-Xfinity-VAAS:␣
↪gggruggvucftvghtrhhoucdtuddrgedugedrtdekgdegudcutefuodetggdotefrodftvfcurfhrohhfihhlvgemucevohhmtgg
↪Xfinity-VMeta: sc=0.00;st=legit\r\nX-Xfinity-Message-Heuristics: IPv6:N;TLS=1;SPF=1;
```


(continues on next page)

(continued from previous page)

```
      "message_id": "<20200117201809.1.82C23D86DE20410C@test.domain.com>",
      "time": "2020-01-17T20:18:08.8Z"
    }
  ]
}
```

Field Explanation:

Name	Type	Description
checks	array	Collection of checks that represent the messages sent to the seed mailboxes.

4.15.7 Get a single inbox placement test check

```
GET /v3/inbox/tests/<test_id>/checks/<address>
```

Retrieve a check sent for an inbox placement test.

Parameter	Description
test_id	The unique identifier for the inbox placement test.
address	The seed address sent to in the inbox placement test.

```
curl -s --user 'api:YOUR_API_KEY' -G \
    https://api.mailgun.net/v3/inbox/tests/TEST_ID/checks/ADDRESS
```

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class MGSample {

    // ...

    public static JsonNode getInboxPlacementTestCheck() throws UnirestException {

        HttpResponse<JsonNode> request = Unirest.get("https://api.mailgun.net/v3/
↪inbox/tests/{test_id}/checks/{address}")
            .basicAuth("api", API_KEY)
            .asJson();

        return request.getBody();
    }
}
```

```
# Currently, the PHP SDK does not support the inbox placement endpoint.
# Consider using the following php curl function.
function get_inbox_placement_test() {
  $ch = curl_init();

  curl_setopt($ch, CURLOPT_HTTPAUTH, CURLAUTH_BASIC);
  curl_setopt($ch, CURLOPT_USERPWD, 'api:PRIVATE_API_KEY');
  curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
```

(continues on next page)

(continued from previous page)

```
  curl_setopt($ch, CURLOPT_CUSTOMREQUEST, 'GET');
  curl_setopt($ch, CURLOPT_URL, 'https://api.mailgun.net/v3/inbox/tests/TEST_ID/
↪checks/ADDRESS');
  $result = curl_exec($ch);
  curl_close($ch);

  return $result;
}
```

```
def get_inbox_placement_test_check():
    return requests.get(
        "https://api.mailgun.net/v3/inbox/tests/TEST_ID/checks/ADDRESS",
        auth=('api', 'YOUR_API_KEY'))
```

```
def get_inbox_placement_test_check
  RestClient.get("https://api:YOUR_API_KEY"\
                 "@api.mailgun.net/v3/inbox/tests/TEST_ID/checks/ADDRESS"\
                 ){|response, request, result| response }
end
```

```
using System;
using System.IO;
using RestSharp;
using RestSharp.Authenticators;

public class InboxPlacementTest
{

    public static void Main (string[] args)
    {
        Console.WriteLine (GetInboxPlacementTestCheck().Content.ToString());
    }

    public static IRestResponse GetInboxPlacementTestCheck()
    {
        RestClient client = new RestClient();
        client.BaseUrl = new Uri("https://api.mailgun.net/v3");
        client.Authenticator =
            new HttpBasicAuthenticator("api",
                                       "YOUR_API_KEY");
        RestRequest request = new RestRequest();
        request.AddParameter ("test_id", "TEST_ID", ParameterType.UrlSegment);
        request.AddParameter ("address", "ADDRESS", ParameterType.UrlSegment);
        request.Resource = "/inbox/tests/{test_id}/checks/{address}";
        return client.Execute(request);
    }


}
```

Example response of getting a single check for an inbox placement test.

```
{
  "address": "aa_ext_test02mg@comcast.net",
  "provider": "comcast",
```

(continues on next page)

(continued from previous page)


```
  "ip": "96.114.154.247",
  "folder": "inbox",
  "headers": "Return-Path: <bounce+1e9aa3.25af4b3-aa_ext_test02mg=comcast.net@test.
↪domain.com>\r\nDelivered-To: aa_ext_test02mg@comcast.net\r\nReceived: from dovdir3-
↪asa-01o.email.comcast.net ([96.114.154.247])\r\n\tby dovback3-asa-07o.email.comcast.
↪net with LMTP\r\n\tid uCAWG4gWIl6IDgAAVWOgEw\r\n\t(envelope-from <bounce+1e9aa3.
↪25af4b3-aa_ext_test02mg=comcast.net@test.domain.com>)\r\n\tfor <aa_ext_
↪test02mg@comcast.net>; Fri, 17 Jan 2020 20:18:16 +0000\r\nReceived: from dovpxy-asc-
↪01o.email.comcast.net ([96.114.154.247])\r\n\tby dovdir3-asa-01o.email.comcast.net␣
↪with LMTP\r\n\tid 0CrqGogWIl7xTAAAwP0GGg\r\n\t(envelope-from <bounce+1e9aa3.25af4b3-
↪aa_ext_test02mg=comcast.net@test.domain.com>)\r\n\tfor <aa_ext_test02mg@comcast.net>
↪; Fri, 17 Jan 2020 20:18:16 +0000\r\nReceived: from resimta-po-40v.sys.comcast.net␣
↪([96.114.154.247])\r\n\t(using TLSv1.2 with cipher ECDHE-RSA-AES256-GCM-SHA384 (256/
↪256 bits))\r\n\tby dovpxy-asc-01o.email.comcast.net with LMTP id␣
↪OD4qI4UWIl5UQQAAyeh4YQ\r\n\t; Fri, 17 Jan 2020 20:18:16 +0000\r\nReceived: from␣
↪so254-22.mailgun.net ([198.61.254.22])\r\n\tby resimta-po-40v.sys.comcast.net with␣
↪ESMTP\r\n\tid sY4Iie8ttwt3TsY4MiR4WV; Fri, 17 Jan 2020 20:18:16 +0000\r\nX-CAA-
↪SPAM: F00000\r\nX-Xfinity-VAAS:␣
↪gggruggvucftvghtrhhoucdtuddrgedugedrtdekgdegudcutefuodetggdotefrodftvfcurfhrohhfihhlvgemucevohhmtgg
↪Xfinity-VMeta: sc=0.00;st=legit\r\nX-Xfinity-Message-Heuristics: IPv6:N;TLS=1;SPF=1;
↪DMARC=\r\nAuthentication-Results: resimta-po-40v.sys.comcast.net;\r\n\tdkim=pass␣
↪header.d=test.domain.com header.b=J/in82+r\r\nDKIM-Signature: a=rsa-sha256; v=1;␣
↪c=relaxed/relaxed; d=test.domain.com; q=dns/txt;\r\n s=k1; t=1579292296; h=Mime-
↪Version: Content-Type: Subject: From: To:\r\n Message-Id: Sender: Date: Content-
↪Transfer-Encoding;\r\n bh=bYuUQQIdEIrM6gyAxa1Xp4Nd0A2cWpdksHogCDsE+j8=; b=J/
↪in82+rjjHVoVLeZIlYl+9y7WFgUOcXlrt+P8gaGduSdCEc6MEWMmY8JHyI0X00OTOLRIqn\r\n␣
↪1me6suiWiv8F2ADgtK2H9PYwRNg5LomNBKn7j1UbdQP4C7oJ3eYtvA6DCA5KRkgsHOTHY+Kq\r\n /
↪S49D6ajqrN4ZyB+XTLnA5IN8ew=\r\nX-Mailgun-Sending-Ip: 198.61.254.22\r\nX-Mailgun-
↪Sid:␣
↪WyJlOThhZiIsICJhYV9leHRfdGVzdDAybWdAY29tY2FzdC5uZXQiLCAiMjVhZjRiMyJd\r\nContent-
↪Transfer-Encoding: quoted-printable\r\nReceived: by luna.mailgun.net with HTTP; Fri,
↪ 17 Jan 2020 20:18:09 +0000\r\nDate: Fri, 17 Jan 2020 20:18:09 +0000\r\nSender:␣
↪user@test.domain.com\r\nMessage-Id: <20200117201809.1.82C23D86DE20410C@test.domain.
↪com>\r\nX-Mailgun-Seed-Test-Id: 5e22167af8424f444ca6d8e2\r\nTo: aa_ext_
↪test02mg@comcast.net\r\nFrom: user@test.domain.com\r\nSubject:␣
↪testSubject\r\nContent-Type: text/html; charset=\"ascii\"\r\nMime-Version: 1.0",
  "message_id": "<20200117201809.1.82C23D86DE20410C@test.domain.com>",
  "time": "2020-01-17T20:18:08.8Z"
}
```


Field Explanation:

Name	Type	Description
address	string	The address used to check for a test message.
provider	string	The provider responsible for maintaining the address.
ip	string	The ip the test message was sent from.
folder	string	The folder the test meassage landed in.
headers	string	The headers attached to the test message when retrieved from the address
message_id	string	The unique identifier attached to the test message when it is sent.
time	string	The time in which the message arrived at the address.

CHAPTER 5

FAQ

5.1 Getting Started / Settings

5.1.1 Why not just use Sendmail + Postfix + Courier IMAP?

You can but you should be aware that there is a constant battle raging between good and evil (i.e., spam) in the email universe. In order to be on the 'good' side of that battle and get your email delivered, there are numerous things you need to do. You need to have the right authentication infrastructure and register your IP and Domain appropriately. Also, you need to have a history of email sending that complies with Mailbox Providers' rules in order to build a good reputation.

Moreover, if you are going to receive, store and host emails, you better be prepared for maintaining this orchestra of software, take care of backups, hardware failures, security patches and monitoring. Stop kidding yourself, it's not 1998 anymore. :-)

Here's a classic post, So You'd Like to Send Some Email (Through Code), from Jeff Atwood about all of the hurdles in order to properly send email, and that's just sending.

5.1.2 What are the differences between free and Flex plans?

Free plan (no credit card required):

- 5,000 messages/month are included
- There is a limit of 300 messages per day on the included sandbox domain
- Data retention for Logs and the Events API is 1 day
- You cannot create custom domains
- You can only send to Authorized Recipients; and there is a maximum of 5 Authorized Recipients

Flex Plan:

If you opt to enter your credit card, you'll receive 5,000 free messages per month for 3 months. After the 3 months is up, if you haven't opted to upgrade into one of our subscription plans, you'll continue on a Pay as you Go plan, in which you'd pay for any messages you send at a rate of $0.80/1,000 messages. Additionally, on flex plans you get:

- No daily sending limit
- You get 5,000 free messages a month for 3 months. If you send over this amount during the first 3 months, additional messages are charged at a rate of $0.80/1,000 messages
- Data retention for Logs and the Events API is 5 days
- You can create custom domains
- Use of Authorized Recipients is not required to send with custom domains; you may send to whomever you like!

If these limits are not enough for you, or if you need to talk to us about a custom contract, you can reach us at sales@mailgun.com.

You can find a full list of features on our pricing page.

5.1.3 Can I get multiple Domains and IP Addresses?

By default we give you one shared IP address. If you would like a dedicated IP address, simply click on the "Add Dedicated IP" button in the **IP Management** section of the Control Panel. You will need to add a credit card to your account first, though, if you have not already done so. If you want multiple IPs, you can contact our Support Team.

You can create up to 1,000 domains on a paid plan in the Control Panel or through the :ref:'Domains API <api-domains>'(Free accounts do not include the ability to create a custom domain).

5.1.4 Do I need a dedicated IP address?

It depends on various factors.

If you are sending a lot of email (greater than 50k per week), it is a good idea to have a dedicated IP in order to isolate your reputation. If you are sharing your IP, you are sharing your reputation with those other senders. In addition, ESPs limit the total volume per IP, per hour. If you are a high volume sender you should consider a pool of IPs. However, you will have trouble establishing your reputation if you are not sending enough volume consistently from an IP - in this case, a shared IP is preferred.

If your email sending is volatile with large spikes of volume, ESPs may assume those large spikes are spam. Also, if your overall volume is too low, they won't acknowledge your reputation. Generally, if you are sending less than 5,000 emails per day, a shared IP may be the right solution.

The other thing to consider is using separate IPs for your bulk and transactional mail. There are a couple reasons for this:

- Delivery of time-sensitive transactional emails may get queued behind a large batch of bulk/marketing emails.
- Your transactional mail will be affected by the reputation created by your bulk/marketing mail.

Mailgun's infrastructure mitigates some of the arguments for a dedicated IP address. First of all, we are constantly monitoring our shared IP addresses for any reputation issues. We also allow you to schedule delivery of your emails by using the `o:deliverytime` parameter. This allows you to delay the delivery by using a time in the future and also allows you to jump other messages in your queue (say from a large bulk mailing) by using a delivery time of now.

5.1.5 How do I pick a domain name for my Mailgun account?

The name of an email domain matters most for receiving messages: If your domain name is `mycompany.com` it means you can receive messages sent to `xxx@mycompany.com`

Domain names do not matter as much if you're only sending. You can send messages from `sales@mycompany.com` even if your domain name is called `anothercompany.org`. Although, it is best for deliverability if you are using the same domain in the From field that the actual sender is using.

There are two types of domains you can configure with Mailgun:

- A sandbox subdomain of mailgun.org. Example: `sandboxXX.mailgun.org`. This option allows for quick testing, without having to setup DNS entries. This domain is provisioned automatically with every new account. But you can send only to authorized recipients.
- Your own domain like `mycompany.com`. This requires you to configure some records at your DNS provider. We provide you with those records and instructions in your Control Panel.

If your company's primary domain is `mycompany.com`, we recommend the following domain names for mailgun:

- `mycompany.com`, unless you're already using this name for your corporate email;
- `m.mycompany.com` or `mail.mycompany.com`;
- `mycompany.net` or `mycompany.org`.

Sometimes, it is a good idea to separate the domains for the type of messages you are sending. For example, some companies will use a different domains or subdomains for bulk marketing mailings and transactional or corporate mail in order to keep the reputations separate.

Finally, if you want multiple addresses and you want to direct certain emails to certain IP addresses, you will need to have a unique domain or subdomain for each IP address. In this situation, it's best to contact our Support Team to discuss your infrastructure.

5.1.6 Can I use the same domain name for Mailgun and for Google Apps (or another email server)?

Yes, for sending. No, for receiving. Only one email server can receive messages for a given domain name. It could be either Mailgun or Google servers, but not both. However, you can use the same domain for sending at multiple servers. If you'd like to register your Domain at multiple servers for sending but you don't want to receive email at Mailgun, just don't configure your MX records to point to Mailgun.

If you are receiving emails elsewhere with your domain, we recommend using a subdomain at Mailgun so you can also receive emails at Mailgun. This helps improve deliverability and allows us to more easily deal with any issues that arise with recipient email servers.

5.1.7 Can I rename a domain?

No, you need to create a new one and delete the old one. It's a good idea to create the new one first.

5.1.8 What if I need multiple SPF records?

If you are using multiple email servers and you want an SPF record for each of them, you should NOT set up a separate TXT record for each. You need to include the different servers in the same record. Below is sample syntax:

'v=spf1 include:myemailserver.com include:mailgun.org ~all'

5.1.9 How do I know if my DNS records are set up correctly

We have a "Check DNS Records Now" button when you click on a domain in the `Domains` tab that will confirm that they are set up correctly and, if not, show the incorrect records in red.

You could also use dig in your command line interface.

5.1.10 Do you support SSL/TLS?

Only TLS version 1.2 is supported. Support for SSL has been dropped due to the POODLE security vulnerability.

5.1.11 Ok, everything is set up, how do I start using Mailgun?

Mailgun is primarily a developer's tool so the best way use Mailgun is through our APIs. They are quite RESTful and we've tried to make them as intuitive as possible. Our Quickstart Guide is a good place to start and you can also use the API Reference for more detail. We also expose a lot of the features through the Control Panel. The User Manual is a good place to get a full overview of all of the capabilities of Mailgun.

5.2 Sending

5.2.1 Should I use SMTP or the HTTP API?

It's really up to you. Whatever you find easier is fine with us. The HTTP API has some advantages, however. First of all, it's faster. Second, we think it's easier to use - you don't have to deal with MIME because we will assemble it on our side. Just use a request library available for your language of choice.

5.2.2 Email clients say "sent via mailgun.us" with messages I send. How do I get rid of this?

Check the following:

- You have a custom domain defined in the `Domains` tab of the Control Panel.
- You've setup the DKIM DNS record (provided in the Control Panel, `Domains` tab).
- You're authenticating (SMTP) or posting (API) against the custom domain. (e.g. https://api.mailgun.net/v3/youcustomdomain.com/messages)

If you're still seeing "via mailgun.org", please contact our Support Team and we'll investigate.

5.2.3 What is the difference between the "From" and "Sender"

Each message you send out has both the sender and from address. Simply put, the sender domain is what the receiving email server sees when initiating the session, and the from address is what your recipients will see. For better deliverability it is recommended to use the same from domain as the sender, but it is not required.

You can technically set the from field to be whatever you like. The sender must always be one of your Mailgun domains.

5.2.4 Where do I specify BCC recipients?

BCC functionality works like this: specify a BCC recipient in the recipients list when sending, but do not include their address in the "To" or "CC" fields. You could also use the API, which has a specific BCC parameter.

5.2.5 How do I send the same message to multiple users using Mailgun?

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.

See the *Batch Sending* section of the *User Manual* for more information.

5.2.6 I am getting timeouts when connecting via SMTP. Why?

Most often, this is caused by internet service providers ("ISP") blocking port #25. This tends to happen if you are using a residential ISP.

To check this, try running telnet in command line:

```
telnet smtp.mailgun.org 25
```

If port 25 is not blocked, you should see something like this:

```
Trying 174.37.214.195...
Connected to mxa.mailgun.org.
Escape character is '^]'.
220 mxa.mailgun.org (Mailgun)
```

If you don't see this, then you are being blocked. There are a couple workarounds:

- Send using our HTTP API
- Try using port #587 or #2525

5.2.7 I have multiple domains at Mailgun. How do I tell Mailgun which domain to send mail from?

For SMTP, you have an SMTP username and password for each domain you have registered at Mailgun. To send mail from a particular domain, just use the appropriate credentials. For the API, the domain is one of the parameters in the URI.

5.2.8 I just submitted a lot of messages. Why is delivery happening so slowly?

There are many factors that can affect the speed of delivery. 1. Your established reputation for the domain and IPs on your account. 2. The total number of IPs allocated to your account. 3. The content quality for the emails being sent.

For newly allocated IPs, Mailgun protects and improves the reputation by gradually increasing sending rates. This means, as time passes, with high quality traffic, being sent from your IPs, your sending rates will increase automatically. If you're seeing slow delivery, please contact us… We'll evaluate your account configuration to ensure it is configured for handling the volume you require.

5.3 Deliverability / Reputation

5.3.1 If I'm just starting to send mail, how do I build a good reputation?

The way to think about your email reputation is much like your credit score. When you haven't sent any email, you don't have a bad reputation but you don't have a good one, either. Also, no ESP is going to allow you to send a million emails to their mailboxes, much like no one is going to give you a credit card with a huge credit limit when you graduate from college. There needs to be a history of performance for you to create a reputation. We use algorithms for our new senders that automatically queues your email and sends them at rates that makes the ESPs happy, increasing those rates as your sending reputation grows.

Some of the factors that help you build a good reputation faster and increase deliverability are:

- Limited spam complaints and bounces.
- Including the ability for recipients to unsubscribe.
- Recipients interacting with your emails in a good way: reading, replying, forwarding and adding your addresses to their contacts.
- Following ESPs' guidelines on sending rates.
- Paying attention to ESPs' feedback to slow or stop sending for a period of time.
- Having good content (see below for more guidance on content).

Also, consider letting your users to reply to your emails. Having a meaningful email conversations with your audience will do wonders for your reputation as a member of email community.

5.3.2 Does the content of my email matter for deliverability?

Absolutely. Ideally, you send email that people want. That's over half the battle. In addition, you should make your content interesting and relevant to the recipient.

There are a few things to keep in mind about your email content. First, we suggest setting up a test mailbox at http://www.mail-tester.com. Mail-Tester will provide you with a full analysis of your email for free. Here are some other things to consider:

- Personalize your emails. Make sure to include the recipient's address in the "To:" field and include his/her name in the greeting.
- It is best to send multi-part emails using both text and HTML or text only. Sending HTML only email is not well received by ESPs. Also, remember that ESPs generally block images by default so HTML only will not look very good unless users are proactive about enabling images.
- Test how your html email looks across all email clients and browsers. Litmus and Return Path have tools to do this.

- Make your content relevant and targeted to the recipient. There are even tools like Movable Ink that let you dynamically update your content after it is delivered.
- The higher the text to link and text to image ratios, the better. Too many links and images trigger spam flags at ESPs.
- Misspellings, spammy words (buy now!, Free!) are big spam flags, as are ALL CAPS AND EXCLAMATION MARKS!!!!!!!!!!!!
- The from field in your emails should match the domain you are sending from. Hotmail is particularly focused on this.
- Make sure you are using unsubscribe links and headers in your emails. Many ESPs (particularly Hotmail) pay attention to this and if they are not there, you are likely to get filtered. You can always use Mailgun's auto unsubscribe handling if you don't want to deal with this on your end.
- Include your physical mailing address. CAN-SPAM requires an unsubscribe link and a physical mailing list. It is also a good idea to provide a link to your privacy policy.
- Gmail pays particularly close attention to Message ID and Received headers. Message IDs that are formed incorrectly (without brackets <> and with wrong domain after @) can make Gmail think you are a spammer. The simplest way to create the right Message ID is to not set Message ID at all. Then Mailgun will create a perfect Message ID for you. Also, if you use the HTTP API, Mailgun will deal with all of this for you.
- Links should include the domain that is sending the email. Also, popular url shorteners can be a bad idea because they are frequently used by spammers.
- Long links may cause bounces. Some ESPs will block emails with links (or any consecutive text) longer than 99 characters.
- A/B test your emails to optimize recipient engagement. Subject lines are particularly important. You can use Mailgun's tagging and tracking statistics in order to measure A/B testing and improve your content.

5.3.3 Should I use my primary corporate domain name to send email?

You can, but remember that your reputation is tied to your domain name as well as the IP address. If you are in danger of being classified as a 'bad' sender of email, you will be affecting your domain reputation, which is very hard to recover from. It may be safer to use a completely separate domain (not a subdomain of your primary corporate domain) for sending marketing or even transactional email if you are worried about issues with domain reputation.

5.3.4 Why does the amount of email I send matter?

Rate limiting allows ESPs proper time to process and filter spam and ensure that transactional email doesn't get backed up. Without rate limiting in place, ESPs would be even more overwhelmed than they already are. The ESPs all have different sending limits on a per hour, per day basis. Once you hit thresholds with the rate limits, send too much spam, or have any number of other issues, the ISP may start returning error messages. Some ESPs will want you to slow down the sending, stop sending for a period of time, or change your habits (due to bad engagement, bad reputation, etc). We automatically adjust your sending rates according to the feedback from these ESPs to keep you in their good graces.

Generally, these rate limits are on a per IP address basis. Contact our Support Team if you wish to purchase additional dedicated IP addresses for your account.

5.3.5 Does the amount of email I send from my IP affect my deliverability?

Yes. Generally speaking, you don't want too few IPs, in case you experience more volume than you expect and you don't want so many IPs that you look suspicious or spread out your volume over too many IPs. There has to be a balance of volume to IP/domain. Sending too much volume from an IP, sending from too many IPs or sending too little from a range of IPs can all lead to deliverability issues.

5.3.6 Where can I learn more about Deliverability and Email?

One of the best resources is the blog Word to the Wise. Also, Return Path is a service that enhances deliverability and they publish a lot of great information through their blog and white papers. Below is are some best practices from the major ESPs.

- AOL Best Practices
- Gmail Best Practices
- Hotmail Best Practices
- Yahoo Best Practices

5.4 Receiving

5.4.1 Do you provide spam filtering for incoming mail?

Yes. Click on your domain in the Control Panel and enable our spam filtering service.

5.4.2 How do you handle quotations from replies and signatures when receiving mail?

We parse them and provide parameters for you to handle them as you wish. Please take a look at our *User Manual* or *API Reference* to see more details on the parameters we provide.

5.4.3 Why am I not receiving an email when sending via the route with the sending address as a destination?

You're most likely using GMail for sending your message. From GMail's documentation (https://support.google.com/mail/troubleshooter/2935079?rd=1):

Finally, if you're sending mail to a mailing list that you subscribe to, those messages will only appear in 'Sent Mail.' This behavior also occurs when sending to an email address that automatically forwards mail back to your Gmail address. To test forwarding addresses or mailing lists, use a different email address to send your message.

When a message from, say, `bob@gmail.com` goes through a route:

```
test@mailgun-domain.com -> bob@gmail.com
```

When this message arrives to GMail, it will have `bob@gmail.com` as both sender and recipient, therefore GMail will not show it.

In other words GMail does not show you messages you sent to yourself.

The other possibility is that the address had previously experienced a Hard Bounce and is on the 'do not send' list. Check the `Suppressions` tab of your Control Panel for a list of these addresses and remove the address in question if it is there.

5.4.4 How do I know if HTTP POST callbacks are coming from Mailgun, and not forged?

Mailgun allows you to check the authenticity of its requests by providing three additional parameters in every HTTP POST request it makes. Please take a look at our webhooks documentation for more information.

5.4.5 How do I know if the sender of an email is spoofed?

There is no 100% guarantee. However, there are some good clues. Mailgun provides DKIM and SPF verification for incoming mail, which is shown in the MIME headers once spam filtering is enabled in the Control Panel. This way you can at least know if the message is coming from an authenticated server.

5.4.6 Can I use Mailgun for my personal email address?

It's not recommended. Honestly, there are plenty of hosted email services better suited for this than Mailgun: Gmail, Google Apps, Outlook, etc. Mailgun is meant to be a tool for developers and their applications.

5.5 Tracking

5.5.1 Can I use Mailgun to track what happens with my emails?

Yep, Mailgun tracks all of the typical events that occur with emails: Opens, Link Clicks, Bounces, Unsubscribes and Spam Complaints. We make that data available to you via the Control Panel or through the API. In addition, you can set up webhooks and we will post events to your URL. Take a look at our tracking documentation for more information.

5.5.2 What about Email List Management?

Mailgun does have features to help you with list management. First of all, we will not deliver again to recipients that have hard bounced, unsubscribed, or complained of spam. This is to maintain your email reputation. You can remove emails from these do not send lists if it was a temporary issue. You can always access this information via the API or Control Panel to update your lists.

5.5.3 What is the difference between hard and soft bounces and how do you handle them?

You can think of hard bounces like permanent errors and soft bounces as temporary errors. We will stop attempting delivery after one hard bounce. With soft bounces, we keep trying to deliver but eventually we will stop trying to delivery in accordance with the receiving ESP's feedback.

5.5.4 Do I control the unsubscribe handling or do you?

It's up to you. You can use Mailgun's unsubscribe handling. You can include our unsubscribe variables: `%unsubscribe_url%` (for the entire domain) and `%tag_unsubscribe_url%` (for just emails with this tag) and we will take care of the unsubscribe handling for you. Take a look at our unsubscribe documentation for more information.

5.5.5 How do I create Campaigns in Mailgun?

It's very simple, just tag your emails with the appropriate `o:tag` parameter and Mailgun will group all of the events that occur to emails with that tag. Our analytics reports include those tags as one of the dimensions by which you can view and filter data. You can have multiple tags per email and up to 4,000 total tags. Take a look at our tagging documentation for more information.

5.5.6 Do you support A/B testing?

Since creating a campaign is as easy as including an arbitrary tag, yes. You can easily view which campaign is performing best by viewing the data grouped by tag in the `Analytics` tab of the Mailgun control panel.

5.5.7 How do I track which email a recipient has replied to?

This has been a popular question, so we wrote a blog post about it. Basically, the Message-ID in the original email is included in the In-Reply-To header in the reply email. So you can use that to track which specific email was replied to. Mailgun will automatically include a unique Message-ID or you can set your own.

CHAPTER 6

Email Best Practices

This guide is a brief summary of email best practices that we have learned from managing mail servers for thousands of customers and sending (and receiving) a lot of email. The objective is to help outline what you need to do to have your emails delivered whether or not you use Mailgun. Of course, if after reading the guide you decide that you have better things to do than maintaining email servers and managing email deliverability, we'd love to help!

In this guide, we will not focus much on the content of email messages. Content can be paraphrased with a few words: 'send something people want' (to paraphrase Y Combinator's motto). Creating good content is probably the hardest part, so apologies for the huge caveat. We focus on the infrastructure and monitoring of email so that if you are sending something people want, they will get it; and if you are not sending something people want, you will know about it and hopefully change that.

6.1 Reputation

One of the most important assets you have in the email world (much like the real world) is your reputation. If you do not have a good reputation tied to your domain and your IP address ("IP" used herein for abbreviation), your email will not reach your recipients' inboxes. Due to its popularity and its unique ability to push information to users, email has been overrun with spammers (as if you didn't notice). Depending on your definition, approximately 90% of all email is spam (source: MAAWG). Due to this, email service providers ("ESPs") like Gmail, AOL, Yahoo and MSN/Hotmail have declared an all-out war on spammers. This has made our inboxes a more pleasant place. This also makes it very important to manage your email reputation. If it is not impeccable, you will get caught in the ESPs' spam filters.

A good analogy for your email reputation is your personal credit score. Obviously, a bad reputation will hurt you. However, not having a reputation will also hurt you. If ESPs don't know you (or more specifically your IP and domain) they will assume the worst and filter you, at least initially. It's tough to blame them given all the spam out there. Due to the importance of reputation, a significant portion of our discussion on best practices revolves around building and maintaining your email reputation.

Our goal with respect to your email reputation is to make sure that the infrastructure is optimized for emails reaching the inbox and doesn't get in your way. We test all of our IPs' reputation before we allocate them and we use the authentication methods that major ESPs require.

Beyond making sure that the infrastructure is properly set up, we also provide the tools to answer some important questions:

- Are emails being delivered and if not, why?
- Is a recipient ESP throttling your traffic and why?
- Are messages bouncing due to incorrect domains or stale addresses?
- Are recipients unsubscribing or complaining of spam?
- Are recipients engaging with your emails by opening them and/or clicking on links?

You should use all of this data to make sure that you are complying with ESPs guidelines and adjust your email sending to stay in their good graces.

We give you all the tools for establishing a good sending reputation, but it's ultimately up to you to send emails appropriately. Some email service providers use F.U.D. about email deliverability to sell you a deliverability fairy that magically gets your emails to the inbox. This is most definitely not the case and your actions, as the email sender, play the biggest part in good deliverability.

However, if you follow a couple rules (along with properly authenticating your email), you will most likely build up a great email sending reputation:

- Only send emails to people that have signed-up to receive them from your website/application/service and always first send a confirmation link to confirm their address is correct (aka, "double opt-in"); and
- Track your email and adjust your sending based on feedback from ESPs and recipients (eg., don't send additional emails to recipients that have unsubscribed or complained of spam).

6.2 Hosting

A brief note on hosting since hosting technology is changing so quickly, it will likely be out of date in a few months. Large, virtual cloud environments are generally not the best environments for email for a few reasons:

- The IP address should be static so that your domain(s) & IP address(es) build a reputation together. Also, some more strict recipients ESPs may require whitelisting your IP address. Unfortunately, these should be IPv4.

- The IP address and surrounding IP addresses should have a good reputation. This is rarely the case at large cloud environments due to their ease of use and lax monitoring (which is inviting to spammers).
- Mail Transfer Agents should ideally be on real (non-virtual) machines, optimized for I/O.

We host Mailgun mostly on dedicated servers. We do use cloud servers for some of the infrastructure (where it makes sense), but for most of Mailgun we like large, robust, dedicated machines.

We use dedicated IP addresses in large subnets and we do background checks and extensive testing on our IP addresses. Because they are in large continuous blocks, they are less likely to be affected by other, external IP addresses. ESPs and blacklists occasionally block entire subnets if any of the IPs have questionable reputations. So even if your IP is clean, it might be blocked because of surrounding IPs. Larger subnets mitigate this risk.

We dream of a day when IP reputation does not matter and we can rely on domain reputation, but unfortunately we are not there yet.

6.3 IP Addresses and Sending Volume

If you are sending a lot of email (greater than 50k per week), it is a good idea to have a dedicated IP in order to isolate your reputation. If you are sharing your IP, you are sharing your reputation with those other senders. In addition, ESPs rate limit your emails based on the IP. So if you are a high volume sender you should consider getting a pool of IPs. However, your reputation can also be hurt if you are not sending enough volume consistently from an IP so it's a tricky balance.

If your email sending is volatile with large spikes of volume, ESPs may assume those large spikes are spam. Also, if you overall volume is too low, they won't acknowledge your reputation. Generally, if you are sending less than 5,000 emails per day, a shared IP may be the right solution.

The other thing to consider is using separate IPs for your bulk and transactional mail if you are sending high volumes of email. There are a couple reasons for this:

- Delivery of time-sensitive transactional emails may get queued behind a large batch of bulk/marketing emails.
- Your transactional mail will be affected by the reputation created by your bulk/marketing mail.

Even if you have a clean IP address, you need to warm up the IP gradually. This means sending emails at a low rate initially and then gradually increasing that rate, taking into account ESP feedback. If you send a ton of emails right away, they will get filtered or dropped by the ESPs. In some cases, they won't even tell you they are dropping them.

Mailgun offers both shared and dedicated IPs. We are constantly monitoring the traffic on these IPs. So even for shared IPs, you can be comfortable that your reputation is not being unduly influenced by others. We also offer pools of IPs for high volume senders. In addition, we have queuing algorithms that gradually warm up your IPs. Our sending rates automatically increase over time as your IP warms up. Finally, we separate our sending queues for each domain you set up at Mailgun, which mitigates the need for multiple IPs for different types of traffic.

6.4 DNS

Your email reputation is not only tied to your IP, but your domain name as well. You should keep this in mind as you set up your email infrastructure. For the same reasons as above, It is a good idea to have separate domains or subdomains for your marketing, transactional and corporate mail. We suggest that you use your top level domain for your corporate mail and using different domains or subdomains for your marketing and transactional mail.

While it is not required to use the same domain in the From field of the message as the actual domain sending the message, it is highly recommended. Hotmail is especially finicky about this requirement and has a higher propensity to filter your messages to junk if the two domains do not match.

You should also make sure that you are using a well regarded DNS provider and that you publish all of your contact information in the WHOIS record. If you are hiding your contact information through a proxy, ESPs may take that as a signal that you are spamming.

Also, make sure you include the appropriate records at your DNS provider for authentication (see below). While it's not required to point mx records to the same domain as you are sending from, it is recommended. There are email providers (albeit, a minority) that will check if mx records for the domain are valid before accepting email.

Mailgun gives you the ability to create multiple domains or subdomains very easily. You are free to create multiple domains and subdomains for each of your transactional, marketing and corporate email. Each domain has an isolated queue, so your transactional emails won't get held up by your bulk mailings.

6.5 Authentication

It is very important that you are using the appropriate authentication methods with your email. If you are not authenticating your email properly, ESPs will assume you are spamming and will filter or just drop your email.

The common types of authentication are:

- SPF
- DKIM
- DomainKeys
- SenderID

Mailgun uses all of these types of authentication. When you sign up for Mailgun, we provide the appropriate records for you to include at your DNS registrar. We also provide a verification button you can use to make sure that your records are set up correctly.

6.6 Mailing Lists

The number one reason we see people get blocked is because they have a bad mailing list. **Don't purchase your list or scrape websites for emails.** It's the easy way out and you will pay the consequences. Most of these lists have bad email addresses and include spam traps. ESPs are very good at recognizing bad mailing lists.

You should only send emails to people that have opted in to receiving your emails on YOUR website. In addition, you should be sending a verification email with a link that confirms their subscription (double opt-in) to make sure their email address is correct and that they are the person that signed up. If everyone did this, the world would be a better place.

You should have your Privacy Policy easily accessible on your website. In addition, you should have a place on your website where users can unsubscribe from your mailings, in addition to a link in every email you send (see *Unsubscribe Handling*).

While we have to rely on you to be responsible about how you procure your mailing list, we do track and give you data to easily see how your emails are being received. We give you information for bounces, unsubscribes, complaints, opens and clicks so that you can modify your mailing lists appropriately. In addition, we automate a lot of the work by keeping track of recipients that have unsubscribed, bounced or complained and stopping future deliveries to those recipients. We give you various levels of unsubscribe granularity so your recipients can unsubscribe to all emails from the domain, just that mailing list or just emails with that "tag" (which you define).

6.7 Bounce and ESP Feedback Handling

A big part of maintaining your email reputation is processing bounces properly. While most major ESPs give bounce replies "on the wire" during the SMTP session, there are some that send bounce messages via email. In order receive these emailed bounce messages, you must have the appropriate return path header included with your email so that recipients know where to reply with bounce information.

You must also process this bounce data and act accordingly. In addition, many ESPs will soft bounce your initial attempts at delivery. This is also called grey-listing or throttling. If you continue to send emails to bad addresses or you do not listen to ESPs feedback, you will get filtered and eventually your emails will just get dropped.

Mailgun automatically processes bounce information and reacts accordingly. A good portion of Mailgun's technology is devoted to the parsing of this feedback and adjusting your sending in accordance with this feedback so that you maintain a good reputation.

If we receive a hard bounce, we will stop sending to that address immediately and will not attempt future deliveries to that address. We will stop sending to an address after multiple soft bounces, according to the ESPs' guidelines. It is possible to remove addresses from the flagged list in your Control Panel or through the API, in case it was a temporary issue.

Please see our *User Manual* for more information.

6.8 Feedback Loops and Spam Complaints

Most of the major ESPs (other than Gmail) provide feedback loops through which they give you information about spam complaints. Here is a thorough list from Word to the Wise. It is important that you sign up for these feedback loops and pay attention to the feedback you are getting. If you ignore this feedback, ESPs will throttle you and eventually block you completely.

We register all of our IPs for these feedback loops. You can access this information through the Control Panel, the API or Webhooks. In addition, we process spam complaints automatically and will stop sending to email addresses after a recipient complains. It is possible to remove addresses from the flagged list in your Control Panel or through the API.

Please see our *User Manual* for more information.

6.9 Unsubscribe Handling

It is important to give your recipients the ability to unsubscribe from emails. First, it is required by the CAN-Spam Act. Second, if you don't give them this option, they are more likely to click on the spam complaint button, which will cause more harm than allowing them to unsubscribe. Finally, many ESPs look for unsubscribe links and are more likely to filter your email if they don't have them.

Mailgun gives you the ability to include an unsubscribe link or email automatically in your email. We give you the ability to link the unsubscribe to a certain campaign, mailing list or make the request global to your domain. You can access this data through the Control Panel, API or via Webhooks. In addition, we will automatically stop sending to email addresses that have unsubscribed. It is possible to remove addresses from the flagged list in your Control Panel or through the API.

Please see our *User Manual* for more information.

6.10 Recipient Engagement

In addition to processing bounces, complaints and unsubscribes, ESPs measure your reputation through the engagement of your recipients. If recipients are opening, forwarding and replying to your emails, it will improve your reputation. This is what makes 'do-not-reply' emails so offensive. At many ESPs, it is also helpful if recipients add your email address to their address books.

Mailgun allows you to track opens and link clicks with our Tracking and Tagging functionality (see our *User Manual* for more information). You are free to create up to 4,000 tags and use them simultaneously for A/B testing. In addition, Mailgun is built to receive and parse emails efficiently. So there is no excuse to not allow your recipients to reply to your emails. Email is not a billboard - it is a conversant technology.

6.11 Email Content

There are a few tricks to remember about content besides the mantra of 'sending something people want'. As mentioned above, you can set up a test mailbox at Mailgun and enable our spam filters to receive a "spamicity" score to test how your content is being judged by spam filters.

- Personalize your emails to each recipient. Ideally, the content should reflect recipient's specific interests or usage patterns in your application. At least address them by their name. . . don't be rude!. Mailgun has recipient variables that you can define and use with your email templates to achieve detailed levels of personalization.
- It is best to send multi-part emails using both text and HTML or text only. Sending HTML only email is not well received by ESPs. Also, remember that ESPs generally block images by default so HTML only will not look very good unless users are proactive about enabling images. There are a few tools available to test how your email will render across ESPs and browsers. Litmus offers one, as does Return Path.
- The higher the text to link and text to image ratios, the better. Too many links and images trigger spam flags at ESPs.
- Misspellings, spammy words (buy now!, Free!) are big spam flags, as are ALL CAPS AND EXCLAMATION MARKS!!!!!!!!!!!!!
- The domains in the from field, return-path and message-id should match the domain you are sending from.
- Make sure you are using unsubscribe links and headers in your emails. Many ESPs (particularly Hotmail) pay attention to this and if they are not there, you are likely to get filtered. You can always use Mailgun's auto unsubscribe handling if you don't want to deal with this on your end.
- Gmail pays particularly close attention to Message ID and Received headers. Message IDs that are formed incorrectly (without brackets <> and with wrong domain after @) can make Gmail think you are a spammer. The simplest way to create the right Message ID with Mailgun is to not include one. Then Mailgun will create a perfect Message ID for you.
- Links should include the domain that is sending the email. Also, popular url shorteners can be a bad idea because they are frequently used by spammers.
- A/B test your emails to optimize recipient engagement. Subject lines are particularly important. You can use Mailgun's tagging and tracking statistics in order to measure A/B testing and improve your content.

Best of luck with your emailing. . . we hope we made it easier!

Index

R