

[sign up or log in](#)

find packages

★ mailgun-js public

Simple Node.js helper module for Mailgun API



npm install mailgun-js
8 dependencies version 0.7.10
36 dependents updated 9 days ago
13 ★ 52,881 downloads in the last month



Installation

```
npm install mailgun-js
```


Usage overview

Please see [Mailgun Documentation](#) for full Mailgun API reference.

This module works by providing proxy objects for interacting with different resources through the Mailgun API. Most methods take a `data` parameter, which is a Javascript object that would contain the arguments for the Mailgun API. All methods take a final parameter callback with two parameters: `error`, and `body`. We try to parse the body into a javascript object, and return it to the callback as such for easier use and inspection by the client. If there was an error a new Error object will be passed to the callback in the error parameter. If the error originated from the (Mailgun) server, the response code will be available in the `statusCode` property of the error object passed in the callback. See the `/docs` folder for detailed documentation. For full usage examples see the `/test` folder.

npm Enterprise

Host your own private, on-premises npm registry. Free 30-day trial. [Learn more...](#)

 `npm install mailgun-`
[how?](#) [learn more](#)

 [bojand](#) published a wee...

0.7.10 is the latest of 53 rele...

github.com/bojand/mailgun...

MIT 

Collaborators



Stats

1.841 downloads in the last ...

12.595 downloads in the las...

51.076 downloads in the las...

[5 open issues](#) on GitHub

```
var api_key = 'key-XXXXXXXXXXXXXXXXXXXXXXX';
var domain = 'mydomain.mailgun.org';
var mailgun = require('mailgun-js')({apiKey: api_k

var data = {
  from: 'Excited User <me@samples.mailgun.org>',
  to: 'serobnic@mail.ru',
  subject: 'Hello',
  text: 'Testing some Mailgun awesomness!'
};

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



Messages stored using the Mailgun `store()` action can be retrieved using `messages(<message_key>).info()` function. Optionally the MIME representation of the message can be retrieved if `MIME` argument is passed in and set to `true`.

Something more elaborate. Get mailing list info, create a member and get mailing list members and update member. Notice that the proxy objects can be reused.

No open pull requests on Git...

Try it out

Test mailgun-js in your ...

Keywords

mailgun, email

Dependencies (8)

debug, scmp, q, proxy-agent,
path-proxy, inflection, form-
data, async

Dependents

parse-server-transform,
@m1r4ge/parse-server,
nodebb-plugin-emailer-
mailgun, parse-server-hotfix,
hapi-email-kue, hapi-mailgun,
parse-server-leman-test-
transform, parse-server,
loopback-connector-mailgun,
nodemill, turmail, parse-server-
simple-mailgun-adapter,
sender-js, parse-server-
httpauth-fork, nodemailer-
mailgun-transport, scraper-
watcher, campaign-mailgun,
feathers-mailgun, bakat-
mailer, bunyan-mailgun,
winston-mailgun, kalabox-
email, express-boilerplate,
meshblu-mailgun,
machinepack-mailgun, ark-

```
var list = mailgun.lists('mylist@mycompany.com');

list.info(function (err, data) {
  // `data` is mailing list info
  console.log(data);
});

var bob = {
  subscribed: true,
  address: 'bob@gmail.com',
  name: 'Bob Bar',
  vars: {age: 26}
};

list.members().create(bob, function (err, data) {
  // `data` is the member details
  console.log(data);
});

list.members().list(function (err, members) {
  // `members` is the list of members
  console.log(members);
});

list.members('bob@gmail.com').update({ name: 'Foo'
  console.log(body);
});
```

mailer, jazz.email, sublayer,
beagle-heroku, clever-email,
jekyll-discuss, bip-pod-
mailgun, venn-messaging,
sinopia-htaccess-gpg-email,
textmail, digger-mailgun,
hubot-email-hashtag

[PaperG](#) is hiring. View
more...

Options

Mailgun object constructor options:

- `apiKey` - Your Mailgun API KEY
- `domain` - Your Mailgun Domain
- `mute` - Set to true if you wish to mute the console error logs in `validateWebhook()` function
- `proxy` - The proxy URI in format

`http[s]://[auth@]host:port.ex:`

`'http://proxy.example.com:8080'`

- `timeout` - Request timeout in milliseconds
- `host` - the mailgun host (default: '**api.mailgun.net**')
- `protocol` - the mailgun protocol (default: 'https:', possible values: 'http:' or 'https:')
- `port` - the mailgun port (default: '443')
- `endpoint` - the mailgun host (default: '/v3')
- `retry` - the number of **total attempts** to do when performing requests. Default is 1. That is, we will try an operation only once with no retries on error.

Attachments

Attachments can be sent using either the `attachment` or `inline` parameters. `inline` parameter can be use to send an attachment with `inline` disposition. It can be used to send inline images. Both types are supported with same mechanisms as described, we will just use `attachment` parameter in the documentation below but same stands for `inline`.

Sending attachments can be done in a few ways. We can use the path to a file in the `attachment` parameter. If the `attachment` parameter is of type `string` it is assumed to be the path to a file.

```
var filepath = path.join(__dirname, 'mailgun_logo.  
  
var data = {  
  from: 'Excited User <me@samples.mailgun.org>',  
  to: 'serobnic@mail.ru',  
  subject: 'Hello',  
  text: 'Testing some Mailgun awesomness!',  
  attachment: filepath  
};  
  
mailgun.messages().send(data, function (error, bod  
  console.log(body);  
});
```



We can pass a buffer (has to be a Buffer object) of the data. If a buffer is used the data will be attached using a generic filename "file".

```
var filepath = path.join(__dirname, 'mailgun_logo.  
var file = fs.readFileSync(filepath);  
  
var data = {  
  from: 'Excited User <me@samples.mailgun.org>',  
  to: 'serobnic@mail.ru',  
  subject: 'Hello',  
  text: 'Testing some Mailgun awesomness!',  
  attachment: file  
};  
  
mailgun.messages().send(data, function (error, bod  
  console.log(body);  
});
```

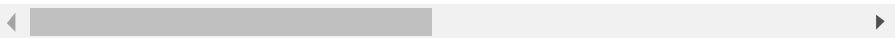


We can also pass in a stream of the data. This is useful if you're attaching a file from the internet.

```
var request = require('request');
var file = request("https://www.google.ca/images/b

var data = {
  from: 'Excited User <me@samples.mailgun.org>',
  to: 'serobnic@mail.ru',
  subject: 'Hello',
  text: 'Testing some Mailgun awesomness!',
  attachment: file
};

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



Finally we provide a `Mailgun.Attachment` class to add attachments with a bit more customization. The Attachment constructor takes an options object. The options parameters can have the following fields:

- `data` - can be one of
 - a string representing file path to the attachment
 - a buffer of file data
 - an instance of `Readable` which means it is a readable stream.
- `filename` - the file name to be used for the attachment. Default is 'file'
- `contentType` - the content type. Required for case of `Readable` data. Ex. `image/jpeg`.
- `knownLength` - the content length in bytes. Required for case of `Readable` data.

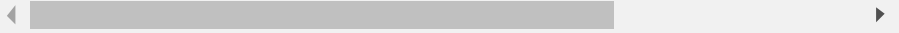
If an attachment object does not satisfy those valid conditions it is ignored. Multiple attachments can be sent by passing an array in the attachment parameter. The array elements can be of any one of the valid types and each one will be handled appropriately.

```
var mailgun = require('mailgun-js')({apiKey: api_k
var filename = 'mailgun_logo.png';
var filepath = path.join(__dirname, filename);
var file = fs.readFileSync(filepath);

var attch = new mailgun.Attachment({data: file, fi

var data = {
  from: 'Excited User <me@samples.mailgun.org>',
  to: 'serobnic@mail.ru',
  subject: 'Hello',
  text: 'Testing some Mailgun awesomness!',
  attachment: attch
};


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



```
var mailgun = require('mailgun-js')({apiKey: api_k
var filename = 'mailgun_logo.png';
var filepath = path.join(__dirname, filename);
var fileStream = fs.createReadStream(filepath);
var fileStat = fs.statSync(filepath);

msg.attachment = new mailgun.Attachment({
  data: fileStream,
  filename: 'my_custom_name.png',
  knownLength: fileStat.size,
  contentType: 'image/png'});

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



Sending MIME messages

Sending messages in MIME format can be accomplished using the `sendMime()` function of the `messages()` proxy object. The `data` parameter for the function has to have `to` and `message` properties. The `message` property can be a full file path to the MIME file, a stream of the file (that is a `Readable` object), or a string representation of the MIME message. To build a MIME string you can use the [Mail Composer] (<https://www.npmjs.org/package/mailcomposer>) library. Some examples:


```
var domain = 'mydomain.mailgun.org';
var mailgun = require('mailgun-js')({ apiKey: "YOU
var mailcomposer = require('mailcomposer');

var mail = mailcomposer({
  from: 'you@samples.mailgun.org',
  to: 'mm@samples.mailgun.org',
  subject: 'Test email subject',
  body: 'Test email text',
  html: '<b> Test email text </b>'
});

mail.build(function(mailBuildError, message) {

  var dataToSend = {
    to: 'mm@samples.mailgun.org',
    message: message.toString('ascii')
  };

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



Referencing MIME file

```
var filepath = '/path/to/message.mime';

var data = {
  to: fixture.message.to,
  message: filepath
};

mailgun.messages().sendMime(data, function (err, b
  console.log(body);
});
```



```
var filepath = '/path/to/message.mime';

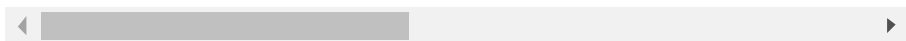
var data = {
  to: fixture.message.to,
  message: fs.createReadStream(filepath)
};

mailgun.messages().sendMime(data, function (err, b
  console.log(body);
});
```

Creating mailing list members

`members().create({data})` will create a mailing list member with data. Mailgun also offers a resource for creating members in bulk. Doing a POST to `/lists/<address>/members.json` adds multiple members, up to 1,000 per call, to a Mailing List. This can be accomplished using `members().add()`.

```
var members = [  
  {  
    address: 'Alice <alice@example.com>',  
    vars: { age: 26 }  
  },  
  {  
    name: 'Bob',  
    address: 'bob@example.com',  
    vars: { age: 34 }  
  }  
];  
  
mailgun.lists('mylist@mycompany.com').members().ad  
  console.log(body);  
});
```



Generic requests

Mailgun-js also provides helper methods to allow users to interact with parts of the api that are not exposed already. These are not tied to the domain passed in the constructor, and thus require the full path with the domain passed in the resource argument.

- `mailgun.get(resource, data, callback)` - sends GET request to the specified resource on api.
- `mailgun.post(resource, data, callback)` - sends POST request to the specified resource on api.
- `mailgun.delete(resource, data, callback)` - sends DELETE request to the specified resource on api.
- `mailgun.put(resource, data, callback)` - sends PUT request to the specified resource on api.

Example: Get some stats

```
mailgun.get('/samples.mailgun.org/stats', { event:  
  console.log(body);  
});
```



Promises

Module works with Node-style callbacks, but also implements promises with the **Q** library.

```
mailgun.lists('mylist@mydomain.com').info().then(f  
  console.log(data);  
, function (err) {  
  console.log(err);  
});
```



The function passed as 2nd argument is optional and not needed if you don't care about the fail case.

Webhook validation

The Mailgun object also has a helper function for validating Mailgun Webhook requests (as per the **mailgun docs for securing webhooks**). This code came from **this gist**.

Example usage:

```
var mailgun = require('mailgun-js')({apiKey: api_k

function router(app) {
  app.post('/webhooks/mailgun/*', function (req, r
    var body = req.body;

    if (!mailgun.validateWebhook(body.timestamp, b
      console.error('Request came, but not from Ma
      res.send({ error: { message: 'Invalid signat
      return;
    }

    next();
  });

  app.post('/webhooks/mailgun/catchall', function
    // actually handle request here
  });
}
```



Tests

To run the test suite you must first have a Mailgun account with a domain setup. Then create a file named `./test/auth.json`, which contains your credentials as JSON, for example:

```
{ "api_key": "key-XXXXXXXXXXXXXXXXXXXXXXX", "domai
```



You should edit `./test/fixture.json` and modify the data to match your context.

Then install the dev dependencies and execute the test suite:

```
$ npm install
```

```
$ npm test
```

The tests will call Mailgun API, and will send a test email, create route(s), mailing list and mailing list member.

Notes

This project is not endorsed by or affiliated with **Mailgun**. The general design and some code was heavily inspired by **node-heroku-client**.

License

Copyright 2012, 2013, 2014 OneLobby

Licensed under the MIT License.

You Need Help About npm Legal Stuff

Documentation	About npm, Inc	Terms of Use
Support / Contact Us	Jobs	Code of Conduct
Registry Status	npm Weekly	Package Name Disputes
Website Issues	Blog	Privacy Policy
CLI Issues	Twitter	Reporting Abuse
Security	GitHub	Other policies

npm loves you