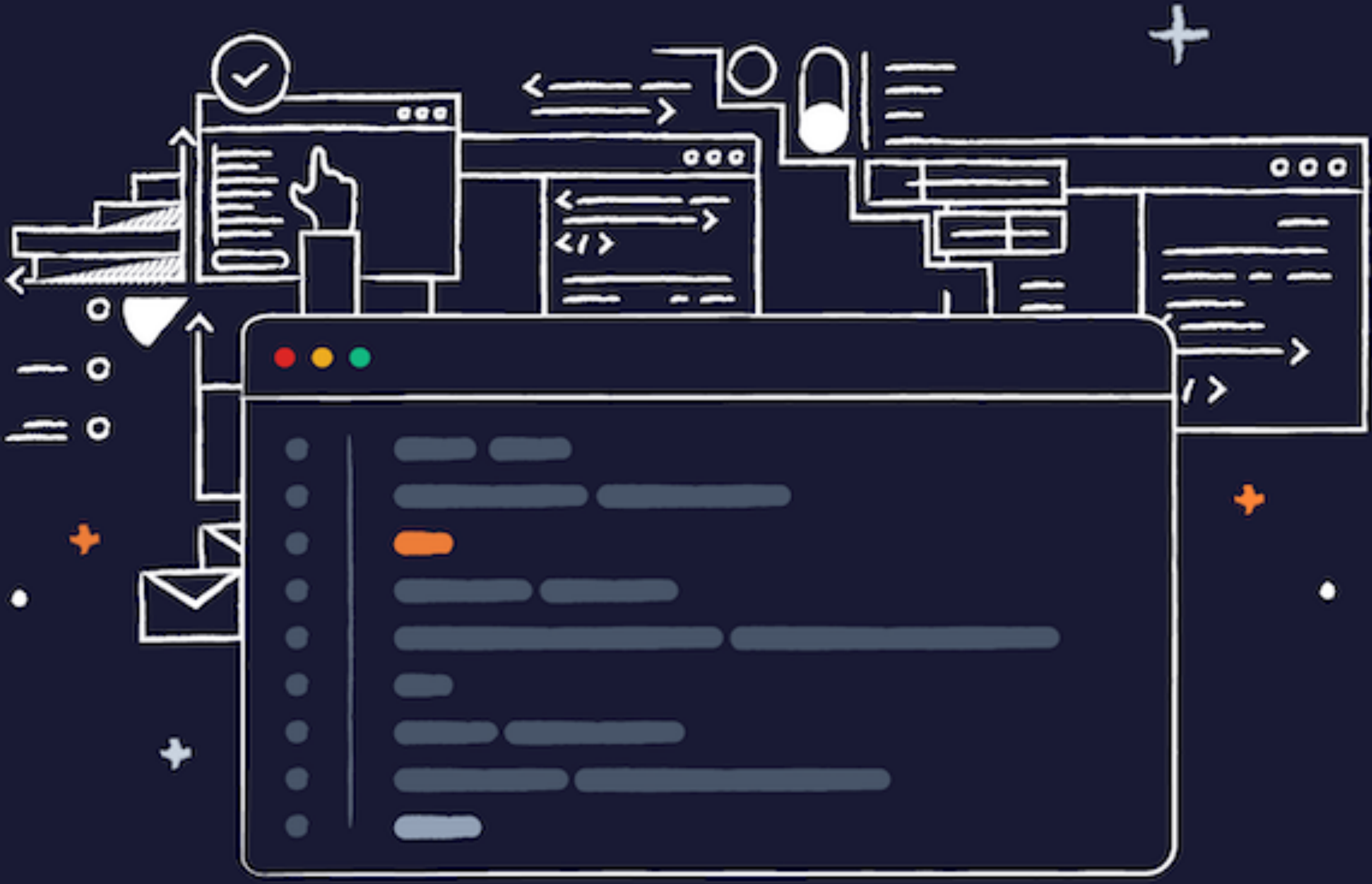
SocketLabs

SocketLabs Complex Sender API Quickstart Guide



Contents

- Authentication
- Creating an IP Pool
- Creating a Subaccount
- Subaccount Custom Bounce Domain
- Subaccount Authentication
- Subaccount Engagement Tracking
- Injection API
- SMTP
- Account Level Event Webhook
- Subaccount Event Webhook

Getting Started

Welcome to SocketLabs!

SocketLabs's Complex Sender product was built from the ground up with the indirect sender use case in mind. If you send mail on behalf of others or need intelligent mailstream separation, SocketLabs is the product for you!

API Features

- Integration is quick, lightweight, and can be done in one engineering sprint.
- Once initial integration steps are taken, the rest of the platform is optionally codeless.
- SocketLabs was built API first: tasks you can do in the application can also be done via API.

This guide provides overviews of the API calls you'll need to authenticate and perform a basic setup of your Complex Sender SocketLabs account. Ready? Let's get started!

No SocketLabs account? No problem! Click here to chat with our Sales team

Getting Started

API Conventions

- The SocketLabs API follows REStful conventions
- Trailing slashes are ignored: /transmissions is equivalent to /transmissions/.
- URL paths, URL query parameter names, and JSON field names are case insensitive.
- URL paths use lower case, with dashes separating words.
- Query parameters and JSON fields use camel casing for JSON field names.
- The HTTP status indicates whether an operation failed or succeeded, with extra information included in the HTTP response body.
- All APIs return standard HTTP error code formats.
- Unexpected query parameters and request body fields are ignored.

Authentication

Creating an API Key

To get started with the SocketLabs API, you'll first need to create an API key. This can be done on the API Key Management page in the SocketLabs Performance Dashboard.

It is important to keep your API key secure because it can be used to modify features for your SocketLabs account.

Endpoints

All calls to the SocketLabs API need to start with the appropriate base URL:

```
https://api.socketlabs.com/
```

Bearer Token

API calls to the SocketLabs API are authenticated using the API key that you generated. Authenticate your calls to the SocketLabs API using the Authorization header with the Bearer authentication scheme. That should look like:

```
Authorization: Bearer YOUR-API-KEY
```

Authentication

Example Request

```
var client = new HttpClient();

var request = new HttpRequestMessage(HttpMethod.Get,
    “https://api.socketlabs.com/v2/ip-allocation?” +
    “pageNumber=0&pageSize=5&sortField=IpAddress&sortDirection=asc” +
    “&filters=IpAddress=like:10.28.100”);

request.Headers.Add(“Authorization”, “Bearer <token>”);

var response = await client.SendAsync(request);
response.EnsureSuccessStatusCode();

Console.WriteLine(await response.Content.ReadAsStringAsync());
```

Now let’s go create an IP Pool.

Creating IP Pools

IP Pool Overview

IP Pools are how your subaccount mail gets routed. IP Pools are useful for helping separate mail streams and managing throttling for warmup.

Endpoint

```
https://api.socketlabs.com/
```

Attributes

Property Name	Data Type	Details
ipAssignmentIDs	string	Unique ID of each IP address to assign
name	string	IP Pool display name

Creating IP Pools

Get all IP Address Allocations

To create an IP Pool, you will need to know what IPs are available to be allocated to the pool. The response from this request will be used to create the IP Pool.

Example Request

```
var client = new HttpClient();

var request = new HttpRequestMessage(HttpMethod.Get,
    “https://api.socketlabs.com/v2/ip-allocation?” +
    “pageNumber=0&pageSize=5&sortField=IpAddress&sortDirection=asc” +
    “&filters=IpAddress=like:10.28.100”);

request.Headers.Add(“Authorization”, “Bearer <token>”);

var response = await client.SendAsync(request);
response.EnsureSuccessStatusCode();

Console.WriteLine(await response.Content.ReadAsStringAsync());
```

Let’s move on to creating new subaccounts.

Creating Subaccounts

Subaccount Overview

Subaccounts are how you provision, manage, and report on senders separately. Subaccounts are useful for any type of complex sender, whether you send on behalf of others or want to otherwise separate various mail streams.

Endpoint

```
https://api.socketlabs.com/subaccount
```

Attributes

One of the attributes below must be set to create a subaccount. If neither are set or both attributes are set, the API call will fail.

Property Name	Data Type	Details
name	string	Subaccount display name
ipPoolID	number	Unique ID of each IP address to assign
useRuleEngine	boolean	IP Pool display name

Creating Subaccounts

Example Request with ‘useRuleEngine’ set

```
var client = new HttpClient();

var request = new HttpRequestMessage(HttpMethod.Post,
    “https://api.socketlabs.com/v2/subaccount”);

request.Headers.Add(“Authorization”, “Bearer <token>”);

var content = new StringContent(@”{
          ‘name’: ‘Test Name’,
          ‘useRuleEngine’: true
}”, null, “text/plain”);

request.Content = content;

var response = await client.SendAsync(request);
response.EnsureSuccessStatusCode();

Console.WriteLine(await response.Content.ReadAsStringAsync());
```

Creating Subaccounts

Example Response with ‘useRuleEngine’ set

```
{
  “data”: {
    “subaccountId”: 12345,
    “accountId”: 98765,
    “name”: “My Subaccount”,
    “status”: “Active”,
    “createdOn”: “2022-01-01T07:00:00-05:00”,
    “streamScore”: 85,
    “dailyVolume”: 123456,
    “urgency”: “Low”,
    “hasDkim”: false,
    “hasSpf”: false,
    “useRuleEngine”: true,
  }
}
```

Creating Subaccounts

Example Request with ‘ipPoolId’ set

```
var client = new HttpClient();

var request = new HttpRequestMessage(HttpMethod.Post,
    “https://api.socketlabs.com/v2/subaccount”);

request.Headers.Add(“Authorization”, “Bearer <token>”);

var content = new StringContent(@”{
         ‘name’: ‘Test Name’,
         ‘useRuleEngine’: true
}”, null, “text/plain”);

request.Content = content;

var response = await client.SendAsync(request);
response.EnsureSuccessStatusCode();

Console.WriteLine(await response.Content.ReadAsStringAsync());
```

Creating Subaccounts

Example Response with ‘ipPoolId’ set

```
{
  “data”: {
    “subaccountId”: 12345,
    “accountId”: 98765,
    “name”: “My Subaccount”,
    “status”: “Active”,
    “createdOn”: “2022-01-01T07:00:00-05:00”,
    “streamScore”: 85,
    “dailyVolume”: 123456,
    “urgency”: “Low”,
    “hasDkim”: false,
    “hasSpf”: false,
    “assignedIpPool”: {
          “ipPoolId”: 12345,
          “accountId”: 98765,
          “name”: “My Ip Pool”,
          “status”: “Active”,
          “createdOn”: “2022-01-01T07:00:00-05:00”,
          “isDefault”: true,
          “streamScore”: 85,
          “dailyVolume”: 123456
    }
  }
}
```

After creating subaccounts, you’ll want security. First, we’ll start with custom bounce domain setup to enable SPF.

Subaccount Custom Bounce Domain Authentication

Bounce Domain

The Custom Bounce Domain is how SPF is implemented with SocketLabs.

Setup

Before you can set up a bounce domain with the API, that domain needs to exist and have CNAME set up to point to tracking.socketlabs.com. You can re-use the CNAME record created for custom bounce domain to set up your engagement tracking domain.

Endpoint

```
https://api.socketlabs.com/subaccount/:subaccountId/bounce
```

Path Variables

- subaccountId (number, required)

Subaccount Custom Bounce Domain Authentication

Attributes

Property Name	Data Type	Details
domain	string	Domain or subdomain you want to set up for custom bounce domain handling.
isDefault	boolean	If this is set as the default, any mail that is sent that does not match a custom bounce domain in the account will fall back to using this domain. If no defaults are set in your account, email-od.com is used as the default value.

Subaccount Custom Bounce Domain Authentication

Example Request

```
var client = new HttpClient();

var request = new HttpRequestMessage(HttpMethod.Post,
    “https://api.socketlabs.com/v2/subaccount/:subaccountID/bounce”);

request.Headers.Add(“Authorization”, “Bearer <token>”);

var content = new StringContent(@”{
   “domain”: “do”,
   “isDefault”: true
}”, null, “text/plain”);

request.Content = content;

var response = await client.SendAsync(request);
response.EnsureSuccessStatusCode();

Console.WriteLine(await response.Content.ReadAsStringAsync());
```

Subaccount Custom Bounce Domain Authentication

Example Response with ‘useRuleEngine’ set

```
{
  “data”: {
    “domain”: “example.com”,
    “isDefault”: false,
    “validationResult”: “Success”,
    “createdOn”: “1950-07-03T00:39:30.153Z”,
    “updatedOn”: “2008-09-07T16:07:17.709Z”
  }
}
```

You’ve got SPF enabled, so now it’s time to set up subaccount DKIM authentication.

Subaccount DKIM Authentication

DKIM Overview

SocketLabs will sign outbound messages where the From address domain matches the domain of one of your DKIM entries.

To create a new DKIM entry, please provide the domain, selector, and private key. We will attempt to validate the private key that you provide against the public key for your domain and selector. If we are unable to validate the public key, an error message will be returned.

You can create your own private and public keys to use with this endpoint. If you would prefer to have SocketLabs generate the key pairs for you, we provide a Generate end-point that will generate a public/private key pair for your provided domain and selector combination.

Options

DKIM entries can be set up by having SocketLabs generate a private and public key pair or by providing SocketLabs directly with the public/private key pair. The private key must exist on the domain you are setting up in order for it to pass validation.

Endpoint

```
https://api.socketlabs.com/subaccount/:subaccountId/dkim
```

Subaccount DKIM Authentication

Path Variables

- subaccountId (number, required)

Generate a DKIM Key for a Subaccount

Attributes

Property Name	Data Type	Details
domain	string	Domain or subdomain you want to set up for your DKIM entry
selector	string	Selector for the DKIM record

Example Request

```
var client = new HttpClient();

var request = new HttpRequestMessage(HttpMethod.Post,
    "https://api.socketlabs.com/v2/subaccount/:subaccountID/dkim?do-
main=:domain&selector=:selector");

request.Headers.Add("Authorization", "Bearer <token>");

var response = await client.SendAsync(request);
response.EnsureSuccessStatusCode();

Console.WriteLine(await response.Content.ReadAsStringAsync());
```

Subaccount DKIM Authentication

Example Response

```
{
  "data": {
    "DnsHostName": "dkim1._domainkey.example.com",
    "Domain": "example.com",
    "Selector": "dkim1",
    "DnsRecord": "key",
    "PublicKey": "public key",
    "PrivateKey": "private key"
  }
}
```

Subaccount DKIM Authentication

Providing SocketLabs directly with a DKIM key pair

If the privateKey is not provided, we will generate a new public/private key pair for you. Otherwise, we will attempt to validate your private key against the public key for your domain.

Attributes

- **domain** (string)
  Domain or subdomain you want to set up for your DKIM entry
- **privateKey** (string)
- **selector** (string)

Subaccount DKIM Authentication

Example Request

```
var client = new HttpClient();

var request = new HttpRequestMessage(HttpMethod.Post,
    “https://api.socketlabs.com/v2/subaccount/:subaccountID/dkim?do-
main=:domain&selector=:selector”);

request.Headers.Add(“Authorization”, “Bearer <token>”);

var content = new StringContent(@”{
   “domain”: “example.com”,
   “selector”: ”dkim”,
   ”privateKey”: ”abcdefghijklmnop1234567890”
}”, null, “text/plain”);

request.Content = content;
var response = await client.SendAsync(request);
response.EnsureSuccessStatusCode();

Console.WriteLine(await response.Content.ReadAsStringAsync());
```

Subaccount DKIM Authentication

Example Response

```
{
  "data": {
    "Domain": "example.com",
    "Selector": "dkim",
    "truncatedPrivateKey": "truncated private key...",
    "recordType": "TXT",
    "createdOn": "1901-01-01T01:01:01.000Z",
    "updatedOn": "1901-01-01T01:01:01.000Z"
  }
}
```

With the basics in place, you'll move on to connecting your account to engagement tracking. This reporting will give you insight into how recipients are interacting with your mail.

Subaccount Engagement Tracking

The Engagement Tracking feature allows you to detect how your email recipients are interacting with messages sent to them; specifically, if they are opening the message, clicking on links within the message, or if the recipient has requested to be unsubscribed from the messages they are receiving.

To use the Engagement Tracking feature, you must configure one or more Tracking Domains. Tracking Domains are used to label the links and images in your email messages, allowing the domains in the URL to match your own domain while still allowing SocketLabs to track engagement.

Setup

Prior to setting up a Tracking Domain, a CNAME record must be established with your DNS service provider. This CNAME must point a subdomain to tracking.socketlabs.com. NOTE: You can re-use the CNAME record created for engagement tracking to set up your engagement tracking domain.

Endpoint

```
https://api.socketlabs.com/subaccount/:subaccountId/dkim
```

Subaccount Engagement Tracking

Path Variables

- subaccountId (number, required)

Attributes

Property Name	Data Type	Details
domain	string	Domain or subdomain you want to set up for engagement tracking
opensEnabled	boolean	
clicksEnabled	boolean	
unsubscribesEnabled	boolean	
automaticTrackingEnabled	boolean	
googleAnalyticsEnabled	boolean	
isDefault	boolean	

Subaccount Engagement Tracking

Example Request

```
var client = new HttpClient();

var request = new HttpRequestMessage(HttpMethod.Post,
    "https://api.socketlabs.com/v2/subaccount/:subaccountID/tracking"");

request.Headers.Add("Authorization", "Bearer <token>");

var content = new StringContent(@"{
   domain": "example.com",
   "opensEnabled": "false",
   "clicksEnabled": "false",
   "unsubscribesEnabled": "false",
   "automaticTrackingEnabled": "true",
   "googleAnalyticsEnabled": "true",
   "isDefault": "false",
}", null, "text/plain");

request.Content = content;
var response = await client.SendAsync(request);
response.EnsureSuccessStatusCode();

Console.WriteLine(await response.Content.ReadAsStringAsync());
```

Subaccount Engagement Tracking

Example Response

```
{
  “data”: {
    “Domain”: “tracking.example.com”
    “opensEnabled”: “true”,
    “clicksEnabled”: “true”,
    “automaticTrackingEnabled”: “true”,
    “gogleAnalyticsEnabled”: “true”,
    “encryptedTrackingStatus”: “Active”,
    “createdOn”: “1901-01-01T01:01:01.000Z”,
    “updatedOn”: “1901-01-01T01:01:01.000Z”
  }
}
```

Ready to send mail? You have some options. First, let’s move on to our Injection API.

Injection API

The injection API lets you send email. Each endpoint can send an email to a single recipient or thousands. SocketLabs generates and sends the messages using the options you've defined in the emailMessage object.

Create an Injection API Key

API calls to the Injection API are authenticated using the API key that you generated. Authenticate your calls to the Injection API using the Authorization header with the Bearer authentication scheme.

Endpoint

```
 https://api.socketlabs.com/v2/subaccount/:subaccountId/credentials/in-
jection-api
```

Path Variables

- subaccountId (number, required)

Example Request

Injection API

Example Request

```
var client = new HttpClient();

var request = new HttpRequestMessage(HttpMethod.Post,
    “https://api.socketlabs.com/v2/subaccount/credentials/injection-api
request.Headers.Add(“Authorization”, “Bearer <token>”);

var response = await client.SendAsync(request);

response.EnsureSuccessStatusCode();
Console.WriteLine(await response.Content.ReadAsStringAsync());
```

Example Response

```
{
  “data”: {
    “serverId”: 12345,
   “apiKey”: “xyzabc1233456”,
    “gateway”: “https://inject-cx.socketlabs.com/api/v2/email”,
  }
}
```

Injection API

Available Code Libraries

To make it easy to get started with the Injection API, we provide client libraries in several different programming languages. These libraries provide you with a quick start to start using our injection API in your chosen language.

We also maintain GitHub repositories for each of the client libraries. These repositories contain the full source code for the library, as well as detailed examples for many different use cases. We encourage you to take a look at these examples when getting started.

Injection API

If you do not want to use one of our prebuilt libraries, the easiest process for sending mail with the injection API is to:

1. Create a new Injection API Key
2. Create Email Message Object
3. Build and Add the Recipient Array to the Message Object
4. (Optional) Build and Add Custom Headers
5. (Optional) Build and Add Merge Data
6. Send with the Injection API

Create Email Message Object

The email message object includes options for different email body types, CC and BCC, merge data, custom headers, and more to give you the most flexibility around your email sending needs.

For example code, please refer to our library code examples.

Email Message Required Attributes

Property Name	Data Type	Details
To	(array[Recipient])	An array of recipient EmailAddress/ FriendlyName value pairs representing the recipients of an email message.
From	Recipient	The EmailAddress/FriendlyName value pair for the sender of the message.
Subject	String	The subject line of the email message. 200 character limit.

Injection API

Email Message Optional Attributes

Property Name	Data Type	Details
ReplyTo	Recipient	The EmailAddress/FriendlyName value pair for the sender of the message
TextBody	string	Body of text that would be the content of the mesage--this or HtmlBody are required
Htmlbody	string	Body of text that would be the content of the message- this or TextBody are required
AmpBody	string	body portion
MergeData	string	Data storage for the inline merge feature
MessageId	string	SocketLabs header used to tag individual messages
MailingId	string	SocketLabs header used to track batches of messages
Charset	string	The charset name to be used when creating the message. Default is UTF8.
CustomHeaders	array[CustomHeader]	An array of header field data stored in Name/ Value pairs.
CC	array[Recipient]	An array of recipient EmailAddress/ FriendlyName value pairs representing the CC'd recipients of an email message.

Injection API

Email Message Optional Attributes (cont.)

Property Name	Data Type	Details
BCC	array[Recipient]	An array of recipient EmailAddress/ FriendlyName value pairs representing the BCC'd recipients of an email message.
Attachments	array[Attachment]	An array of attached content blobs, such as images, documents, and other binary files. This is not recommended for bulk sending, as many receivers will either not accept the mail or will mark it as spam.

Injection API

If you do not want to use one of our prebuilt libraries, the easiest process for sending mail with the injection API is to:

1. Create a new Injection API Key
2. Create Email Message Object
3. Build and Add the Recipient Array to the Message Object
4. (Optional) Build and Add Custom Headers
5. (Optional) Build and Add Merge Data
6. Send with the Injection API

Create Email Message Object

The email message object includes options for different email body types, CC and BCC, merge data, custom headers, and more to give you the most flexibility around your email sending needs.

For example code, please refer to our library code examples.

Email Message Required Attributes

Property Name	Data Type	Details
To	(array[Recipient])	An array of recipient EmailAddress/ FriendlyName value pairs representing the recipients of an email message.
From	Recipient	The EmailAddress/FriendlyName value pair for the sender of the message.
Subject	String	The subject line of the email message. 200 character limit.

Injection API

Recipient Array Attributes

Property Name	Data Type	Details
emailAddress	string	An email address string such as foo@bar.com. This is required.
friendlyName	string	An alias for an email address. This is optional.

(Optional) Build and Add Custom Headers

Property Name	Data Type	Details
Name	string	The name of the header field to be added, such as “Content-Type”
Value	string	The value of the header field to be added, such as ‘application/json’

Injection API

(Optional) Build and Add Merge Data

Property Name	Data Type	Details
PerMessage	array[array[MergeValueData]]	A two dimensional (2D) array of Field/Value pairs, used to define merge field data for each message. Variables can be freely named, with the exception of a single reserved word, ‘DeliveryAddress‘, which defines the recipient of the current message.
Global	array[MergeValueData]	A array of Field:Value pairs that will be applied globally to all recipients.

MergeValueData

Property Name	Data Type	Details
Field	string	Optional
Value	string	Optional

Injection API

Send with the Injection API Example Request

```
var msg = new
{
    ServerId= "YOUR-SERVER-ID",
    APIKey = "YOUR-API-KEY",
    Messages = new object[]
    {
        new
        {
            To = new object[]
            {
                new { emailAddress = "recipient1@example.com" }
            }
            From = new object[]
            {
                new { emailAddress = "from@example.com" }
            }
            Subject = "Sending a Basic Message",
            TextBody = "This is the Plain Text Body of my message.",
            HtmlBody = "<html>This is the Html Body.</html>"
        }
    }
};
```

Injection API

Send with the Injection API Example Request (cont.)

```
var content = new StringContent(JsonConvert.SerializeObject(msg),
Encoding.UTF8, “application/json”);

var client = new HttpClient();

var request = new HttpRequestMessage(HttpMethod.Post,
    “https://inject.socketlabs.com/api/v1/email”);

request.Headers.Add(“Authorization”, “Bearer <API Key>”);

request.Content = content;
var response = await client.SendAsync(request);
response.EnsureSuccessStatusCode();

Console.WriteLine(await response.Content.ReadAsStringAsync());
```

Injection API

Example Success Response

```
{
  "ErrorCode": "Success",
  "MessageResults": "null",
  "TransactionReceipt": "null"
}
```

Example Failure Response

```
{
  "data": {
    "error": [
      {
        "errorType": "string",
        "message": "string"
      }
    ]
  }
```

Failure Codes

Please see [the API documentation] for more details regarding failure codes.

Want to set up SMTP? Our next section will assist.

SMTP

The SocketLabs API can be used for getting and updating SMTP credentials for a subaccount.

Endpoint

```
 https://api.socketlabs.com/v2/subaccount/:subaccountId/credentials/
smtp
```

Path Variables

- subaccountId (number, required)

Get SMTP Credentials for a Subaccount

```
var client = new HttpClient();

var request = new HttpRequestMessage(HttpMethod.Post,
    "https://api.socketlabs.com/v2/servers/12345/credentials/smtp");

request.Headers.Add("Authorization", "Bearer <token>");

var response = await client.SendAsync(request);

response.EnsureSuccessStatusCode();
Console.WriteLine(await response.Content.ReadAsStringAsync());
```

SMTP

Example Response

```
{
  “data”: {
    “username”: “subaccount12345”,
    “password”: “xyzabc1233456”,
    “gateway”: “smtp.socketlabs.com”
  }
}
```

SMTP

Update SMTP Password for the Subaccount

Example Request

```
var client = new HttpClient();

var request = new HttpRequestMessage(HttpMethod.Put,
    "https://api.socketlabs.com/v2/servers/12345/credentials/smtp");

request.Headers.Add("Authorization", "Bearer <token>");

var content = new StringContent(@"{
    'password'='abcdefghijklmnop1234'
}", null, "text/plain");
request.Content = content;

var response = await client.SendAsync(request);
response.EnsureSuccessStatusCode();

response.EnsureSuccessStatusCode();
Console.WriteLine(await response.Content.ReadAsStringAsync());
```

SMTP

Update SMTP Password for the Subaccount

Example Response

```
{
  "data": {
    "username": "subaccount12345",
    "password": "xyzabc1233456",
    "gateway": "smtp.socketlabs.com"
  }
}
```

Account Level Event Webhook

The Event Webhook provides the ability to subscribe to automatically generated HTTP POST event notifications, which can be consumed by your own applications. Notifications are generated for SMTP events and recipient engagement events. This webhook makes it easy to receive real-time notification of these events and send the data back to your own platform.

Events

Property Name	Details
Queued	The mail has been injected and has been accepted for delivery.
Sent	SocketLabs sent the email and it was accepted by the recipient email server.
Failed	SocketLabs could not deliver the email to the recipient email server.
Deferred	The recipient’s mail server has temporarily refused delivery of a message
Complaint	The email recipient clicked to report “this is spam” within their email client.
Click	The email recipient clicked on a link in the email. Click tracking must be enabled and the CNAME record for the engagement tracking domain must point to tracking.socketlabs.com.
Open	The email recipient opened the email.

Account Level Event Webhook

A note on Asynchronous Bounces:

The way an asynchronous bounce works is that a receiving server will accept an email and will send a bounce later, sometimes days later. Because of this, in the case of an asynchronous bounce, the webhooks will show a delivered event and will then show a failure when the bounce is received.

Authentication and Access

1. Write and deploy a secure (HTTPS) endpoint that can process and respond to notifications. The URL of this endpoint is required to validate and enable this feature.
2. Either:
    1. Log into the SocketLabs platform and choose “Event Webhook” under “Configuration”, ensuring you are in the account overview and are **NOT in a subaccount**. Add a webhook to get the secret key. Your endpoint must respond with this secret key to validate.
    2. Generate the secret key programmatically:

```
var client = new HttpClient();

var request = new HttpRequestMessage(HttpMethod.Post,
    ““https://api.socketlabs.com/v2/event-webhook/generate-secret-key”);
request.Headers.Add(“Authorization”, “Bearer <token>”);

var response = await client.SendAsync(request);

response.EnsureSuccessStatusCode();
Console.WriteLine(await response.Content.ReadAsStringAsync());
```

Account Level Event Webhook

Error Handling and Rate Limiting

Once your endpoint is configured properly, there is no hard limit to the number of notifications you can receive. If an endpoint is temporarily unavailable due to a system outage on our end, your end, or somewhere between — all notifications will be queued up for another delivery attempt in exactly the same way as email deferral works. Notification API messages will be not be permanently failed and lost unless no contact can be made for at least four days.

In the case where your endpoint becomes unresponsive for long periods of time, we may disable the notification feature on your account to prevent generating large numbers of notifications which cannot be delivered.

Endpoint Implementation

To use webhooks, users must write a handler that can process and respond to notification messages via HTTP POST. This handler must be located at a public-facing URL and must be validated by the SocketLabs platform before the URL can be activated.

To validate, the handler must support the following functionality:

- Accepting HTTP POST messages
- Recognize the Secret Key field to confirm that SocketLabs is the sender of the HTTP POST message
- Respond to official notification messages with `200 OK` HTTP Response.

Account Level Event Webhook

Endpoint Implementation

HTTPS communication is required on your endpoint. Using unencrypted HTTP can pose a number of security risks. By the nature of the data being transmitted, Event Webhook-related calls made to your endpoint contain your SocketLabs subaccount ID and Secret Key, as well as PII (Personally Identifiable Information) belonging to end-users, such as email address and IP address. Insecure transfer of such data over HTTP may pose data security risks.

Implementation Notes

If the Secret Key does not match in a notification message, your application should return a 401 error response.

All new endpoints will be in the JSON (`application/json`) data format.

Endpoint Validation

Once a handler application is installed and configured at a URL, this URL should be entered into the 'Endpoint URL' field in the SocketLabs platform before using the Validate button.

Validation Failures

Likely reasons for this validation to fail include:

- The endpoint is unreachable by SocketLabs, perhaps due to being behind a firewall.
- The handler is not accepting HTTP POST messages.
- The handler is not responding to notification messages with a `200` HTTP response.

Account Level Event Webhook

Validation Failures

We recommend checking network security settings as well as examining the POST and POST Response data in the handler if the Endpoint URL validation initially fails.

When testing your handler, you can use the configuration page in the SocketLabs platform to send test notifications.

If your endpoint resides behind a firewall, you may need to allow SocketLabs' infrastructure to access your network to receive notifications at your endpoint. Event Webhooks can send notifications from multiple pieces of our infrastructure. It is therefore recommended that you allow our entire outbound IP range, 142.0.176.0/20.

Endpoint Testing

Once you have deployed and validated your endpoint, you can use the "Event Tester" feature in the SocketLabs platform to send sample notifications to your endpoint for testing. Select the type of sample notification to generate from the dropdown, then click the "Test" button. This will generate and send a notification to your endpoint. We will also generate a preview of what the body of the notification will look like in your preferred data format.

Once you send a test notification, an indicator will pop up showing the HTTP response code that we received from your endpoint as well as whether the notification was accepted.

Account Level Event Webhook

Endpoint

```
https://api.socketlabs.com/v2/event-webhook/:webhookID
```

Example Request: Sent Messages

```
var client = new HttpClient();

var request = new HttpRequestMessage(HttpMethod.Get,
    "https://api.socketlabs.com/v2/event-webhook/:webhookId/per-
form-test?type=Sent");
request.Headers.Add("Authorization", "Bearer <token>");

var response = await client.SendAsync(request);
response.EnsureSuccessStatusCode();

response.EnsureSuccessStatusCode();
Console.WriteLine(await response.Content.ReadAsStringAsync());
```

Account Level Event Webhook

Example Response: Sent Messages

```
"data": {
    "Type": "Delivered",
    "Response": "Sample Response",
    "LocalIp": ":01",
    "RemoteMta": "Sample RemoteMTA",
    "DateTime": "2022-09-07T17:49:08.8139901Z",
    "MailingId": "SLNL-0-9999999-9999999",
    "MessageId": "SampleMessageId",
    "Address": "email@example.com",
    "ServerId": :subaccountID,
    "SecretKey": "ASecretKeyGoesHere!",
         "Data": {
              "Meta": {
                   "Key": "x-mycustommetadata",
                   "Value": "I am custom metadata"
              },
              "Tags": [
                   "Sample Tag",
                   "Sample Message"
              ]
         }
  }
}
```

For more code examples and to dig into our subaccount level event webhook, see our Event Webhook documentation.

Thank you and Resources

Thanks for taking the time to read through our Complex Sender Quickstart Guide! This should get you up and ready to go with ease.

For more assistance, please see:
- Our Developer hub
- API Documentation