Mail Sender Documentation

Release 0.1.0

Anthony Ruhier

Aug 16, 2017

Contents

CHAPTER 1

Quickstart

Mail Sender is a Python application allowing to send emails through 2 providers, AmazonSES and Mailgun, with an automatic failover. It provides a REST API, documented with Swagger, and a python client to use it.

An demo is hosted on https://mail-sender.uber.aruhier.fr.

A client can be found here.

Table of Contents

Installation

First clone the project:

```
$ git clone https://github.com/Anthony25/mail-sender-daemon.git
$ cd mail-sender-daemon
```

Install it via pip3 (requires `python3-pip` or `python-pip`, depending whether python 2 or 3 is the default):

```
$ pip3 install -e
```

Then use a WSGI container, like Gunicorn, by refering to the Flask documentation. The application can be imported with `mail_sender_daemon:app`.

Configuration

A configuration file is needed for the daemon to run. Copy and tweak the self documented `config.yml.default` file (available at the repository root) in one of the following paths:

- `~/.config/mail-sender-daemon/config.yml`: user separated configuration
- `/etc/mail-sender-daemon/config.yml`: systemd-wide configuration

CHAPTER 2

Design

This part will present the different design choices made for this project.

Table of Contents

Application

A separation between a frontend and a backend application has been adopted: the last one provide a REST API, used by the first one to interact with it.

Python has been chosen for its simplicity. Compute time is not important here, as the application relies mainly on its providers. The REST API is done by flask-restplus, using Swagger to provide data verification and documentation. `requests` is used to implement a client for the 2 providers' web API.

Infrastructure

Different redundancy techniques have been used to host the daemon:

Fig. 2.1: Mail Sender infrastructure

The infrastructure is split as 2 main parts: my home infrastructure, and 2 vm hosted on Digital Ocean. Round Robin DNS is used between each entry point.

On Digital Ocean, each host runs a HAProxy and Mail Sender Daemon (in a docker). `keepalived` is used to setup VRRP, detecting if HAProxy is still up, otherwise the other host will take up the relay. Digital Ocean only provides IPv4 floating addresses, that is why the IPv6 is not included into VRRP.

On my home infrastructure, a public IP is dynamically routed through OSPF from a Online server. HAProxy is used, and load balances to 3 nodes, running on LXC. VRRP cannot be used here, as Online does not provide a floating API, and I do not have the control on the IP external announcement.

Tradeoffs

Synchronous vs asynchronous

When sending an email, a synchronous design has been chosen: the web API will try to send the email when requested, and the client will wait for an answer indicating if it failed or not.

However, an asynchronous design has been thought: a client could send a request to the API, which then transmits it to a message broker, and returns a token to the client as a response. The message broker then load balances the mail sending to different nodes, which store the sending status in a database. The client could check the sending status by calling a method in the API with the same token they previously received.

This design has the advantage to handle much more capacity, as all messages could be stacked into the message broker queue, and be sent when a node is free. Clients will not timeout if their mail take longer than usual to be sent, and a retry could be implemented if no provider is available.

However, it is way more complex: users should be able to request the sending status, a database and a message broker have to be added. On the infrastructure side, it also means that, in order to avoid any SPOF, the database and the message broker should both be in a cluster. Also, as the current infrastructure is divided into 2 sites (home infrastructure and Digital Ocean), a split brain could occur.

Regarding the delay of this project, an asynchronous design would have brought too much complexity to be implemented. For this reason, the synchronous design has followed.

CHAPTER 3

API

A Swagger documentation is already available here, so this part only contains quick description of each method.

Table of Contents

Send: `/send`

Allows to send an email. Depending on the provider, the destination address has to be validated, or the email will be unauthorized.

Validation: `/validation`

Some providers does not allow email sending to addresses that have not been validated before (AmazonSES for example). To check the validation status of an address, use `GET /validation/{address}`. To validate an address, use `POST /validation/{address}`. An email will be send to the specified address, asking if wanted to be white-listed.

CHAPTER 4

Indices and tables

- genindex
- search