Communications Platform Team Coding Challenge

Multiple Email Providers

Overview

In order to prevent downtime during an email service provider outage, you're tasked with creating a service that provides an abstraction between two different email service providers. This way, if one of the services goes down, you can quickly failover to a different provider without affecting your customers.

Specifications:

Please create an HTTP service that accepts POST requests with JSON data to a '/email' endpoint with the following parameters:

- 'to' - The email address to send to
- 'to_name' - The name to accompany the email
- 'from' - The email address in the from and reply fields
- 'from_name' - the name to accompany the from/reply emails
- 'subject' - The subject line of the email
- 'body' - the HTML body of the email

Example Request Payload:

```
{
   “to”: “fake@example.com”,
   “to_name”: “Ms. Fake”,
   “from”: “noreply@uber.com”,
   “from_name”: “Uber”,
   “subject”: “A Message from Uber”,
   “body”: “<h1>Your Bill</h1><p>$10</p>”
}
```

Your service should then do a bit of data processing on the request:

- Do the appropriate validations on the input fields (NOTE: all fields are required).
- Convert the 'body' HTML to a plain text version to send along to the email provider. You can simply remove the HTML tags. Or if you'd like, you can do something smarter.

Once the data has been processed and meets the validation requirements, it should send the email by making an HTTP request (don't use SMTP) to one of the following two services:

- *Mailgun*
  - Main Website: www.mailgun.com
  - Simple Send Documentation: http://documentation.mailgun.com/quickstart.html#sending-messages
- *Mandrill*

    - Main Website: www.mandrillapp.com
    - Simple Send Documentation: https://mandrillapp.com/api/docs/messages.JSON.html#method-send

Both services are free to try and are pretty painless to sign up for, so please register your own test accounts on each.

Your service should send emails using one of the two options by default, but a simple configuration change and re-deploy of the service should switch it over to the other provider.

Implementation Requirements:

- Please do not use the client libraries provided by Mandrill or Mailgun. In both cases, you're making simple post requests - do this with a lower level package or your language's built-in commands.
- Stay away from big frameworks like Rails or Django. Freel free to use a microframework like Flask, Sinatra, Express, Silex, etc.
- This is a simple exercise, but organize, design, document and *test* your code as if it were going into production.
- Most of Uber's back end is in Python or Node.JS. If you know one of these languages, please use it. If not, that's totally OK; write up your service in whatever language you're most comfortable in.
- When you're finished, post your code to a github or bitbucket page so we can check it out. Please don't commit your email provider API keys in the repository.
- Please include a README file in your repository with the following information:
    - How to install your application
    - Which language and/or microframework you chose and why
    - Trade-offs you might have made, anything you left out, or what you might do differently if you were to spend additional time on the project
    - Anything else you wish to include.

How We Review

Your application will be reviewed by at least three of our engineers. The aspects of your code we will judge include:

- Functionality: Does the application do what was asked? If there is anything missing, does the README explain why it is missing?
- Code quality: Is the code simple, easy to understand, and maintainable? Are there any code smells or other red flags?
- Testing: How thorough are the automated tests? Will they be difficult to change if the requirements of the application were to change?
- Technical choices: Do choices of libraries, architecture, etc. seem appropriate for the chosen application?

Bonus Points!

If you have some extra time, here's a few features you could add. They definitely aren't required, but might be fun to hack on!

- Instead of relying on a configuration change for choosing which email provider to use, dynamically select a provider based on their error responses. For instance, if Mailgun started to timeout or was returning errors, automatically switch to Mandrill.
- Keep a record of emails passing through your service in some queryable form of data storage.
- Both Mandrill and Mailgun have webhooks for email opens and clicks. Implement endpoints on your service to receive those webhook POST requests and store that information in some form of data storage.
- Both services offer delayed delivery. Implement a delivery date / time parameter for POST requests to your service.
- Or any other fun feature you want to add!

Thanks for working on this. We really appreciate you putting the time in to show us what you've got!