Envío de correos electrónicos con Python

Tabla de contenido

Probablemente queramos enviar correos electrónicos usando Python. Quizás deseemos recibir recordatorios por correo electrónico de su código, enviar un correo electrónico de confirmación a los usuarios cuando crean una cuenta o enviar correos electrónicos a los miembros de su organización para recordarles que paguen sus cuotas. Enviar correos electrónicos manualmente es una tarea que consume mucho tiempo y es propensa a errores, pero es fácil de automatizar con Python.

En este tutorial aprenderemos a:

- Configurar una **conexión segura** usando `SMTP_SSL()`y `.starttls()`

- Utilizar la biblioteca `smtplib` incorporada de Python para enviar **correos electrónicos básicos**
- Enviar correos electrónicos con **contenido HTML** y **archivos adjuntos** utilizando el paquete `email`
- Enviar varios **correos electrónicos personalizados** utilizando un archivo CSV con datos de contacto
- Usar el paquete **Yagmail** para enviar correo electrónico a través de su cuenta de Gmail usando solo unas pocas líneas de código

Encontraras algunos servicios de correo electrónico transaccional al final de este tutorial, que resultarán útiles cuando se quieran enviar una gran cantidad de correos electrónicos.

Empezando

Python viene con el módulo `smtplib` incorporado para enviar correos electrónicos usando el Protocolo simple de transferencia de correo (SMTP). `Smtplib` utiliza el protocolo RFC 821 para SMTP. Los ejemplos de este tutorial utilizarán el servidor SMTP de Gmail para enviar correos electrónicos, pero los mismos principios se aplican a otros servicios de correo electrónico. Aunque la mayoría de los proveedores de correo electrónico utilizan los mismos puertos de conexión que los de este tutorial, puede ejecutar una búsqueda rápida en Google para confirmar el vuestro.

Para comenzar con este tutorial, configure una cuenta de Gmail para el desarrollo o configure un servidor de depuración SMTP que descarte los correos electrónicos que envíe y los imprima en el símbolo del sistema. Ambas opciones se presentan a continuación. Un servidor de depuración SMTP local puede ser útil para solucionar cualquier problema con la funcionalidad de correo electrónico y garantizar que sus funciones de correo electrónico estén libres de errores antes de enviar cualquier correo electrónico.

Opción 1: configurar una cuenta de Gmail para el desarrollo

Si decidimos usar una cuenta de Gmail para enviar los correos electrónicos, se recomienda que configuremos una cuenta desechable para el desarrollo

de su código. Esto se debe a que tendrá que ajustar la configuración de seguridad de su cuenta de Gmail para permitir el acceso desde su código Python, y porque existe la posibilidad de que accidentalmente exponga sus datos de inicio de sesión. Además, descubrí que la bandeja de entrada de mi cuenta de prueba se llenó rápidamente de correos electrónicos de prueba, razón suficiente para configurar una nueva cuenta de Gmail para el desarrollo.

- Crea una nueva cuenta de Google .
- Tenemos que *Permitir aplicaciones menos seguras a EN* . Tenga en cuenta que esto facilita que otros accedan a su cuenta.

Si no desea reducir la configuración de seguridad de su cuenta de Gmail, consulte la documentación de Google sobre cómo obtener credenciales de acceso para su secuencia de comandos de Python, utilizando el marco de autorización OAuth2.

Opción 2: configuración de un servidor SMTP local

Podemos probar la funcionalidad del correo electrónico ejecutando un servidor de depuración SMTP local, utilizando el módulo `smtpd` que viene preinstalado con Python. En lugar de enviar correos electrónicos a la dirección especificada, los descarta e imprime su contenido en la consola. Ejecutar un servidor de depuración local significa que no es necesario lidiar con el cifrado de mensajes o usar credenciales para iniciar sesión en un servidor de correo electrónico.

Se puede iniciar un servidor de depuración SMTP local escribiendo lo siguiente en el símbolo del sistema:

```
$  ython -m smtpd -c DebuggingServer -n localhost:1025
```

En Linux, use el mismo comando precedido por `sudo`.

Todos los correos electrónicos enviados a través de este servidor se descartarán y se mostrarán en la ventana del terminal como un objeto de `bytes` para cada línea:

```
---------- MESSAGE FOLLOWS ----------
b'X-Peer: ::1'
b''
```

```
b'From: my@address.com'
b'To: your@address.com'
b'Subject: a local test mail'
b''
b'Hello there, here is a test email'
------------ END MESSAGE ------------
```

Para el resto del tutorial, asumiré que estamos usando una cuenta de Gmail, pero si estás usando un servidor de depuración local, solo asegúrate de usarlo `localhost` como tu servidor SMTP y usa el puerto 1025 en lugar del puerto 465 o 587. Además de esto, no necesitará usar `login()` o encriptar la comunicación usando SSL / TLS.

Envío de un correo electrónico de texto sin formato

Antes de sumergirnos en el envío de correos electrónicos con contenido HTML y archivos adjuntos, aprenderemos a enviar correos electrónicos de texto sin formato con Python. Estos son correos electrónicos que se pueden escribir en un editor de texto simple. No hay cosas sofisticadas como formato de texto o hipervínculos. Lo aprenderemos un poco más tarde.

Inicio de una conexión SMTP segura

Cuando se envía correos electrónicos a través de Python, debe de asegurarse que nuestra conexión SMTP esté encriptada, de modo que otros no puedan acceder fácilmente a su mensaje y a sus credenciales de inicio de sesión. SSL (Secure Sockets Layer) y TLS (Transport Layer Security) son dos protocolos que se pueden utilizar para cifrar una conexión SMTP. No es necesario utilizar ninguno de estos cuando se utiliza un servidor de depuración local.

Hay dos formas de iniciar una conexión segura con su servidor de correo electrónico:

- Iniciar una conexión SMTP que esté asegurada desde el principio utilizando `SMTP_SSL()`.
- Iniciar una conexión SMTP no segura que luego se puede cifrar con `.starttls()`.

En ambos casos, Gmail cifrará los correos electrónicos mediante TLS, ya que es el sucesor más seguro de SSL. Según las consideraciones de seguridad de Python , se recomienda encarecidamente que se utilice `create_default_context()` desde el módulo `ssl`. Esto cargará los certificados de CA de confianza del sistema, habilitará la verificación del nombre de host y la validación del certificado, e intentará elegir un protocolo y una configuración de cifrado razonablemente seguros.

Si deseamos verificar el cifrado de un correo electrónico en su bandeja de entrada de Gmail, vaya a *Más* → *Mostrar original* para ver el tipo de cifrado que se encuentra debajo del encabezado *Recibido*.

`smtplib` es el módulo integrado de Python para enviar correos electrónicos a cualquier máquina de Internet con un demonio de escucha SMTP o ESMTP.

Primero mostraremos cómo usar SMTP_SSL(), ya que instancia una conexión que es segura desde el principio y es un poco más concisa que la `.starttls()`alternativa. Hay que tener en cuenta que Gmail requiere que se conecte al puerto 465 si se usa `SMTP_SSL()` y al puerto 587 cuando lo usa `.starttls()`.

Opción 1: usar `SMTP_SSL()`

El siguiente ejemplo de código crea una conexión segura con el servidor SMTP de Gmail, utilizando el `SMTP_SSL()`de `smtplib` para iniciar una conexión encriptada TLS. El contexto predeterminado de `ssl` valida el nombre de host y sus certificados y optimiza la seguridad de la conexión. Se debe de ingresar vuestra propia dirección de correo electrónico en lugar de la de mi script (sendMail.py).

sendMail.py:

```
import smtplib, ssl
import getpass

#Puerto para conectar mediante SSL
port = 465

#Version con input
##password = input('Password:')

#El getpass() es como un input pero no se muestra lo que se
escribe.
#NOOOO funciona desde el IDLE.
```

```
#O bien gastamos el CMD de Windows o
#usamos el Visual Studio.
password = getpass.getpass('Password:')

#Estos valores cambiarlos por vuestro correo de pruebas
sender_mail = "pruebasdaw2021@gmail.com"
receiver_mail = "pruebasdaw2021@gmail.com"
server_domain = "smtp.gmail.com"

#Poned al menos dos lineas entre el Subject y el cuerpo del
mensaje#
#para que lo pueda interpretar bien el protocolo SMTP

msg = '''Subject: Hola mundo



Este es un mensaje de prueba'''

#Creamos un conexion SSL

context = ssl.create_default_context()

#Conexion SMTP al dominio smtp.gmail.com

with smtplib.SMTP_SSL(server_domain
                      , port, context=context) as s:

    #Hacemos login con nuestro usuario
    s.login(sender_mail, password)
    s.sendmail(sender_mail, receiver_mail, msg)

print("Email enviado!!!")
```

El uso de `with smtplib.SMTP_SSL() as server:` asegura que la conexión se cierre automáticamente al final del bloque de código sangrado. Si `port` es cero, o no se especifica, `.SMTP_SSL()`usará el puerto estándar para SMTP sobre SSL (puerto 465).

No es una práctica segura almacenar la contraseña de correo electrónico en nuestro código, especialmente si tenemos la intención de compartirlo con otras personas. En su lugar, utilizamos `input()` para permitir que el usuario escriba su contraseña cuando ejecute el script. Si no deseamos que nuestra contraseña se muestre en pantalla cuando la escribamos, se puede importar el módulo `getpass` y usar `.getpass()`para ingresar la contraseña a ciegas.

Opción 2: Usar `.starttls()`

En lugar de usar `.SMTP_SSL()`para crear una conexión que sea segura desde el principio, podemos crear una conexión SMTP no segura y encriptarla usando `.starttls()`.

Para hacer esto, creamos una instancia de `smtplib.SMTP`, que encapsula una conexión SMTP y nos permite acceder a sus métodos. Recomiendo definir el servidor y puerto SMTP al comienzo de la secuencia de comandos para configurarlos fácilmente.

El fragmento de código a continuación usa la construcción `server = SMTP()`, en lugar del formato `with SMTP() as server:` que usamos en el ejemplo anterior. Para asegurarse de que el código no se bloquee cuando algo sale mal, colocamos el código principal en un bloque `try` y dejamos que un bloque `except` imprima cualquier mensaje de error en `stdout`.

sendMail_TLS.py:

```
import smtplib, ssl
import getpass

smtp_server = "smtp.gmail.com"
port = 587  # Puerto para starttls

#Estos valores cambiarlos por vuestro correo de pruebas
sender_mail = "pruebasdaw2021@gmail.com"
receiver_mail = "pruebasdaw2021@gmail.com"



msg = '''Subject: Hola mundo



Este es un mensaje de prueba desde Python.'''


#password = input("Type your password and press enter: ")
password = getpass.getpass('Password:')

# Creamos un contexto para una conexión segura de SSL
context = ssl.create_default_context()

# Intentamos logearnos en el servidor y enviar un mail
try:
    server = smtplib.SMTP(smtp_server,port)
    server.ehlo() # Se puede omitir
    server.starttls(context=context) # Securizamos la conexión
    server.ehlo() # Can be omitted
```

```
    server.login(sender_email, password)
    # TODO: Send email
    server.sendmail(sender_mail, receiver_mail, msg)
except Exception as e:
    # Imprimimos los errores en stdout
    print(e)
finally:
    #Pase lo que pase, cerramos la conexión.
    server.quit()
```

Para identificarse en el servidor, se debe llamar a `.helo()`(SMTP) o `.ehlo()`(ESMTP) después de crear un objeto `.SMTP()`, y nuevamente después `.starttls()`. Esta función es llamada implícitamente por `.starttls()` y `.sendmail()` si es necesario, por lo que a menos que desee verificar las extensiones del servicio SMTP del servidor, no es necesario usar `.helo( )`o `.ehlo()` explícitamente.

Envío de su correo electrónico de texto sin formato

Después de iniciar una conexión SMTP segura utilizando cualquiera de los métodos anteriores, se puede enviar nuestro correo electrónico utilizando `.sendmail()`:

```
server.sendmail(sender_email, receiver_email, msg)
```

Recomiendo definir las direcciones de correo electrónico y el contenido del mensaje en la parte superior de su secuencia de comandos, después de las importaciones, para que pueda cambiarlos fácilmente:

```
smtp_server = "smtp.gmail.com"
port = 587  # Puerto para starttls

#Estos valores cambiarlos por vuestro correo de pruebas
sender_mail = "pruebasdaw2021@gmail.com"
receiver_mail = "pruebasdaw2021@gmail.com"
msg = '''Subject: Hola mundo



Este es un mensaje de prueba desde Python.'''
```

La cadena `msg` comienza con `"Subject: Hola mundo"` seguida de dos nuevas líneas (`\n`). Esto asegura que `Hola mundo` aparezca como el asunto del

correo electrónico, y el texto que sigue a las nuevas líneas se tratará como el cuerpo del mensaje.

Ya hemos visto como enviar mensajes de correo con texto sin formato. Veamos como darles formato a nuestros mensajes.

Envío de correos electrónicos elegantes

El paquete `email` integrado de Python nos permite estructurar correos electrónicos más sofisticados, que luego se pueden transferir como ya lo hemos hecho con el módulo `smtplib`. A continuación, aprenderemos cómo usar el paquete `email` para enviar correos electrónicos con contenido HTML y archivos adjuntos.

Incluyendo contenido HTML

Si deseamos formatear el texto en nuestro correo electrónico (**negrita** , *cursiva* , etc.), o si deseamos agregar imágenes, hipervínculos o contenido receptivo, HTML es muy útil. El tipo de correo electrónico más común en la actualidad es el correo electrónico multiparte MIME (Extensiones multipropósito de correo de Internet), que combina HTML y texto sin formato. Los mensajes MIME son manejados por el módulo `email.mime` de Python . Para obtener una descripción detallada, consulte la documentación.

Como no todos los clientes de correo electrónico muestran contenido HTML de forma predeterminada, y algunas personas eligen recibir solo mensajes de texto sin formato por razones de seguridad, es importante incluir una alternativa de texto sin formato para los mensajes HTML. Como el cliente de correo electrónico representará primero el último archivo adjunto de varias partes, debemos asegurarnos de agregar el mensaje HTML después de la versión de texto sin formato.

En el siguiente ejemplo, nuestros objetos `MIMEText()` contendrán las versiones HTML y de texto sin formato de nuestro mensaje, y la instancia `MIMEMultipart("alternative")` las combina en un solo mensaje con dos opciones de representación alternativas.

sendMail_HTML.py:

```
import smtplib, ssl
import getpass
from email.mime.text import MIMEText
```

```
from email.mime.multipart import MIMEMultipart

port = 465


sender_mail = "pruebasdaw2021@gmail.com"
receiver_mail = "pruebasdaw2021@gmail.com"
server_domain = "smtp.gmail.com"

password = getpass.getpass('Password:')

message = MIMEMultipart("alternative")

message["Subject"] = "multipart test"
message["From"] = sender_mail
message["To"] = receiver_mail

#Creamos el texto plano y el HTML para nuestro mensaje

text = '''\
Hola
Como estas??
Espero que todo bien.
Luis
www.google.com
'''

html = '''\
<html>
 <body>
 <p>Hola, <br>como estas??<br><b>espero que bien</b>
 <a href="http://www.google.com">Enlace</a>
 </p>
 </body>
 </html>
'''

#Convertimos estos plain/html en objetos MIMEText
parte1 = MIMEText(text, "plain")
parte2 = MIMEText(html, "html")


#Añadimos estos objetos MIMEText de plain y HTML a nuestro
#mensaje MIMEMultipart

message.attach(parte1)
message.attach(parte2)

#Creamos un conexion segura de SSL

context = ssl.create_default_context()

#Conexion SMTP al dominio smtp.gmail.com

with smtplib.SMTP_SSL(server_domain
                      , port, context=context) as s:
```

```
    #Hacemos login con nuestro usuario
    s.login(sender_mail, password)
    s.sendmail(sender_mail, receiver_mail, message.as_string())

print("Email enviado!!!")
```

En este ejemplo, primero se define el mensaje HTML y de texto sin formato como cadenas literales y luego los almacenamos como objetos `MIMEText plain/ html`. Luego, estos se pueden agregar en este orden al mensaje `MIMEMultipart("alternative")` y enviarse a través de nuestra conexión segura con el servidor de correo electrónico. Recordemos agregar el mensaje HTML después de la alternativa de texto sin formato, ya que los clientes de correo electrónico intentarán representar primero la última subparte.

Adición de archivos adjuntos mediante el paquete email

Para enviar archivos binarios a un servidor de correo electrónico que está diseñado para trabajar con datos textuales, deben codificarse antes del transporte. Esto se hace más comúnmente usando base64, que codifica datos binarios en caracteres ASCII imprimibles.

El siguiente ejemplo de código muestra cómo enviar un correo electrónico con un archivo PDF como archivo adjunto.

sendMailAdjunto.py:

```
import smtplib, ssl
import getpass
from email.mime.text import MIMEText
from email.mime.multipart import MIMEMultipart
from email.mime.base import MIMEBase
from email import encoders


port = 465


sender_mail = "pruebasdaw2021@gmail.com"
receiver_mail = "pruebasdaw2021@gmail.com"
server_domain = "smtp.gmail.com"

password = getpass.getpass('Password:')

subject = "Mail con adjunto"
body = "Esto es un mensaje con archivo adjunto Python."
```

```
#Creamos el objeto MIMEMultipart y añadimos las cabeceras
message = MIMEMultipart()

message["Subject"] = subject
message["From"] = sender_mail
message["To"] = receiver_mail
message["Bcc"] = receiver_mail

#Añadimos el body
message.attach(MIMEText(body, "plain"))

filename = "documento.pdf"


#Abrimos el archivo PDF en modo binario
with open(filename, 'rb') as f:

    part = MIMEBase('application', 'octet-stream')
    part.set_payload(f.read())


#Codificamos archivo en caracteres ASCII para enviarlos vía Mail
encoders.encode_base64(part)

part.add_header("Content-Disposition",
                "atachment;filename={}".format(filename))

#Añadimos la parte al mensaje y lo convertimos a string
message.attach(part)
text = message.as_string()

#Creamos un conexion SSL
context = ssl.create_default_context()

#Conexion SMTP al dominio smtp.gmail.com
with smtplib.SMTP_SSL(server_domain
                      , port, context=context) as s:

    #Hacemos login con nuestro usuario
    s.login(sender_mail, password)
    s.sendmail(sender_mail, receiver_mail, text)

print("Email enviado!!!")
```

El mensaje `MIMEultipart()` acepta parámetros en forma de pares clave / valor estilo RFC5233 , que se almacenan en un diccionario y se pasan al método `.add_header` de la clase base `Message`.

Consulte la documentación del módulo `email.mime` de Python para obtener más información sobre el uso de clases MIME.

Envío de varios correos electrónicos personalizados

Imaginemos que se desea enviar correos electrónicos a los miembros de nuestra organización para recordarles que paguen sus cuotas de contribución. O tal vez deseamos enviar a los estudiantes de clase correos electrónicos personalizados con las calificaciones de su tarea reciente. Estas tareas son muy sencillas en Python.

Crear un archivo CSV con información personal relevante

Un punto de partida sencillo para enviar varios correos electrónicos personalizados es crear un archivo CSV (valores separados por comas) que contenga toda la información personal necesaria. (Hay que asegurarse de no compartir información privada de otras personas sin su consentimiento). Un archivo CSV puede considerarse como una tabla simple, donde la primera línea a menudo contiene los encabezados de las columnas.

A continuación, se muestra el contenido del archivo `contacts_file.csv`, que guardé en la misma carpeta que mi código Python. Contiene los nombres, direcciones y calificaciones de un conjunto de personas ficticias. Se usan construcciones `mydirection+modifier@gmail.com` para asegurarnos de que todos los correos electrónicos terminan en mi propia bandeja de entrada, que en este ejemplo es pruebasDAW2021@gmail.com :

```
name,email,grade
Ron Obvious,pruebasDAW2021+ovious@gmail.com,B+
Killer Rabbit of Caerbannog,pruebasDAW2021+rabbit@gmail.com,A
Brian Cohen,pruebasDAW2021+brian@gmail.com,C
```

Al crear un archivo CSV nos aseguramos de separar sus valores con una coma sin espacios en blanco alrededor.

Recorramos filas para enviar varios correos electrónicos

El siguiente ejemplo de código muestra cómo abrir un archivo CSV y recorrer sus líneas de contenido (omitiendo la fila del encabezado). Para asegurarnos de que el código funciona correctamente antes de enviar correos electrónicos a todos sus contactos, imprimí `Sending email to ...` para cada contacto, que luego podemos reemplazar con una funcionalidad que realmente envía correos electrónicos:

```
import csv

with open("contacts_file.csv") as file:
    reader = csv.reader(file)
    next(reader)  # Omitimos la primera fila
    for name, email, grade in reader:
        print("Sending email to {}".format(name))
        # Send email here
```

En el ejemplo anterior, el uso de `with open(filename) as file:` asegura que el archivo se cierre al final del bloque de código. `csv.reader()` facilita la lectura de un archivo CSV línea por línea y extrae sus valores. La línea `next(reader)` se salta la fila de cabecera, de modo que la siguiente línea `for name, email, grade in reader:` se divide filas subsiguientes en cada coma, y almacena los valores resultantes en las variables `name`, `email` y `grade` para el contacto actual.

Si los valores en el archivo CSV contienen espacios en blanco en uno o ambos lados, podemos eliminarlos usando el método `.strip()`.

Contenido personalizado

Podemos poner contenido personalizado en un mensaje usando `str.format()` para completar los marcadores de posición entre corchetes como hemos hecho en el print del código anterior.

Con esto en mente, podemos configurar un cuerpo de mensaje general, con marcadores de posición que se pueden adaptar a las personas.

Ejemplo de código

El siguiente ejemplo de código nos permite enviar correos electrónicos personalizados a varios contactos. Se repite un archivo CSV con `name,email,grade` para cada contacto, como en el ejemplo anterior.

El mensaje general que se define en el principio del script, y para cada contacto en el archivo CSV sus marcadores de posición `{}` y `{}` se rellenan, y un correo electrónico personalizado se envía a través de una conexión segura con el servidor de Gmail, como se vio antes:

```
import csv, smtplib, ssl, getpass

message = """Subject: Your grade

Hi {}, your grade is {}"""
```

```
from_address = "pruebasDAW2021@gmail.com"
##password = input("Type your password and press enter: ")
password = getpass.getpass('Password:')

context = ssl.create_default_context()
with smtplib.SMTP_SSL("smtp.gmail.com", 465, context=context) as
server:
    server.login(from_address, password)
    with open("contacts_file.csv") as file:
        reader = csv.reader(file)
        next(reader)  # Omitimos la primera linea
        for name, email, grade in reader:
            server.sendmail(
                from_address,
                email,
                message.format(name,grade))
            print("Enviado mail a {}".format(name))
```

Yagmail

Hay varias bibliotecas diseñadas para facilitar el envío de correos electrónicos, como Envelopes , Flanker y Yagmail . Yagmail está diseñado para funcionar específicamente con Gmail y simplifica enormemente el proceso de envío de correos electrónicos a través de una API amigable , como se puede ver en el ejemplo de código a continuación.

sendMail_YAGMAIL.py:

```
#pip install yagmail
#pip install keyring
import yagmail

receiver = "pruebasDAW2021@gmail.com"
body = "Hola desde Yagmail!!!!"
filename = "documento.pdf"

yag = yagmail.SMTP("pruebasDAW2021@gmail.com")
yag.send(
    to=receiver,
    subject="Yagmail test con un archivo adjunto",
    contents=body,
    attachments=filename,
)

print("Enviado con EXITO!!!")
```

Este ejemplo de código envía un correo electrónico con un archivo PDF adjunto en una fracción de las líneas necesarias para nuestro ejemplo usando `email` y `smtplib`.

Al configurar Yagmail, puede agregar sus validaciones de Gmail al llavero de su sistema operativo, como se describe en la documentación. Si no lo hace, Yagmail le pedirá que ingrese su contraseña cuando sea necesario y la guardará en el llavero automáticamente.

Servicios de correo electrónico transaccional

Si planeamos enviar un gran volumen de correos electrónicos, deseamos ver estadísticas de correo electrónico y deseamos garantizar una entrega confiable, puede valer la pena buscar en los servicios de correo electrónico transaccional. Aunque todos los siguientes servicios tienen planes pagos para enviar grandes volúmenes de correos electrónicos, también vienen con un plan gratuito para que pueda probarlos. Algunos de estos planes gratuitos son válidos indefinidamente y pueden ser suficientes para sus necesidades de correo electrónico.

A continuación, se muestra una descripción general de los planes gratuitos para algunos de los principales servicios de correo electrónico transaccional. Al hacer clic en el nombre del proveedor, accederá a la sección de precios de su sitio web.

Proveedor	Plan gratuito
Sendgrid	40.000 correos electrónicos durante los primeros 30 días, luego 100 / día
Sendinblue	300 correos electrónicos / día
Mailgun	Primeros 10,000 correos electrónicos gratis
Mailjet	200 correos electrónicos / día
Amazon SES	62.000 correos electrónicos / mes

Puede ejecutar una búsqueda en Google para ver qué proveedor se adapta mejor a sus necesidades, o simplemente probar algunos de los planes gratuitos para ver con qué API le gusta trabajar más.

Ejemplo de código de Sendgrid

Aquí hay un ejemplo de código para enviar correos electrónicos con Sendgrid para ver una idea de cómo usar un servicio de correo electrónico transaccional con Python:

```
import os
import sendgrid
from sendgrid.helpers.mail import Content, Email, Mail

sg = sendgrid.SendGridAPIClient(
    apikey=os.environ.get("SENDGRID_API_KEY")
)
from_email = Email("pruebasDAW2021@gmail.com")
to_email = Email("pruebasDAW2021@gmail.com")
subject = "Email test desde Sendgrid"
content = Content(
    "text/plain", "Aquí un mensaje desde Python con Sengrid"
)
mail = Mail(from_email, subject, to_email, content)
response = sg.client.mail.send.post(request_body=mail.get())

# Estos prints son para depuración y ver que todo ha ido de
forma correcta.
print(response.status_code)
print(response.body)
print(response.headers)
```

Para ejecutar este código, primero debe:

- Regístrese para obtener una cuenta Sendgrid (gratuita)
- Solicite una clave de API para la validación del usuario
- Agregue su clave de API escribiendo `setx SENDGRID_API_KEY "YOUR_API_KEY"`en el símbolo del sistema (para almacenar esta clave de API de forma permanente) o `set SENDGRID_API_KEY YOUR_API_KEY`para almacenarla solo para la sesión actual del cliente

Podemos encontrar más información sobre cómo configurar Sendgrid para Mac y Windows en el archivo README del repositorio en Github .

Conclusión

¡Ahora podemos iniciar una conexión SMTP segura y enviar múltiples correos electrónicos personalizados a las personas en su lista de contactos!

Has aprendido a enviar un correo electrónico HTML con una alternativa de texto sin formato y adjuntar archivos a sus correos electrónicos.
El paquete Yagmail simplifica todas estas tareas cuando usa una cuenta de Gmail. Si planea enviar grandes volúmenes de correo electrónico, vale la pena considerar los servicios de correo electrónico transaccional.

Disfrutad enviando correos electrónicos con Python y recordad: ¡ No ser gremlins malos ! JE JE JE