CFG Advanced Python Course - Session 4 (Summer 2016)

Using external APIs

APIs

APIs are of vital importance to developers; they enable you to programmatically access data from various different services such as Twitter, Facebook and Spotify to name a few. APIs enable you to create applications powered by data from any service of your liking.

Mailgun

When building a website, for example a landing page for what is to come, one vital thing to do is capture the details of people that might be interested in what you are going to launch.

Most importantly, you want to capture their email address so you can communicate things to them. In this session we will learn how to capture a user's e-mail address (and any other details for that matter) and programmatically send them an email to confirm that they have signed up to your awesome website's newsletter!

This will be an interactive session based on what we did in **Session 3**. if you look at what we did during that session under the **GET & POST** section, you will see that we already learnt how to capture a user's name and e-mail and print it out on our terminal. Now we are going to use that data to send an e-mail.

We will be using Mailgun, an email service for developers - or in simpler terms, a service that enables you to programmatically send e-mails. One great thing about Mailgun is that it lets you send up to 10,000 e-mails for free every month, so no need to worry about any costs!

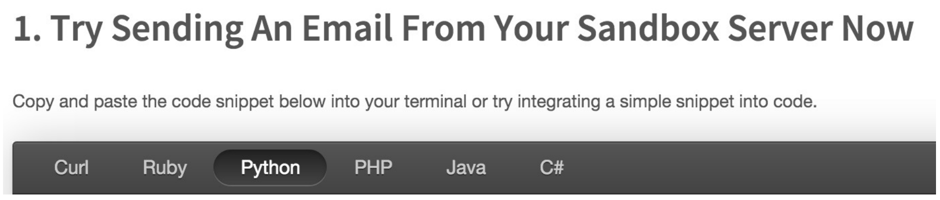
Mailgun makes use of a very popular Python library - requests - so let's install it using pip and get started.

pip install requests

In short, requests enables you to - as the name implies - make a request! You can for example fetch data, using a **GET HTTP** request, or pass some data using a **POST HTTP** request.

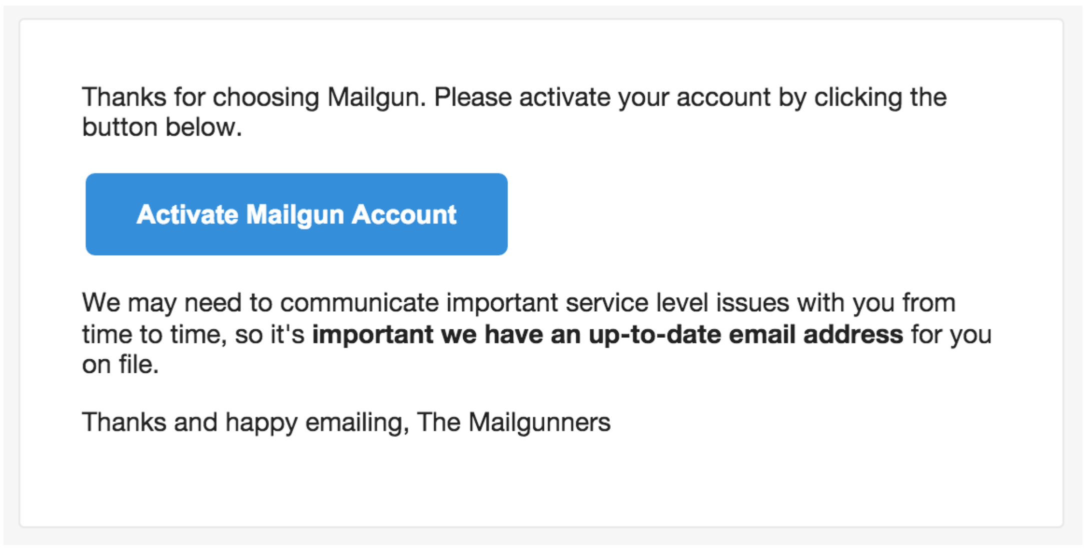
Using external APIs - Session 4

Head over to the Mailgun signup page and create an account. You should now be all set and ready to start sending e-mails from a Sandbox account - essentially a test account. Make sure you pick Python from the available tabs:


1. Try Sending An Email From Your Sandbox Server Now

Copy and paste the code snippet below into your terminal or try integrating a simple snippet into code.

Before doing anything else, head over to your e-mail and make sure you activate your account:

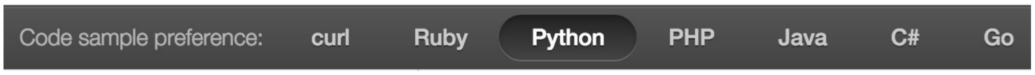

Thanks for choosing Mailgun. Please activate your account by clicking the button below.

We may need to communicate important service level issues with you from time to time, so it's **important we have an up-to-date email address** for you on file.

Thanks and happy emailing, The Mailgunners

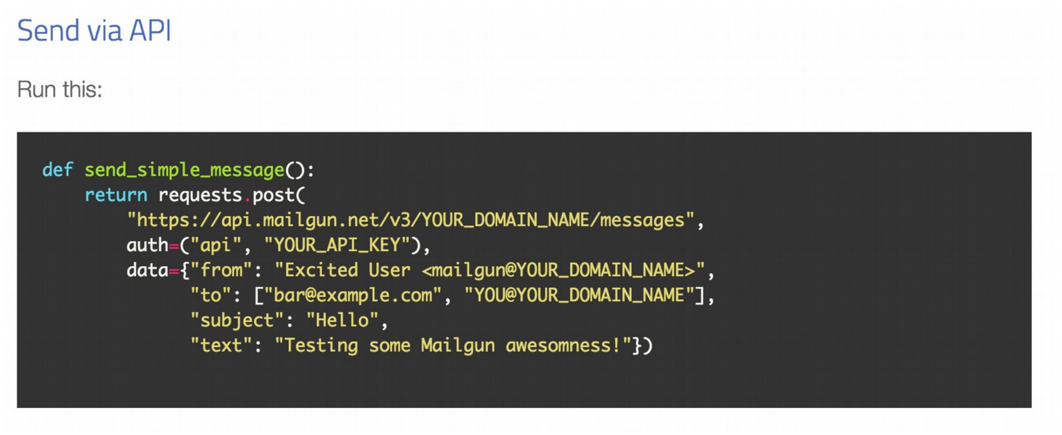
After you click on that link you will likely be asked to enter your phone number to verify that you are a genuine person and not someone trying to create multiple free accounts to spam people via e-mail! Go ahead and do that and you should be all set to start using Mailgun.

Using requests makes it very easy to send e-mails, so go ahead and head over to the Mailgun documentation page. On the Code Sample Preferences tab, make sure you click on Python:



Using external APIs - Session 4

Click on “Quickstart Guide” and then “Sending E-mail”. Scroll down to the Send via API section and let’s make use of this code snippet:

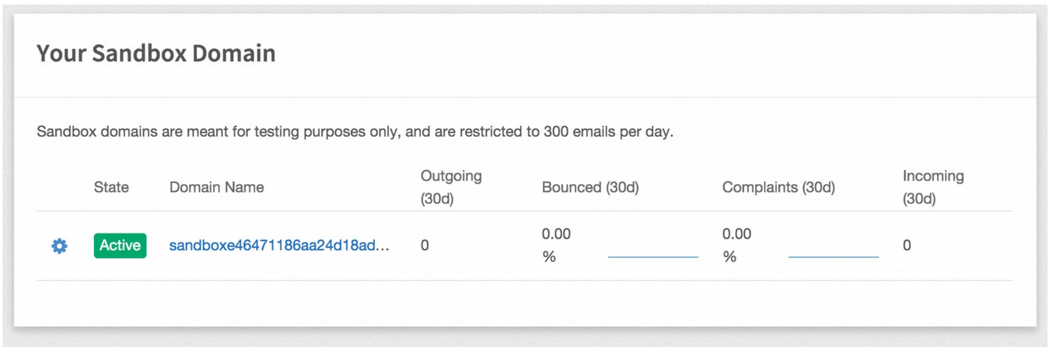


Send via API

Run this:

```
def send_simple_message():
    return requests.post(
        "https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/messages",
        auth=("api", "YOUR_API_KEY"),
        data={"from": "Excited User <mailgun@YOUR_DOMAIN_NAME>",
              "to": ["bar@example.com", "YOU@YOUR_DOMAIN_NAME"],
              "subject": "Hello",
              "text": "Testing some Mailgun awesomness!"})
```

There is some information we need to amend, so let’s get right to it - YOUR_DOMAIN_NAME and YOUR_API_KEY.
In order to programmatically send e-mails, we need to get an API guy, so head over to your Mailgun Dashboard and scroll down to the section called your Sandbox Domain:



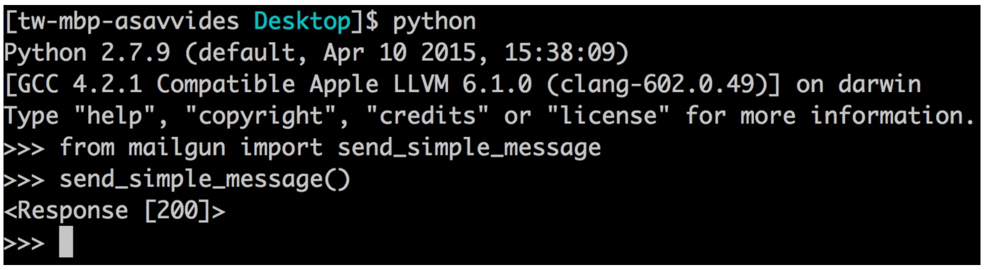
Your Sandbox Domain

Sandbox domains are meant for testing purposes only, and are restricted to 300 emails per day.

	State	Domain Name	Outgoing (30d)	Bounced (30d)	Complaints (30d)	Incoming (30d)
	Active	sandboxe46471186aa24d18ad...	0	0.00 %	0.00 %	0

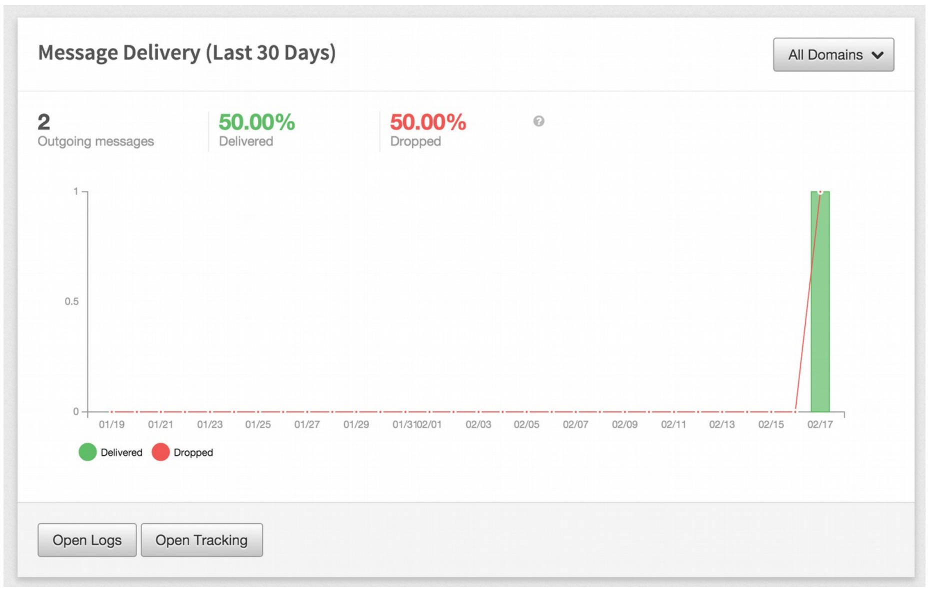
Click on what’s under the Domain Name, which should start with sandbox - this will take you to a page with all the information you need to make use of Mailgun. You’ll see there is an API Key listed and the Domain Name is that long URL starting with sandbox. Amend the code above using your API Key and Domain name. Save the file and let’s test it out from the command line in an interactive session:

Using external APIs - Session 4



```
[tw-mbp-asavvides Desktop]$ python
Python 2.7.9 (default, Apr 10 2015, 15:38:09)
[GCC 4.2.1 Compatible Apple LLVM 6.1.0 (clang-602.0.49)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> from mailgun import send_simple_message
>>> send_simple_message()
<Response [200]>
>>>
```

Getting back a response of 200 means everything went well as far as Mailgun knows. We can check our Mailgun dashboard to see whether an e-mail has been sent or not:



If there is a failure - marked as Dropped and highlighted in red - you can see the details of that failure but clicking on “Open Logs” - this will give you a list of all e-mail send attempts and whether they were successful or not, along with a representative error message.

This example is completely independent from a Flask app, so what you should do now is figure out how to apply the above code in the context of your Flask app that we created over the past few sessions.

To be more precise, create a form that takes a user’s e-mail and name - when that form is submitted, you should have a route in your Flask app that accepts a POST request and access those submitted values (look at last week’s notes to figure out how to do this). Using this information, use the requests library to trigger an e-mail send to that user. Add whatever content you want! Ensure that you are sending an email to a valid email address, set your own e-mail address as the **from**.

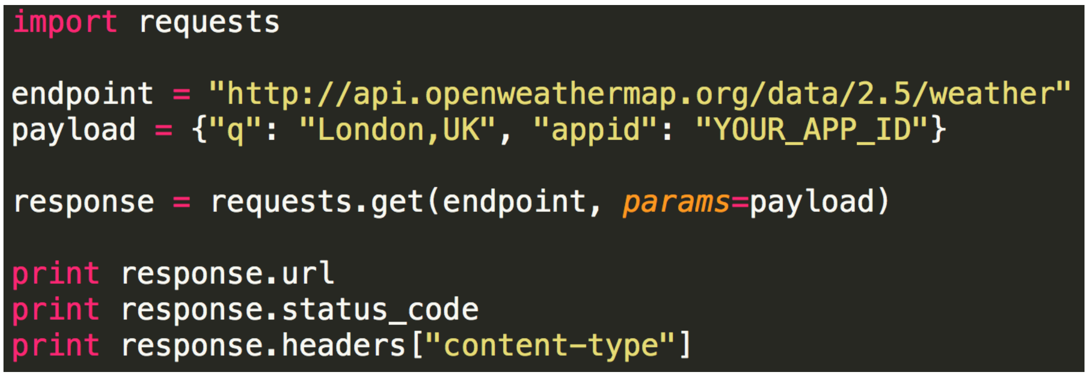
Have a look at the Mailgun API Reference for Sending Messages - can you figure out how to:

- Send a message written and styled with HTML?
- Attach a file to an e-mail?
- Can you set a specific delivery time to an e-mail (so that an e-mail is not sent immediately, but say 5 minutes later or at a specific time)? What's the maximum time in the future messages can be scheduled for?

Let's play with the weather API made available by OpenWeatherMap, it will allow us to query the weather for a given city and use the data in our web app.

Start off by creating a new file, say **weather.py** and add the following content:

Using external APIs - Session 4



```python
import requests

endpoint = "http://api.openweathermap.org/data/2.5/weather"
payload = {"q": "London,UK", "appid": "YOUR_APP_ID"}

response = requests.get(endpoint, params=payload)

print response.url
print response.status_code
print response.headers["content-type"]
```

There is some information that you need to substitute though - that's **YOUR_APP_ID**. For testing purposes, you can just use **"44db6a862fba0b067b1930da0d769e98"**.

The *appid* is a way for the API owners to identify you as a "customer" of their API. Very often, when using an API, you will need to register and use private ids to identify yourself. This way, API owners can identify you and monitor your usage.

We print some of the information we get from the response:

- *response.url*: This is the actual URL that requests used to get a response for you. You can copy and paste it in your browser; you will get the same response that requests got back.
- *response.status_code*: HTTP status code of the response. In HTTP, when a request succeeds, we expect a 200 Ok (More on Wikipedia)
- *response.headers['content-type']*: Content type of the response defined in the HTTP header.

So now you are surely wondering how to access the data from the response; you can access the response content using *response.text*.

Try it: add a new print statement at the end of the weather.py file:

print response.text

And run the script again!

You might now wonder what are all these ***u*** before the 'strings'. These are unicode strings, which can include many characters which are not possible in normal strings such as characters from other alphabets (Ж, ▯, ش) and emoji (,▯). Unicode encoding of strings is often used in web applications as they may have to handle text input from users all around the world.

Using external APIs - Session 4

Ok so we get the API response, but we would like to print a nice sentence about the weather, not the raw output of the API. Let's do it. First, we are going to convert the response in JSON and store it in a variable:

data = response.json()

Note: do not hesitate to print a variable when you need it, it's a good way to see what the data stored inside the variable look like. ***(Tip: print data)***

To access a JSON field we can use this notation - *data['main']['temp']*. Give it a go:

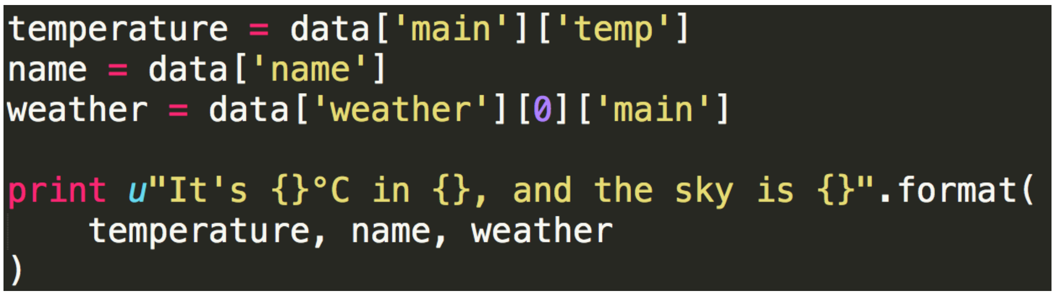
print data['main']['temp']

Does that look ok? Don’t you think the temperature value, *temp*, is a bit odd?

According to the OpenWeatherMap API documentation the default unit for temperature is Kelvin. Let’s change this by passing a new parameter to our URL: units=metric or units=imperial, update the payload variable:

payload = { 'q': 'London,UK', 'units': 'metric', 'appid': 'YOUR_APP_ID' }

Try printing a nice output for our script now:



```
temperature = data['main']['temp']
name = data['name']
weather = data['weather'][0]['main']

print u"It's {}°C in {}, and the sky is {}".format(
    temperature, name, weather
)
```

Given your knowledge now, can you figure out how to use Giphy’s API to get the first GIF returned from doing a search for a user’s favorite animal?

Note: Download this handy tool for viewing JSON files that are easy to read within Chrome