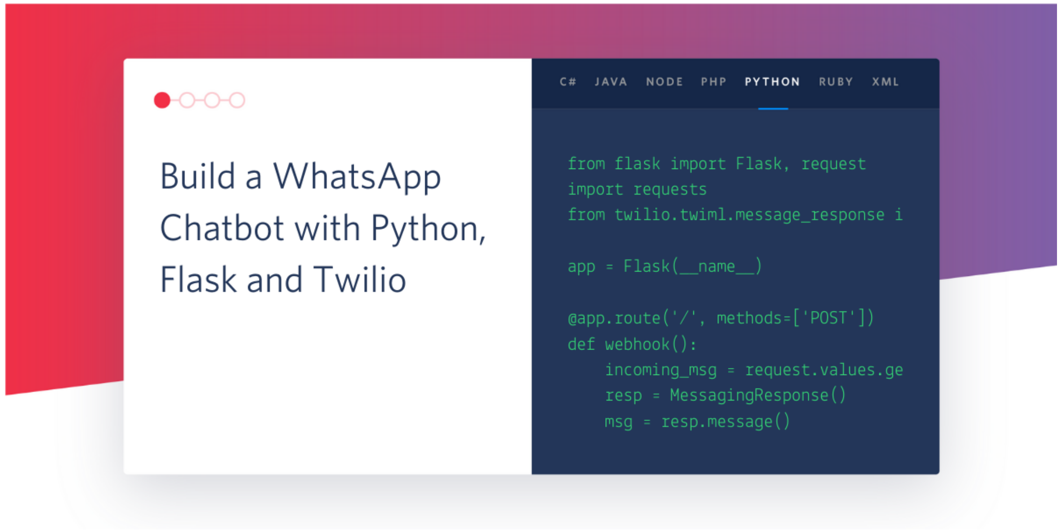
BY MIGUEL GRINBERG • 2019-11-20

TWITTER FACEBOOK LINKEDIN

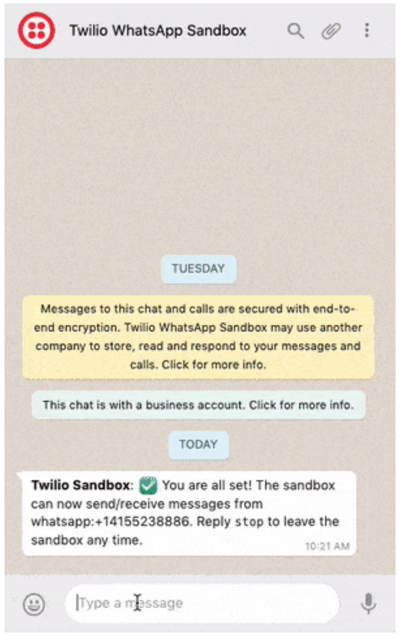
Build a WhatsApp Chatbot With Python, Flask and Twilio


Build a WhatsApp Chatbot with Python, Flask and Twilio
C# JAVA NODE PHP PYTHON RUBY XML

```
from flask import Flask, request
import requests
from twilio.twiml.message_response i

app = Flask(__name__)

@app.route('/', methods=['POST'])
def webhook():
    incoming_msg = request.values.ge
    resp = MessagingResponse()
    msg = resp.message()
```

A chatbot is a software application that is able to conduct a conversation with a human user through written or spoken language. The level of "intelligence" among chatbots varies greatly. While some chatbots have a fairly basic understanding of language, others employ sophisticated artificial intelligence (AI) and machine learning (ML) algorithms to achieve an almost human conversational level.

In this tutorial I'm going to show you how easy it is to build a chatbot for WhatsApp using the Twilio API for WhatsApp and the Flask framework for Python. Below you can see an example interaction I had with this chatbot:



Messages to this chat and calls are secured with end-to-end encryption. Twilio WhatsApp Sandbox may use another company to store, read and respond to your messages and calls. Click for more info.

This chat is with a business account. Click for more info.

Twilio Sandbox: ✅ You are all set! The sandbox can now send/receive messages from whatsapp:+14155238886. Reply `stop` to leave the sandbox any time. 10:21 AM

Tutorial Requirements

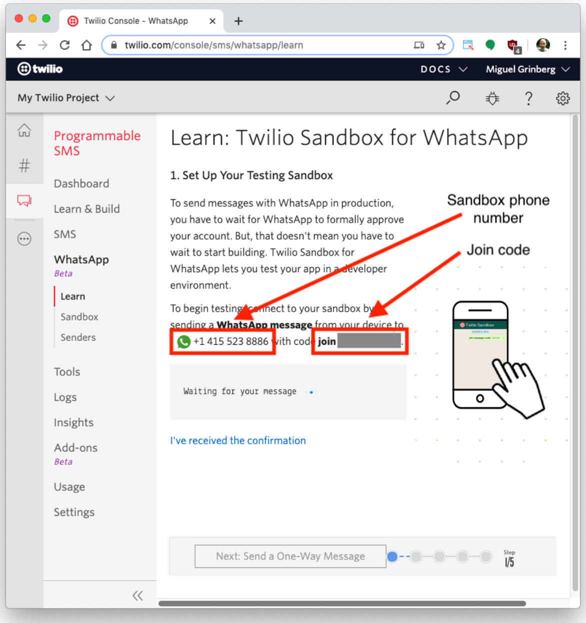
To follow this tutorial you need the following components:

- Python 3.6 or newer. If your operating system does not provide a Python interpreter, you can go to python.org to download an installer.
- Flask. We will create a web application that responds to incoming WhatsApp messages with it.
- ngrok. We will use this handy utility to connect the Flask application running on your system to a public URL that Twilio can connect to. This is necessary for the development version of the chatbot because your computer is likely behind a router or firewall, so it isn't directly reachable on the Internet. If you don't have ngrok it installed, you can download a copy for Windows, MacOS or Linux.
- A smartphone with an active phone number and WhatsApp installed.
- A Twilio account. If you are new to Twilio create a free account now. You can review the features and limitations of a free Twilio account.

Configure the Twilio WhatsApp Sandbox

Twilio provides a WhatsApp sandbox where you can easily develop and test your application. Once your application is complete you can request production access for your Twilio phone number, which requires approval by WhatsApp.

Let's connect your smartphone to the sandbox. From your Twilio Console, select Programmable Messaging, then click on "Try it Out" and finally click on Try WhatsApp. The WhatsApp sandbox page will show you the sandbox number assigned to your account, and a join code.


Learn: Twilio Sandbox for WhatsApp

1. Set Up Your Testing Sandbox

To send messages with WhatsApp in production, you have to wait for WhatsApp to formally approve your account. But, that doesn't mean you have to wait to start building. Twilio Sandbox for WhatsApp lets you test your app in a developer environment.

To begin testing, connect to your sandbox by sending a **WhatsApp message** from your device to +1 415 523 8886 with code **join** [redacted].

```
Waiting for your message
```

I've received the confirmation

To enable the WhatsApp sandbox for your smartphone send a WhatsApp message with the given code to the number assigned to your account. The code is going to begin with the word **join**, followed by a randomly generated two-word phrase. Shortly after you send the message you should receive a reply from Twilio indicating that your mobile number is connected to the sandbox and can start sending and receiving messages.

Note that this step needs to be repeated for any additional phones you'd like to have connected to your sandbox.

Create a Python Virtual Environment

Following Python best practices, we are going to make a separate directory for our chatbot project, and inside it we are going to create a virtual environment. We then are going to install the Python packages that we need for our chatbot on it.

If you are using a Unix or Mac OS system, open a terminal and enter the following commands to do the tasks described above:

```
$ mkdir whatsapp-bot
$ cd whatsapp-bot
$ python3 -m venv whatsapp-bot-venv
$ source whatsapp-bot-venv/bin/activate
(whatsapp-bot-venv) $ pip install twilio flask requests
```

For those of you following the tutorial on Windows, enter the following commands in a command prompt window:

```
$ mkdir whatsapp-bot
$ cd whatsapp-bot
$ python3 -m venv whatsapp-bot-venv
$ whatsapp-bot-venvScripts\activate

(whatsapp-bot-venv) $ pip install twilio flask requests
```

The last command uses `pip`, the Python package installer, to install the three packages that we are going to use in this project, which are:

- The Flask framework, to create the web application
- The Twilio Python Helper library, to work with the Twilio APIs
- The Requests package, to access third party APIs

For your reference, at the time this tutorial was released these were the versions of the above packages and their dependencies tested:

```
certifi==2019.9.11
chardet==3.0.4
Click==7.0
Flask==1.1.1
idna==2.8
itsdangerous==1.1.0
Jinja2==2.10.3
MarkupSafe==1.1.1
PyJWT==1.7.1
pytz==2019.3
requests==2.22.0
six==1.13.0
twilio==6.33.1
urllib3==1.25.7
Werkzeug==0.16.0
```

Create a Flask Chatbot Service

Now we are on to the fun part. Let's build a chatbot!

The type of chatbot that will work best for you is going to be largely dependent on your particular needs. For this tutorial I'm going to build an extremely simple chatbot that recognizes two keywords in messages sent by the user and reacts to them. If the user writes anything that contains the word "quote", then the chatbot will return a random famous quote. If instead the message has the word "cat", then a random cat picture will be returned. If both "quote" and "cat" are present in the message, then the bot will respond with a quote and a cat picture together.

Webhook

The Twilio API for WhatsApp uses a webhook to notify an application when there is an incoming message. Our chatbot application needs to define an endpoint that is going to be configured as this webhook so that Twilio can communicate with it.

With the Flask framework it is extremely easy to define a webhook. Below is a skeleton application with a webhook definition. Don't worry about copying this code, I will first show you all the different parts of the implementation and then once you understand them I'll show you how they are all combined into a working application.

```python
from flask import Flask

app = Flask(__name__)


@app.route('/bot’, methods=['POST'])
def bot():
    # add webhook logic here and return a response


if __name__ == '__main__':
    app.run()
```

If you are not familiar with the Flask framework, its documentation has a quick start section that should bring you up to speed quickly. If you want a more in-depth learning resource then I recommend you follow my Flask Mega-Tutorial.

The important thing to keep in mind about the code above is that the application defines a `/bot` endpoint that listens to `POST` requests. Each time an incoming message from a user is received by Twilio, they will in turn invoke this endpoint. The body of the function `bot()` is going to analyze the message sent by the user and provide the appropriate response.

Messages and Responses

The first thing we need to do in our chatbot is obtain the message entered by the user. This message comes in the payload of the `POST` request with a key of `’Body’`. We can access it through Flask’s `request` object:

```python
from flask import request
incoming_msg = request.values.get('Body', '').lower()
```

Since we are going to perform some basic language analysis on this text, I have also converted the text to lowercase, so that we don’t have to worry about all the different ways a word can appear when you introduce case variations.

The response that Twilio expects from the webhook needs to be given in TwiML or Twilio Markup Language, which is an XML-based language. The Twilio helper library for Python comes with classes that make it easy to create this response without having to create XML directly. Below you can see how to create a response that includes text and media components:

```python
from twilio.twiml.messaging_response import MessagingResponse

resp = MessagingResponse()
msg = resp.message()
msg.body('this is the response text')
msg.media('https://example.com/path/image.jpg')
```

Note how to return an image Twilio expects a URL that points to it instead of the actual image data.

Chatbot logic

For the actual chatbot logic I’m going to use a very simple, yet surprisingly effective approach. What I’m going to do is search the incoming messages for the keywords `’quote’` and `’cat’`. Here is the basic structure of the chatbot.

`'cat'`. Here is the basic structure of the chatbot:

```
    responded = False
    if 'quote' in incoming_msg:
        # add a quote to the response here
        responded = True
    if 'cat' in incoming_msg:
        # add a cat picture to the response here
        responded = True
    if not responded:
        # return a generic response here
```

With this simple structure we can detect references to quotes and/or cats and configure the Twilio response object accordingly. The `responded` boolean is useful to track the case where the message does not include any of the keywords we are looking for, and in that case offer a generic response.

Third-Party APIs

To supply the chatbot with original quotes and cat pictures I'm going to use two publicly available APIs. For famous quotes, I've chosen the Quotable API from Luke Peavey. A `GET` request to https://api.quotable.io/random returns a random quote out of a pool of 1500 of them in JSON format.

For cat pictures I'm going to use the Cat as a Service API from Kevin Balicot. This is an extremely simple API, the https://cataas.com/cat URL returns a different cat image every time (you can test it out by pasting this URL in the browser's address bar and then hitting refresh to get a new cat picture). This is actually very handy because as I mentioned above, Twilio wants the image given as a URL when preparing the TwiML response.

Everything Together

Now you have seen all the aspects of the chatbot implementation, so we are ready to integrate all the pieces into the complete chatbot service. You can copy the code below into a *bot.py* file:

```
from flask import Flask, request
import requests
from twilio.twiml.messaging_response import MessagingResponse

app = Flask(__name__)


@app.route('/bot', methods=['POST'])
def bot():
    incoming_msg = request.values.get('Body', '').lower()
    resp = MessagingResponse()
    msg = resp.message()
    responded = False
    if 'quote' in incoming_msg:
        # return a quote
        r = requests.get('https://api.quotable.io/random')
        if r.status_code == 200:
            data = r.json()
            quote = f'{data["content"]} ({data["author"]})'
        else:
            quote = 'I could not retrieve a quote at this time, sorry.'
        msg.body(quote)
        responded = True
    if 'cat' in incoming_msg:
        # return a cat pic
        msg.media('https://cataas.com/cat')
        responded = True
    if not responded:
        msg.body('I only know about famous quotes and cats, sorry!')
    return str(resp)


if __name__ == '__main__':
    app.run()
```

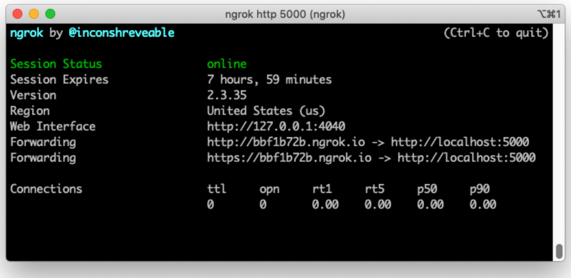
Testing the Chatbot

Are you ready to test the chatbot? After you copy the above code into the *bot.py* file, start the chatbot by running `python bot.py`, making sure you do this while the Python virtual environment is activated. The output should be something like this:

```
(whatsapp-bot-venv) $ python bot.py
 * Serving Flask app "bot" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

The service is now running as a private service on port 5000 inside your computer and will sit there waiting for incoming connections. To make this service reachable from the Internet we need to use ngrok.

Open a second terminal window and run `ngrok http 5000` to allocate a temporary public domain that redirects HTTP requests to our local port 5000. On a Unix or Mac OS computer you may need to use `./ngrok http 5000` if you have the ngrok executable in your current directory. The output of ngrok should be something like this:

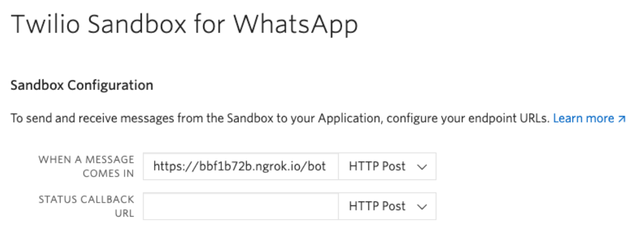


```
ngrok by @inconshreveable                                       (Ctrl+C to quit)

Session Status                online
Session Expires               7 hours, 59 minutes
Version                       2.3.35
Region                        United States (us)
Web Interface                 http://127.0.0.1:4040
Forwarding                    http://bbf1b72b.ngrok.io -> http://localhost:5000
Forwarding                    https://bbf1b72b.ngrok.io -> http://localhost:5000

Connections                   ttl     opn     rt1     rt5     p50     p90
                              0       0       0.00    0.00    0.00    0.00
```

Note the lines beginning with “Forwarding”. These show the public URL that ngrok uses to redirect requests into our service. What we need to do now is tell Twilio to use this URL to send incoming message notifications.

Go back to the Twilio Console, click on Programmable Messaging, then on Settings, and finally on WhatsApp Sandbox Settings. Copy the https:// URL from the ngrok output and then paste it on the “When a message comes in” field. Since our chatbot is exposed under the `/bot` URL, append that at the end of the root ngrok URL. Make sure the request then paste it on the "When a message comes in" field. Since our chatbot is exposed under the `/bot` URL, append that at the end of the root ngrok URL. Make sure the request

method is set to `HTTP Post`. Don't forget to click the red Save button at the bottom of the page to record these changes.



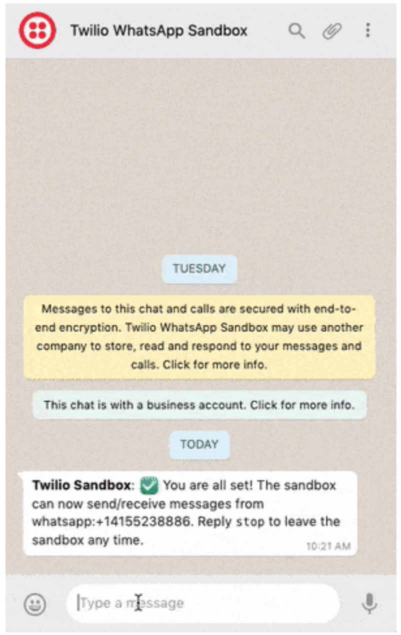
Twilio Sandbox for WhatsApp

Sandbox Configuration

To send and receive messages from the Sandbox to your Application, configure your endpoint URLs. Learn more ↗

WHEN A MESSAGE COMES IN	https://bbf1b72b.ngrok.io/bot	HTTP Post
STATUS CALLBACK URL		HTTP Post

Now you can start sending messages to the chatbot from the smartphone that you connected to the sandbox. You can type any sentences that you like, and each time the words "quote" and "cat" appear in messages the chatbot will invoke the third party APIs and return some fresh content to you. In case you missed it at the top of the article, here is an example session that I held with the chatbot:



Twilio WhatsApp Sandbox

TUESDAY

Messages to this chat and calls are secured with end-to-end encryption. Twilio WhatsApp Sandbox may use another company to store, read and respond to your messages and calls. Click for more info.

This chat is with a business account. Click for more info.

TODAY

Twilio Sandbox: ✅ You are all set! The sandbox can now send/receive messages from whatsapp:+14155238886. Reply `stop` to leave the sandbox any time. 10:21 AM

Keep in mind that when using ngrok for free there are some limitations. In particular, you cannot hold on to an ngrok URL for more than 8 hours, and the domain name that is assigned to you will be different every time you start the ngrok command. You will need to update the URL in the Twilio Console every time you restart ngrok.

Notes on Production Deployment

I thought it would be useful to end this tutorial with a list of things you will need to consider if you decide to deploy a WhatsApp chatbot for production use.

First of all, you've seen that when you start the Flask application there is a pretty scary warning about not using a development server for production. The web server that comes with Flask is very convenient when developing and testing an application, but it isn't robust enough to handle the demands of production use. The two most common production ready web servers for Python web applications are gunicorn and uWSGI, both installable on your virtual environment with `pip`. For example, here is how to run the chatbot with gunicorn:

```
(whatsapp-bot-venv) $ gunicorn -b :5000 bot:app
```

```
(whatsapp-bot-venv) $ gunicorn -b :5000 bot:app
```

Also keep in mind that for a production deployment you will be running the service on a cloud server and not out of your own computer, so there is no need to use ngrok.

Conclusion

Businesses are increasingly turning to chatbots to offer their customers immediate access to support, sales or marketing. A chatbot can be useful in gathering customer information before a human agent is involved, and a well designed chatbot should also be able to handle common customer workflows entirely without human assistance.

I hope this tutorial was useful and you now have a better idea of how to build your WhatsApp chatbot. Below you can find some additional resources on building chatbots with Python in case you want to see more implementations:

- Don't be a robot, build the bot: PyCon 2019 talk by Mariatta.
- Chatterbot: a Python-based machine-learning, conversational dialog engine.

I would love to see the chatbot that you build!

AUTHORS | Miguel Grinberg

Developer stories to your inbox.

Subscribe to the Developer Digest, a monthly dose of all things code.

Enter your email...

You may unsubscribe at any time using the unsubscribe link in the digest email. See our privacy policy for more information.

NEW!

Tutorials

Sample applications that cover common use cases in a variety of languages. Download, test drive, and tweak them yourself.

Get started

SIGN UP AND START BUILDING

Not ready yet? Talk to an expert.