* Python Bot WhatsApp: a step-by-step guide for developer

**Part 1: If you don’t have time to read**

* Respond to an unfamiliar command, this could be an instruction or your welcome message;
* Send regular message;
* Send image;
* Send file;
* Send video;
* Send contact (vCard);
* Send product;
* Create new group, send an invitation and send message to the group;
* Receive and reading incoming messages;

**[GitHub - Whapi-Cloud/python-whatsapp-chatbot: WhatsApp Python Bot. The basic functionality you will…](https://github.com/Whapi-Cloud/python-whatsapp-chatbot?source=post_page-----b11d600ad7f5--------------------------------" \t "_blank)**

[WhatsApp Python Bot. The basic functionality you will need when developing a bot: sending and receiving messages…](https://github.com/Whapi-Cloud/python-whatsapp-chatbot?source=post_page-----b11d600ad7f5--------------------------------" \t "_blank)

[github.com](https://github.com/Whapi-Cloud/python-whatsapp-chatbot?source=post_page-----b11d600ad7f5--------------------------------" \t "_blank)

**Part 2**

In the modern era of instant communication, WhatsApp stands out as one of the most widely used messaging platforms, boasting over 2 billion users worldwide. The application offers a range of functionalities from text messaging to voice and video calls, making it an essential communication tool in both personal and professional realms.



With the evolution of technology, the integration of automated bots within messaging platforms has become increasingly prevalent. Bots can automate repetitive tasks, provide instant responses, and even facilitate various business processes, making them invaluable assets in improving user interaction and engagement.

In this article, we aim to guide you through the process of developing a WhatsApp bot using Python, leveraging the capabilities of [Whapi.Cloud](https://whapi.cloud/" \t "_blank). Python, renowned for its simplicity and readability, is a versatile programming language suitable for a wide array of applications, including bot development.

This tutorial will cover the following aspects:

* **Preparing the Working Environment**: Setting up Python and relevant libraries.
* **Connecting to WhatsApp API**: Understanding and connecting to Whapi.Cloud.
* **Creating a Bot Framework**: Initializing a server and preparing the bot structure.
* **Sending Simple Messages**: Coding the bot to send messages.
* **Handling Incoming Messages**: Developing logic to process received messages.
* **Enhancing Functionality**: Adding advanced features like media messages, group management, and handling group messages.
* **Testing and Debugging**: Running local tests and solving common issues.
* **Deployment and Hosting**: Making the bot accessible to users.
* **Conclusion and Next Steps**: Final thoughts and future improvements.

By the end of this guide, you’ll have the foundational knowledge and practical experience necessary to develop and extend your own WhatsApp bot, paving the way for countless possibilities in enhancing communication and interaction through automated technology. Whether you are aiming to create a bot for customer support, content distribution, or task automation, this tutorial will provide you with the insights to kickstart your journey in WhatsApp bot development using Python.



**Preparing the Working Environment**

Before delving into the development of the bot, it is crucial to prepare the development environment by installing Python and setting up the necessary libraries and tools.

**Installing Python**

If you haven’t installed Python yet, you can download it from the [official Python website](https://www.python.org/downloads/). Python 3.6 or newer is recommended, as it includes many new features and optimizations. The installation process is straightforward, and you can follow the prompts to complete it.

**Setting Up a Virtual Environment**

Once Python is installed, it’s a good practice to create a virtual environment. A virtual environment is an isolated Python environment in which you can install libraries and dependencies without affecting the global Python installation. To create a virtual environment, open a terminal or command prompt and run the following commands:

$ mkdir whatsapp\_bot  
$ cd whatsapp\_bot  
$ python3 -m venv venv

To activate the virtual environment:

$ .\venv\Scripts\activate

On MacOS/Linux:

$ source venv/bin/activate

**Installing Necessary Libraries**

After activating the virtual environment, you’ll need to install the libraries essential for the development of the bot. For this tutorial, we are primarily using Flask for handling HTTP requests and the requests library for interacting with the Whapi.Cloud API.

$ pip install Flask requests

Choose an Integrated Development Environment (IDE) that you are comfortable with. PyCharm, Visual Studio Code, and Jupyter are excellent choices. Once chosen, configure your IDE to use the virtual environment you’ve created as the Python interpreter.

**Connecting to the WhatsApp API**

Whapi.Cloud is an API gateway that facilitates integration with WhatsApp. This makes it possible, with the Cloud API’s tight restrictions and expensive messages, to fully automate any messenger functionality. In this article, we will look at just a few methods for sending, receiving and processing messages in WhatsApp, but the possibilities of this tool are much more.

Connecting to the Whapi.Cloud for WhatsApp automation is a foundational step in creating your WhatsApp bot.

**Registration on Whapi.Cloud**

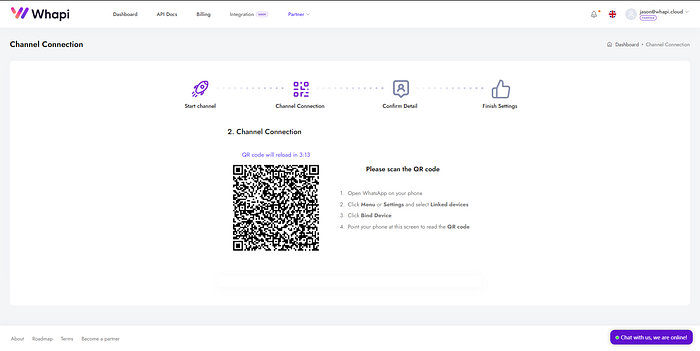
To kickstart your journey, [register on Whapi.Cloud](https://panel.whapi.cloud/register) by creating a free account. This step is straightforward and doesn’t require any credit card details. Post-registration, you gain immediate access to a test channel, albeit with some limitations.

**Connecting Your Phone**

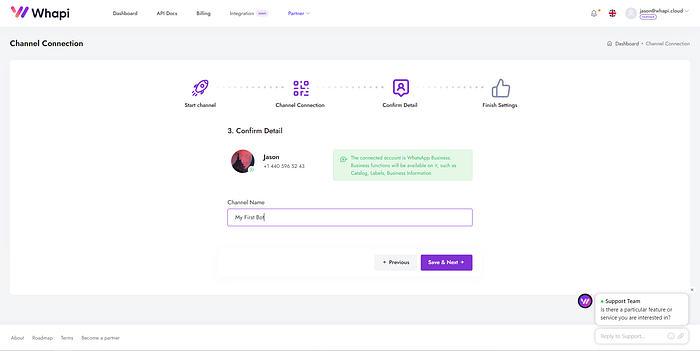
The test channel requires around a minute to initiate. After its activation, connect your phone to enable WhatsApp automation. The service will utilize your connected phone to send messages. Here’s a step-by-step process:

1. Access the QR code by clicking on your trial channel in your Whapi.Cloud account.
2. Open WhatsApp on your mobile device.
3. Navigate to Settings -> Linked devices -> Link a device -> Scan QR code.

This swift connection process is a standout feature of Whapi.Cloud, enabling you to launch and operate within minutes.



To connect your phone, use the QR code



**Channel Customization and Configuration**

During the second and third steps of the connection, you’ll be prompted to personalize your channel. You can name your channel, configure webhooks, and modify other settings as per your preference. However, these steps can be skipped initially and revisited later for refinement.

**Obtaining API Token**

Once your channel is operational, locate your API Token in the center block beneath the limits information. This token is pivotal for authenticating your API requests. Typically, it is included in the request headers as a Bearer Token or as a request parameter, adapting to the API method employed.

**Using Webhooks**

Webhooks are crucial for managing incoming messages effectively. They can be configured in the settings of your Whapi.Cloud account. When a message is received, the configured webhook URL will receive a POST request containing the message data, allowing your bot to process incoming messages and respond accordingly. We will delve deeper into the usage of webhooks in the upcoming sections.

**Send WhatsApp message using Python**

Once the preliminary setup is complete and your environment is prepared, the next fundamental step is sending a simple message. Python, with its rich library set, provides a succinct and straightforward way to perform this using the requests library, which is widely used for making HTTP requests.

Below is a basic example that demonstrates how to send a simple text message using Python.

import requests  
  
url = "https://gate.whapi.cloud/messages/text?token=YOUR\_TOKEN"  
  
payload = {  
 "typing\_time": 5,  
 "to": "1234567891@s.whatsapp.net",  
 "body": "Hello, world!"  
}  
headers = {  
 "accept": "application/json",  
 "content-type": "application/json"  
}  
  
response = requests.post(url, json=payload, headers=headers)  
  
print(response.text)

In this example:

* url: This is the endpoint provided by Whapi.Cloud for sending text messages, including the API token.
* payload: It holds the parameters like typing\_time (how long the user sees the "typing..." status), to (the recipient's WhatsApp number with the domain @s.whatsapp.net), and body (the actual message text).
* headers: These are used to set the content type to JSON.

*A list of all parameters is available in the documentation:*[*https://whapi.readme.io/reference/sendmessagetext*](https://whapi.readme.io/reference/sendmessagetext)

Executing the script will send a “Hello, world!” message to the specified WhatsApp number, and the server’s response will be printed to the console.

**Explanation**

The requests.post() method initiates an HTTP POST request to the specified URL. The json=payload argument is used to send the payload in JSON format, and the headers=headers argument sets the request headers to inform the server about the type of content being sent.

Once the message is successfully sent, the response.text will print the server's response, providing information about the success or failure of the message sending.

**Handling Incoming Messages**

Handling incoming messages is a pivotal step in the development of a WhatsApp bot. In this chapter, we will cover the concept of a webhook, its utility, the setting process via the dashboard or API, and the fundamental logic to handle and respond to incoming messages.

**What is a Webhook?**

A webhook is an HTTP callback that occurs when something happens; it’s a simple way for apps to communicate with each other. In the context of WhatsApp bots, a webhook is used to receive incoming messages or event notifications from WhatsApp users. Whapi.Cloud allows the setting of multiple hooks with various modes (Body, Path, Method) for different events (Message, Ack, Chat, Status), offering flexibility and extensive coverage for various use cases.

**Setting up Webhook via Interface and API**

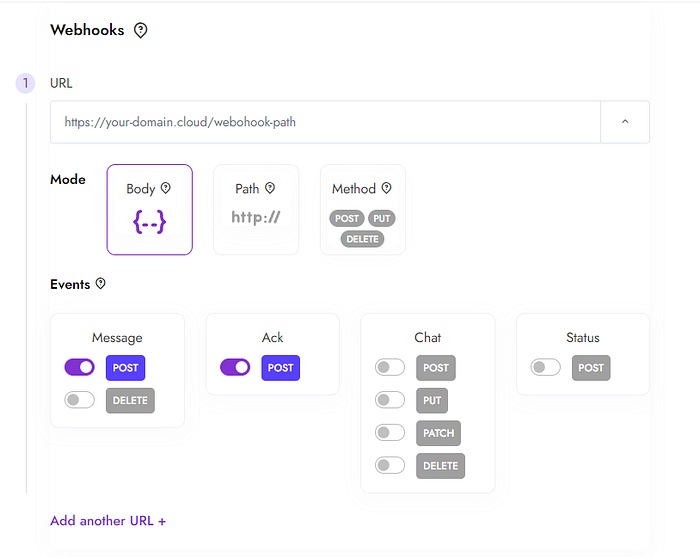
You can set up webhooks either through the interface on your personal account on Whapi.Cloud or programmatically via API. Below is a Python snippet on how to set it up through API:

import requests  
  
url = "https://gate.whapi.cloud/settings?token=YOUR\_TOKEN"  
  
payload = {  
 "callback\_backoff\_delay\_ms": 3000,  
 "max\_callback\_backoff\_delay\_ms": 900000,  
 "callback\_persist": True,  
 "pass\_through": True,  
 "sent\_status": False,  
 "webhooks": [  
 {  
 "events": [  
 {  
 "type": "message",  
 "method": "post"  
 },  
 {  
 "type": "ack",  
 "method": "post"  
 }  
 ],  
 "mode": "body",  
 "url": "<Webhook URL, http or https>"  
 }  
 ]  
}  
headers = {  
 "accept": "application/json",  
 "content-type": "application/json"  
}  
  
response = requests.patch(url, json=payload, headers=headers)  
  
print(response.text)

In this script:

* The url is the Whapi.Cloud endpoint to adjust settings, inclusive of the API token.
* The payload holds the webhook configurations.
* The webhooks array in the payload specifies the events and the URL where the webhook will send the HTTP POST request when the events occur.

The unique feature and advantage of Whapi.Cloud are that it allows setting up multiple webhooks with varying modes on different events, providing versatility and a broader range of interaction capabilities. This feature allows the development of more sophisticated and responsive bots, enhancing user interaction and experience.



Webhooks configuration interface in the personal cabinet

For creating a server that will handle incoming messages, we will be using Flask, a lightweight WSGI web application framework in Python. In this example, the bot will be responding to incoming messages and, depending on the received command, it will be sending different replies.

Below is a sample code for the bot (python code to send and receive whatsapp message):

from flask import Flask, request  
import requests  
  
app = Flask(\_\_name\_\_)  
  
@app.route('/webhook', methods=['POST'])  
def webhook():  
 # Retrieving incoming message  
 incoming\_message = request.json  
   
 # Retrieving the text of the message  
 message\_text = incoming\_message.get('body', '').lower()  
   
 # Deciding the reply based on the command  
 if message\_text == 'hello':  
 response\_text = 'Hi! How can I assist you today?'  
 elif message\_text == 'info':  
 response\_text = 'I am a WhatsApp bot created to assist you!'  
 else:  
 response\_text = 'I am sorry, I do not understand the command.'  
   
 # Sending the reply message  
 send\_message(response\_text, incoming\_message['from'])  
   
 return '', 200  
  
def send\_message(response\_text, to):  
 # URL to send messages through the Whapi.Cloud API  
 url = "https://gate.whapi.cloud/messages/text?token=YOUR\_TOKEN"  
  
 # Forming the body of the message  
 payload = {  
 "to": to,  
 "body": response\_text  
 }  
 headers = {  
 "accept": "application/json",  
 "content-type": "application/json"  
 }  
  
 # Sending the message  
 response = requests.post(url, json=payload, headers=headers)  
 print(response.text)  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 app.run(port=5000, debug=True)

In this sample:

* We are creating a web server using Flask and setting up a webhook on the /webhook endpoint.
* When the bot receives a new message, it analyzes the text of this message and sends an appropriate reply using the send\_message function, in which we employ the message-sending code from previous Chapter.

**WhatsApp Python chatbot source code**

Below is a simple Flask application that uses the provided Python script to send messages through the Whapi.Cloud API. It listens to incoming messages and sends appropriate responses based on the content. Create an .env file with the pre-saved configuration settings.

TOKEN=Your\_token # API token from your channel  
API\_URL=https://gate.whapi.cloud # API endpoint URL  
BOT\_URL=https://your\_bot\_url\_here/messages # Webhook Link to your server. At ( {server link}/messages ), when POST is requested, processing occurs  
PRODUCT\_ID=6559353560856703 # The ID of the product we will send for the example. Create a product in your WhatsApp and find out the product ID: https://whapi.readme.io/reference/getproducts  
GROUP\_ID=120363177599599603@g.us # The ID of the group to which we will send the message. Use to find out the ID: https://whapi.readme.io/reference/getgroups  
PORT=80 # example, 80 or 443

Let me remind you that the entire source code of this project can be downloaded from GitHub: <https://github.com/Whapi-Cloud/python-whatsapp-chatbot>

from flask import Flask, request, jsonify  
import requests  
from requests\_toolbelt.multipart.encoder import MultipartEncoder  
import os  
from dotenv import load\_dotenv  
  
load\_dotenv() # Load environment variables from a .env file  
  
app = Flask(\_\_name\_\_)  
  
COMMANDS = {  
 'TEXT': 'Simple text message',  
 'IMAGE': 'Send image',  
 'DOCUMENT': 'Send document',  
 'VIDEO': 'Send video',  
 'CONTACT': 'Send contact',  
 'PRODUCT': 'Send product',  
 'GROUP\_CREATE': 'Create group',  
 'GROUP\_TEXT': 'Simple text message for the group',  
 'GROUPS\_IDS': "Get the id's of your three groups"  
}  
  
FILES = {  
 'IMAGE': './files/file\_example\_JPG\_100kB.jpg',  
 'DOCUMENT': './files/file-example\_PDF\_500\_kB.pdf',  
 'VIDEO': './files/file\_example\_MP4\_480\_1\_5MG.mp4',  
 'VCARD': './files/sample-vcard.txt'  
}  
  
  
def send\_whapi\_request(endpoint, params=None, method='POST'):  
 headers = {  
 'Authorization': f"Bearer {os.getenv('TOKEN')}"  
 }  
 url = f"{os.getenv('API\_URL')}/{endpoint}"  
 if params:  
 if 'media' in params:  
 details = params.pop('media').split(';')  
 with open(details[0], 'rb') as file:  
 m = MultipartEncoder(fields={\*\*params, 'media': (details[0], file, details[1])})  
 headers['Content-Type'] = m.content\_type  
 response = requests.request(method, url, data=m, headers=headers)  
 elif method == 'GET':  
 response = requests.get(url, params=params, headers=headers)  
 else:  
 headers['Content-Type'] = 'application/json'  
 response = requests.request(method, url, json=params, headers=headers)  
 else:  
 response = requests.request(method, url, headers=headers)  
 print('Whapi response:', response.json())  
 return response.json()  
  
def set\_hook():  
 if os.getenv('BOT\_URL'):  
 settings = {  
 'webhooks': [  
 {  
 'url': os.getenv('BOT\_URL'),  
 'events': [  
 {'type': "message", 'method': "post"}  
 ],  
 'mode': "method"  
 }  
 ]  
 }  
 send\_whapi\_request('settings', settings, 'PATCH')  
  
  
@app.route('/messages', methods=['POST'])  
def handle\_new\_messages():  
 try:  
 messages = request.json.get('messages', [])  
 endpoint = None  
 for message in messages:  
 if message.get('from\_me'):  
 continue  
 sender = {'to': message.get('chat\_id')}  
 command\_input = message.get('text', {}).get('body', '').strip()  
 command = list(COMMANDS.keys())[int(command\_input) - 1] if command\_input.isdigit() else None  
  
 if command == 'TEXT':  
 sender['body'] = 'Simple text message'  
 endpoint = 'messages/text'  
 elif command == 'IMAGE':  
 sender['caption'] = 'Text under the photo.'  
 sender['media'] = FILES['IMAGE'] + ';image/jpeg'  
 endpoint = 'messages/image'  
 elif command == 'DOCUMENT':  
 sender['caption'] = 'Text under the document.'  
 sender['media'] = FILES['DOCUMENT'] + ';application/pdf'  
 endpoint = 'messages/document'  
 elif command == 'VIDEO':  
 sender['caption'] = 'Text under the video.'  
 sender['media'] = FILES['VIDEO'] + ';video/mp4'  
 endpoint = 'messages/video'  
 elif command == 'CONTACT':  
 sender['name'] = 'Whapi Test'  
 with open(FILES['VCARD'], 'r') as vcard\_file:  
 sender['vcard'] = vcard\_file.read()  
 endpoint = 'messages/contact'  
 elif command == 'PRODUCT':  
 # Example: You need to replace config.product with an actual product ID  
 product\_id = os.getenv('PRODUCT\_ID') # Replace with your product ID  
 endpoint = f'business/products/{product\_id}'  
 elif command == 'GROUP\_CREATE':  
 # Example: You need to replace config.phone with an actual phone number  
 participants = [message.get('chat\_id').split('@')[0]] # Replace with the phone number  
 response = send\_whapi\_request('groups', {'subject': 'Whapi.Cloud Test', 'participants': participants})  
 sender['body'] = f"Group created. Group id: {response.get('group\_id')}" if response.get('group\_id') else 'Error'  
 endpoint = 'messages/text'  
 elif command == 'GROUP\_TEXT':  
 sender['to'] = os.getenv('GROUP\_ID') # Replace with your group ID  
 sender['body'] = 'Simple text message for the group'  
 endpoint = 'messages/text'  
 elif command == 'GROUPS\_IDS':  
 groups\_response = send\_whapi\_request('groups', {'count': 3}, 'GET')  
 groups = groups\_response.get('groups', [])  
 sender['body'] = ',\n '.join(f"{group['id']} - {group['name']}" for group in groups) if groups else 'No groups'  
 endpoint = 'messages/text'  
 else:  
 sender['body'] = "Hi. Send me a number from the list. Don't forget to change the actual data in the code!\n\n" + \  
 '\n'.join(f"{i + 1}. {text}" for i, text in enumerate(COMMANDS.values()))  
 endpoint = 'messages/text'  
  
 if endpoint is None:  
 return 'Ok', 200  
 response = send\_whapi\_request(endpoint, sender)  
 print(f"Response from Whapi: {response}")  
 return 'Ok', 200  
  
 except Exception as e:  
 print(e)  
 return str(e), 500  
  
  
@app.route('/', methods=['GET'])  
def index():  
 return 'Bot is running'  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 set\_hook()  
 port = os.getenv('PORT') or (443 if os.getenv('BOT\_URL', '').startswith('https:') else 80)  
 app.run(port=port, debug=True)

For this to work: 1. Run the Flask application. 2. Set up your WhatsApp channel to receive messages on your *webhook*. 3. Try sending the message “hello” to your WhatsApp bot and see how it responds.

**Extending Functionality**

In this chapter, we will delve into extending the functionality of our WhatsApp bot, enabling it to send media messages, create groups, count users in a group, and handle incoming group messages.

**Sending WhatsApp Media Messages**

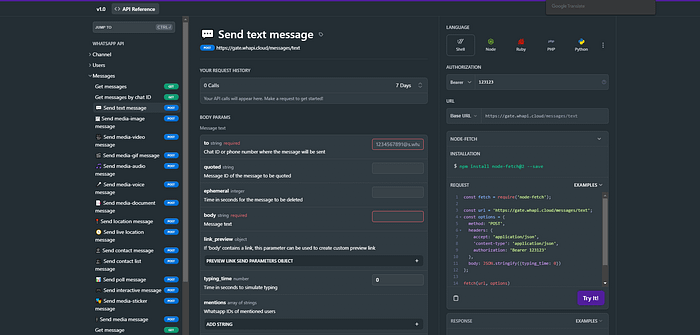
Sending media messages involves transmitting different types of media like images, videos, gifs, audio, voice notes, documents, and stickers through WhatsApp. Here is an example of how you can send an image:

import requests  
  
url = "https://gate.whapi.cloud/messages/media/image?caption=Name%20of%20pic&to=1234567891%40s.whatsapp.net&mentions=&token=YOUR\_TOKEN"  
  
payload = "data:image/png;name=Untitled2.png;base64,iVBORw0KGgoAAAANSUhEUgAAAx...V//5Tq42UXih5JAAAAAElFTkSuQmCC"  
headers = {  
 "accept": "application/json",  
 "content-type": "application/pdf"  
}  
  
response = requests.post(url, data=payload, headers=headers)  
  
print(response.text)

In the example above:

* url specifies the API endpoint for sending images, with a caption and a recipient included as parameters.
* payload contains the base64 encoded image along with its name.
* The content-type in headers specifies the type of media being sent, and in this case, it is an image.

This method can be generalized to send other types of media by modifying the url and content-type accordingly, ensuring they match the media being sent.



WhatsApp API documentation with examples of how to send different types of messages

**WhatsApp API Group Management**

**Retrieving Group Details and Counting Participants**

To retrieve the details of a group, including the participants, you can make a GET request to the respective endpoint provided by Whapi. Below is an example of how to retrieve details of a group:

import requests  
  
url = "https://gate.whapi.cloud/groups/120363175154208908%40g.us?token=YOUR\_TOKEN"  
  
headers = {"accept": "application/json"}  
  
response = requests.get(url, headers=headers)  
  
print(response.text)

The response from this request will contain various details about the group, such as its ID, name, type, timestamp, and a list of the participants along with their ranks within the group. Here’s an example of the response:

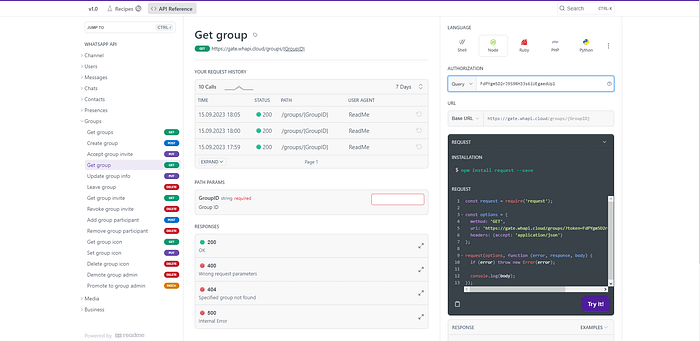
{  
 "id": "120363175154208908@g.us",  
 "name": "Hello",  
 "type": "group",  
 "timestamp": 1694784032,  
 "not\_spam": true,  
 "participants": [  
 {"id": "14406616972", "rank": "creator"},  
 {"id": "15057426924", "rank": "member"},  
 {"id": "15056452483", "rank": "member"}  
 ],  
 "name\_at": 1694612985,  
 "created\_at": 1694612985,  
 "created\_by": "14406616972"  
}

To count the number of participants in the group, you can simply calculate the length of the participants list in the response. Here’s how you can do it:

import requests  
import json  
  
url = "https://gate.whapi.cloud/groups/120363175154208908%40g.us?token=YOUR\_TOKEN"  
  
headers = {"accept": "application/json"}  
  
response = requests.get(url, headers=headers)  
  
group\_data = json.loads(response.text)  
participants\_count = len(group\_data['participants'])  
  
print(f'The number of participants in the group is {participants\_count}.')

In this example, json.loads(response.text) is used to convert the JSON response string into a Python dictionary, enabling you to access the participants list and calculate its length to determine the number of participants in the group.

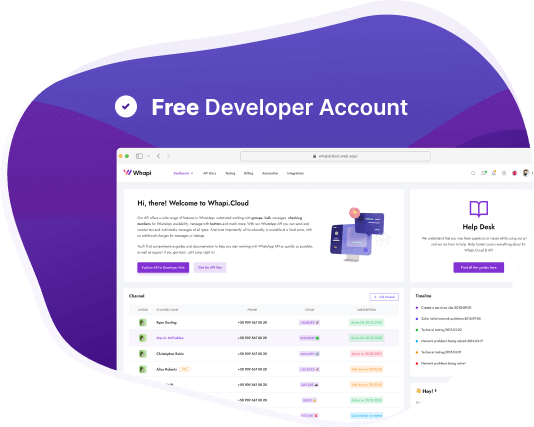
*More methods to automate Groups on Whatsapp can be found here:*[*https://whapi.readme.io/reference/getgroups*](https://whapi.readme.io/reference/getgroups)



Our documentation for group automation

**Testing and Debugging: Local Testing**

When developing a bot, it is essential to continuously test and debug your application to ensure that it is working correctly and is free of errors. For local testing, you can use Postman or any other API testing tool to simulate incoming messages and check the responses from your bot.



[**Try for Free**](https://panel.whapi.cloud/dashboard)

Additionally, you can use Python’s built-in logging module to log important information, errors, and exceptions that can occur during the execution of your bot, which will assist in identifying and resolving issues more efficiently.

Here’s a brief example of implementing logging in your Python application:

import logging  
  
logging.basicConfig(level=logging.DEBUG)  
logger = logging.getLogger(\_\_name\_\_)  
  
try:  
 # Your bot logic here  
 pass  
except Exception as e:  
 logger.exception("An error occurred: %s", e)

**Deployment and Hosting**

Once you have thoroughly tested your bot and are satisfied with its functionality, it is time to deploy it to a server. You can host your bot on any cloud service provider like AWS, Google Cloud, or Azure, or on any server that supports Python.

Consider using services like Docker to containerize your application, making it easier to manage, scale, and deploy. Also, make sure to secure your application, especially if it handles sensitive data, by implementing proper authentication and encryption mechanisms.

**Conclusion and Next Steps**

By now, you should have a fully functional WhatsApp bot that can interact with users, process incoming messages, and send different types of media. The combination of Python and Whapi.Cloud makes it relatively straightforward to build robust and versatile WhatsApp bots, but this is just the beginning.

Continue to build on your bot by adding more advanced features, like natural language processing for more intelligent interactions, or integrating with other APIs to expand the functionality.

[**Try for Free**](https://panel.whapi.cloud/dashboard)

In our [detailed documentation](https://whapi.cloud/docs), you’ll find step-by-step instructions and sample methods that allow you to send a variety of message types, from any file format, locations, and contacts to stickers, audio, polls, and merchandise. In addition, you can dynamically interact with your messages: putting reactions, quoting them, marking them as read, or simulating real-time printing.

Our methods in documentation will automate your business: send products and catalog for leads, process shopping cart, return those who abandoned the cart. Send surveys as feedback collection. Create bots to book seats for events, etc.

*Please replace YOUR\_API\_TOKEN with your real API token from Whapi.Cloud. This code is a basic example and can be augmented with other functions and handlers to create more complex and interactive bots.*