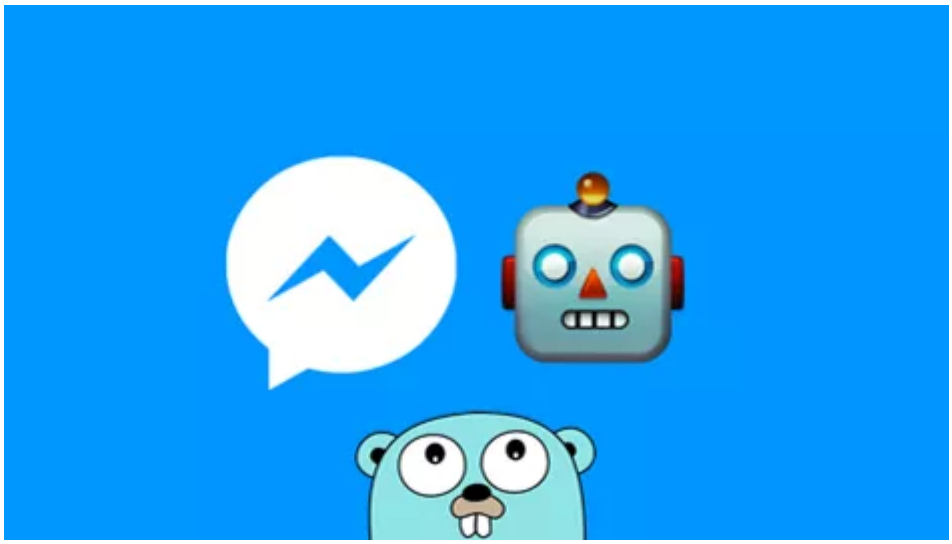
Mohamed Labouardy

Software Engineer/DevOps Engineer, 3x AWS Certified Interested in Serverless, Containers, Go, Distributed Systems & NLP.



Build a Facebook Messenger bot with Go and Messenger API

In this first tutorial of the "**ChatOps**" series , I will show you quickly how to create a Facebook Messenger Bot in Golang. All the code used in this demo can be found on my Github.

1 – Messenger bot

We will start by creating a dummy web server with one endpoint to print a hello world message ✌. I'll use "**gorilla/mux**" package to do that. I found it much easier to setup routes for our server rather than using the **go standard library**.

We first need to install "**gorilla/mux**" library:

```
1 go get github.com/gorilla/mux
```

Then, create a file called **app.go** with the following content:

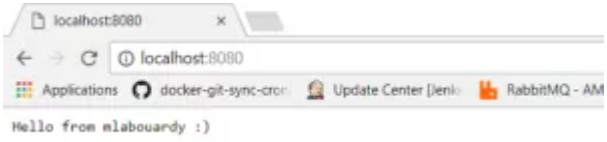
```go
package main

import (
    "fmt"
    "log"
    "net/http"

    "github.com/gorilla/mux"
)

func HomeEndpoint(w http.ResponseWriter, r *http.Request) {
    fmt.Fprintln(w, "Hello from mlabouardy :)")
}

func main() {
    r := mux.NewRouter()
    r.HandleFunc("/", HomeEndpoint)
    if err := http.ListenAndServe(":8080", r); err != nil {
        log.Fatal(err)
    }
}
```

So basically, we have a ***main*** function which create a new route "/", and associate to this path a ***HomeEndpoint*** function that use **ResponseWrite** reference to print a message. Then it start a **HTTP server** on port **8080**.

To verify that things are working, start your local server with:

```
go run app.go
```

Then in your browser, visit **http://localhost:8080** and you should see:



Now that you've got your first endpoint working, we will need to expose 2 endpoints to make it work with Facebook platform:

1.1 – Verification Endpoint: (GET /webhook)

```
func VerificationEndpoint(w http.ResponseWriter, r *http.Request) {
    challenge := r.URL.Query().Get("hub.challenge")
    token := r.URL.Query().Get("hub.verify_token")

    if token == os.Getenv("VERIFY_TOKEN") {
        w.WriteHeader(200)
        w.Write([]byte(challenge))
    } else {
        w.WriteHeader(404)
        w.Write([]byte("Error, wrong validation token"))
    }
}
```

It simply looks for the **Verify Token** and responds with the **challenge** sent in the verification request.

1.2 – Messages Handler Endpoint: (POST /webhook)

```
func MessagesEndpoint(w http.ResponseWriter, r *http.Request) {
    var callback Callback
    json.NewDecoder(r.Body).Decode(&amp;callback)
    if callback.Object == "page" {
        for _, entry := range callback.Entry {
            for _, event := range entry.Messaging {
                ProcessMessage(event)
            }
        }
        w.WriteHeader(200)
        w.Write([]byte("Got your message"))
    } else {
        w.WriteHeader(404)
        w.Write([]byte("Message not supported"))
    }
}
```

It serialize the request body into **Callback object** , then it parse it and fetch the message object and pass it as an argument to ***ProcessMessage*** function that will use **Facebook Graph API** to send the response to the user (in this case we will send an image):

```
func ProcessMessage(event Messaging) {
    client := &amp;http.Client{}
    response := Response{
        Recipient: User{
            ID: event.Sender.ID,
        },
        Message: Message{
            Attachment: &amp;Attachment{
                Type: "image",
                Payload: Payload{
                    URL: IMAGE,
                },
            },
        },
    }
    body := new(bytes.Buffer)
    json.NewEncoder(body).Encode(&amp;response)
    url := fmt.Sprintf(FACEBOOK_API, os.Getenv("PAGE_ACCESS_TOKEN"))
    req, err := http.NewRequest("POST", url, body)
    req.Header.Add("Content-Type", "application/json")
    if err != nil {
        log.Fatal(err)
    }

    resp, err := client.Do(req)
    if err != nil {
        log.Fatal(err)
    }
    defer resp.Body.Close()
}
```

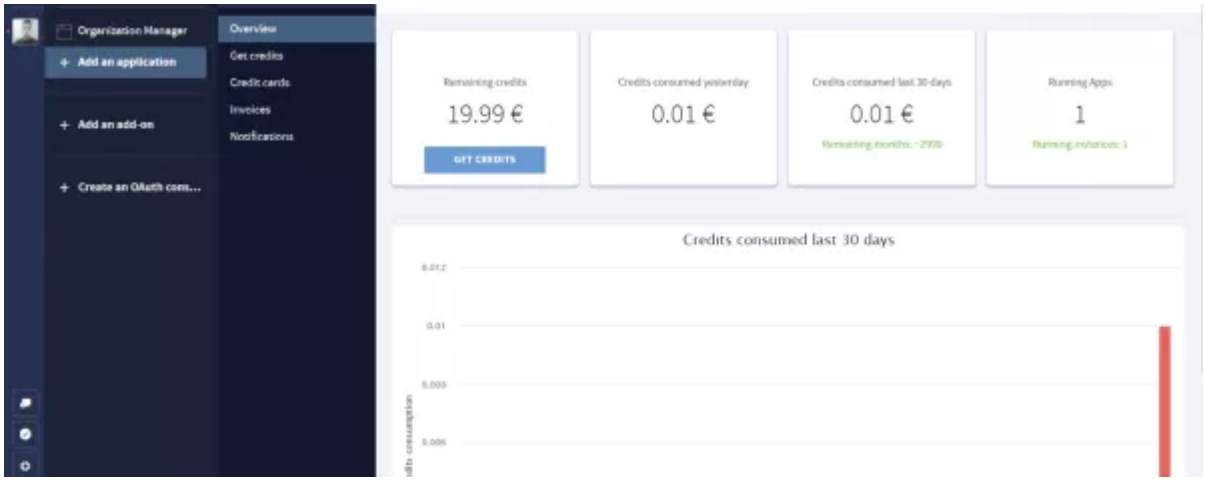
Our local server url (**http://localhost:8080**) is not available to all the other people in the internet and doesn't support **HTTPS** which is necessary for Facebook Messenger bot. Therefore, we need to expose it to the public.

2 – Deployment

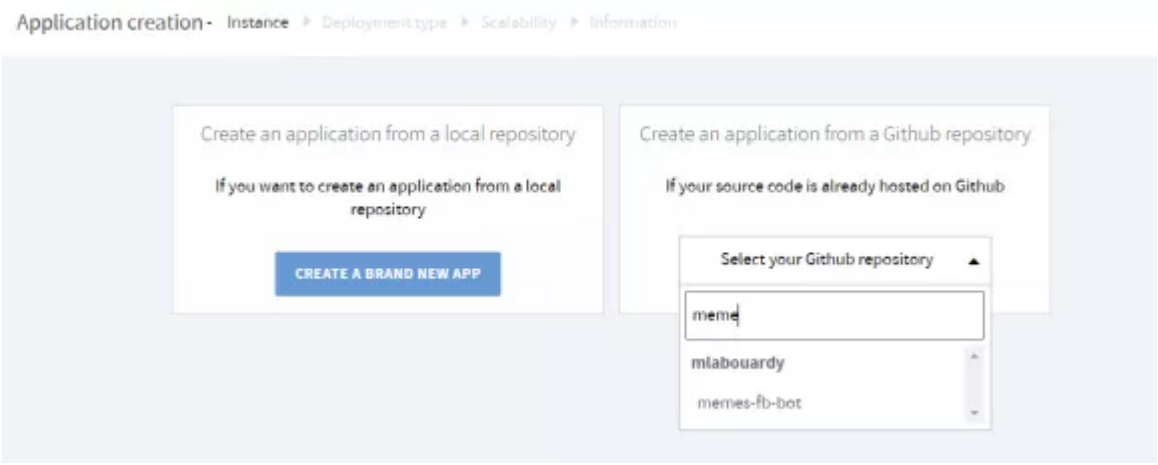
Note: You could also use a tool like **ngrok**. It basically creates a secure tunnel on your local machine along with a public URL you can use for browsing your local server. Keep in mind, to use your bot in production, you need to use a real **IaaS** like **AWS**, **Heroku**, **Clever Cloud** ...etc

In this tutorial I will choose **CleverCloud** as IaaS provider, it deploys your Go application for free and offer you extra add ons like monitoring, logs, scaling, continuous delivery ...

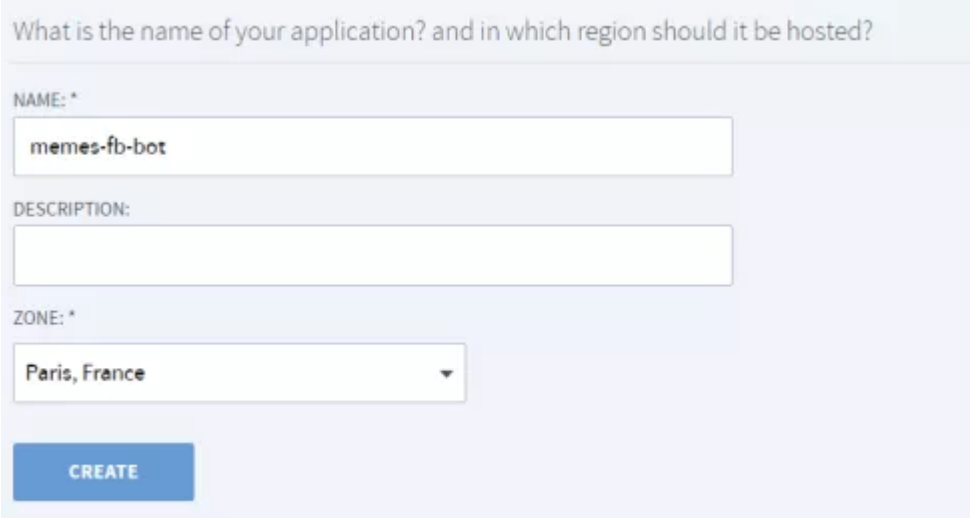
In order to deploy to **CleverCloud** you'll need a **CleverCloud** user account. **Signup is free and instant**. After signing up:



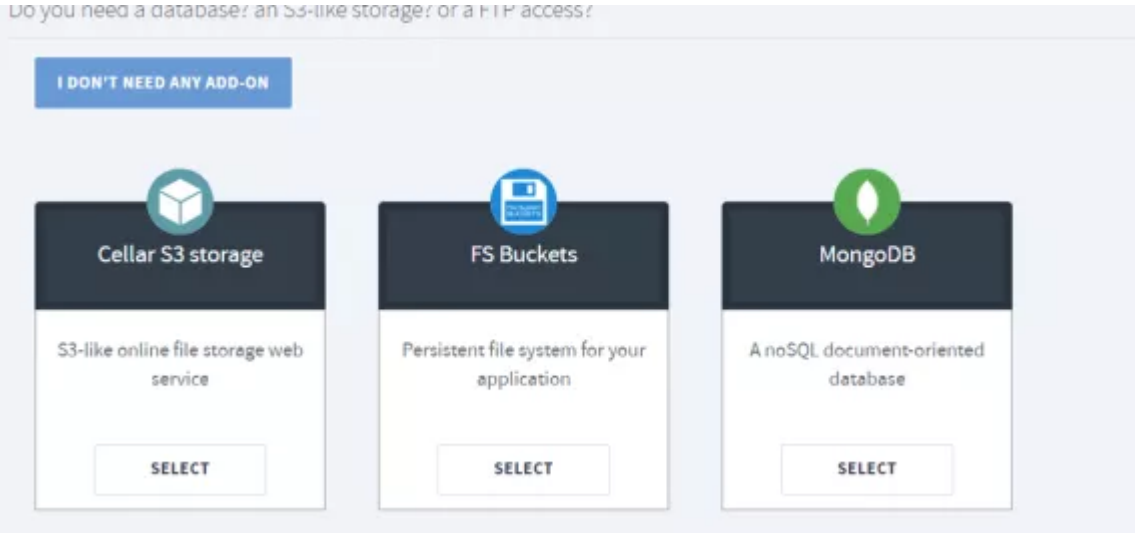
We click on "**Add an application**", then you can either upload your app code from local repository or from **Github**:



Next, We choose **GO** as our server side language. Then, we click on "**Next**"

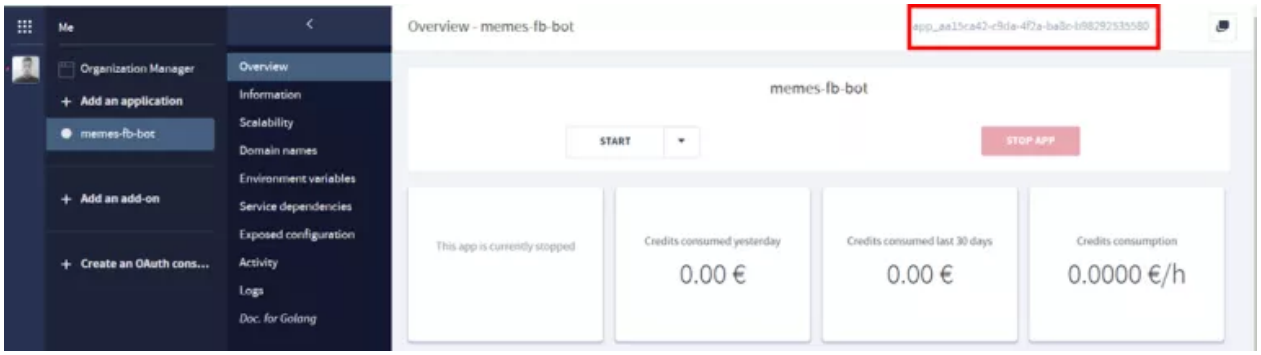


We left all fields as default, and we click on "**Create**"



Our server does not use any external resources (MySQL, Redis ...) so we will simply skip this part by clicking on "**I don't need any add-on**"

Congratulations ! You have successfully deployed your server.



Note: The app URL is :

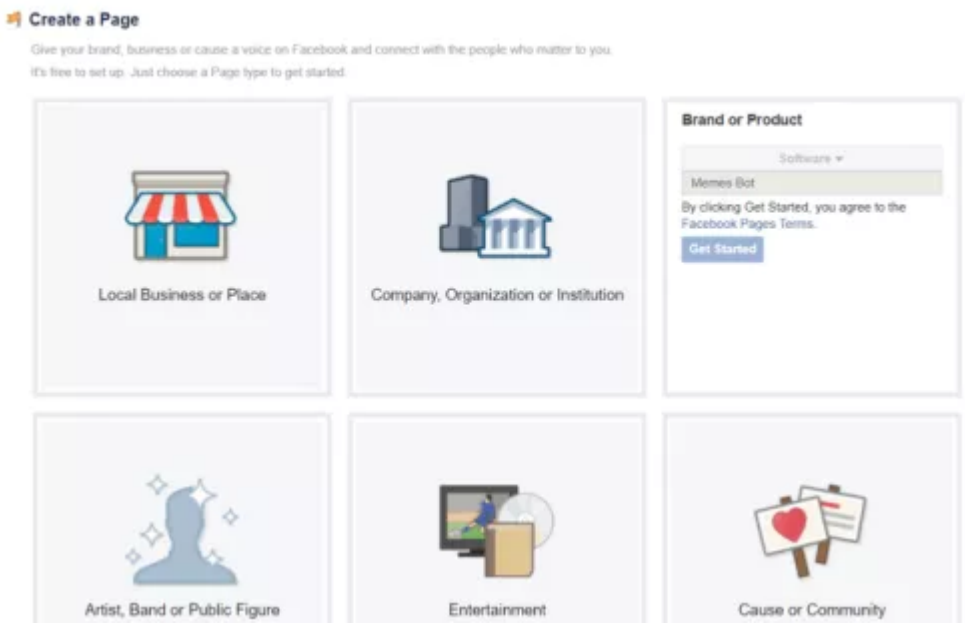
https://ID.cleverapps.io

ID is the string on the bottom right of the dashboard,

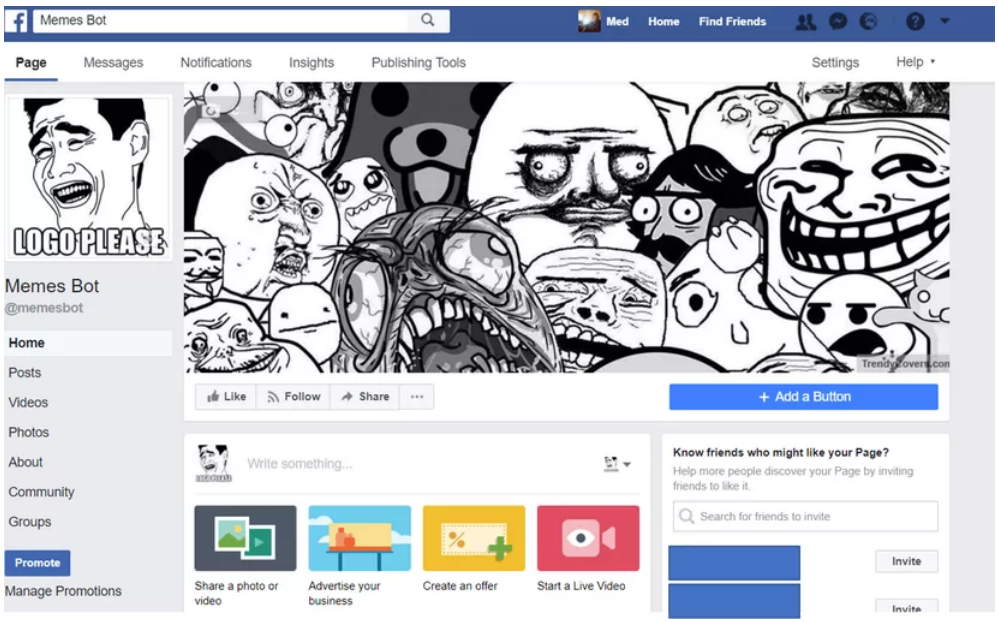
3 – Facebook Setup

3.1 – Create Facebook Page

If you don't already have one, you need to **create a facebook page** that we will use to connect our bot to.



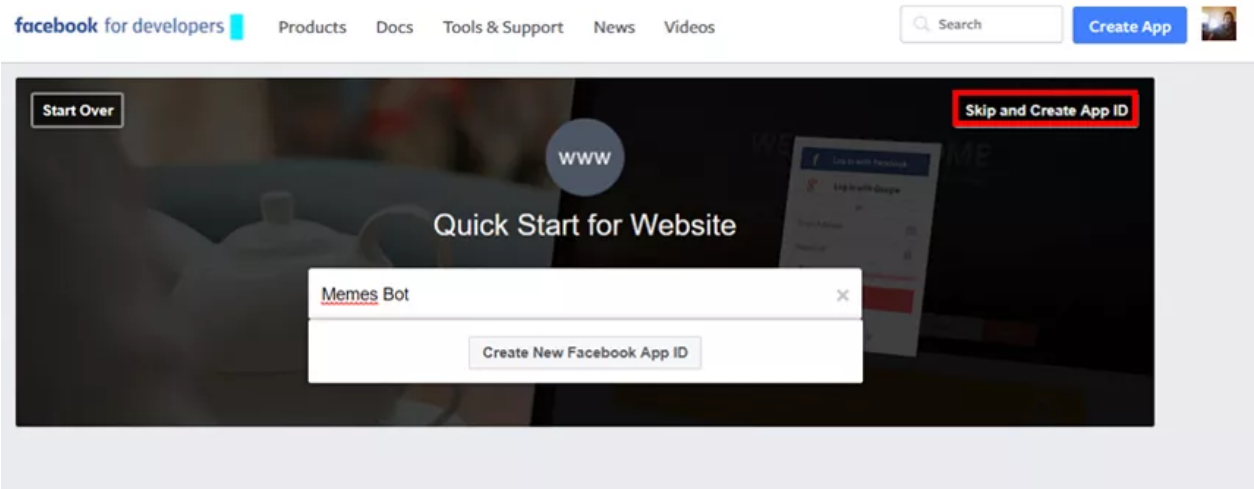
Just give it a name, and that's it now you have created a **Facebook** page for you **Bot**



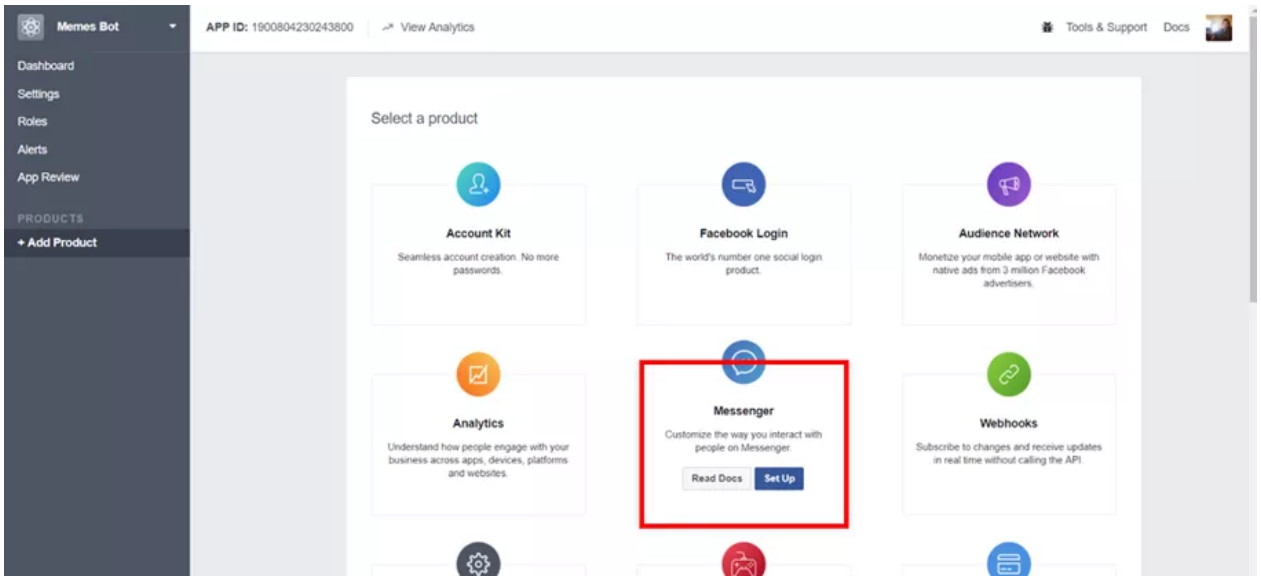
3.2 – Create Facebook App

Once the page was created, we will **create a facebook app** which will be connected your webhook server and your public page: which works as middleware that connects your Webhook (APP URL) and your public page.

You need to give your app a name then click on "**Skip and Create App ID**"



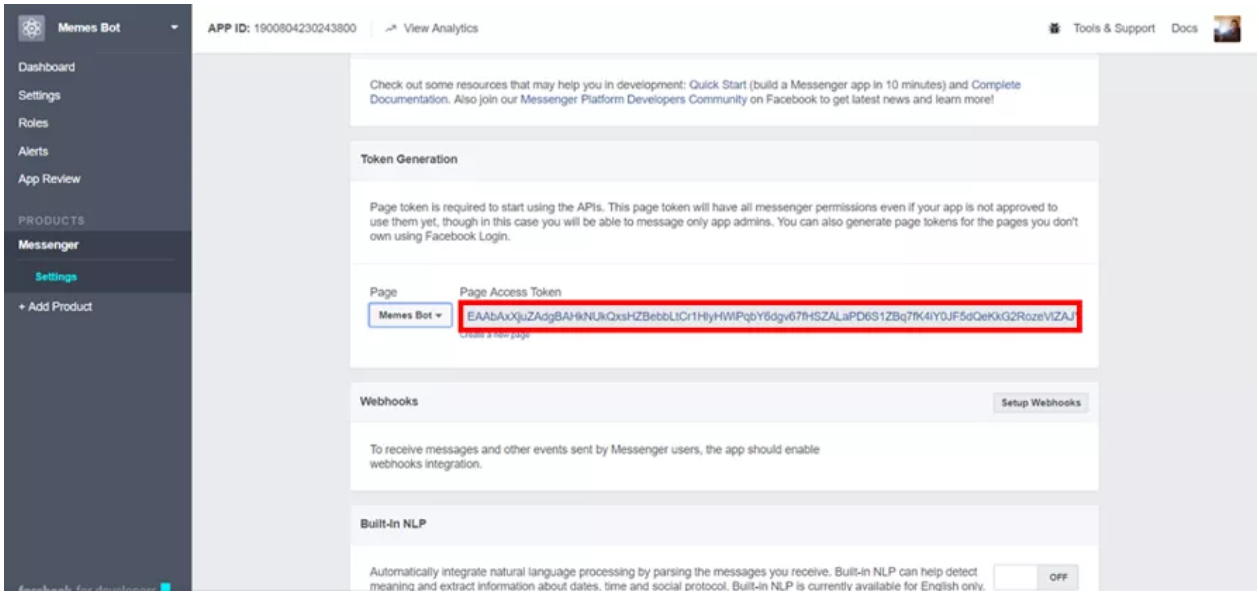
After creating an app, we need to add **Messenger** as a platform. So we click on "**Add Product**" then select **Messenger** as a product



Now you're in the **Messenger** settings. There are few things in here you'll need to fill out in order to get your chatbot wired up to the server endpoint we set up earlier.

3.2.1 – Generate a Page Access Token

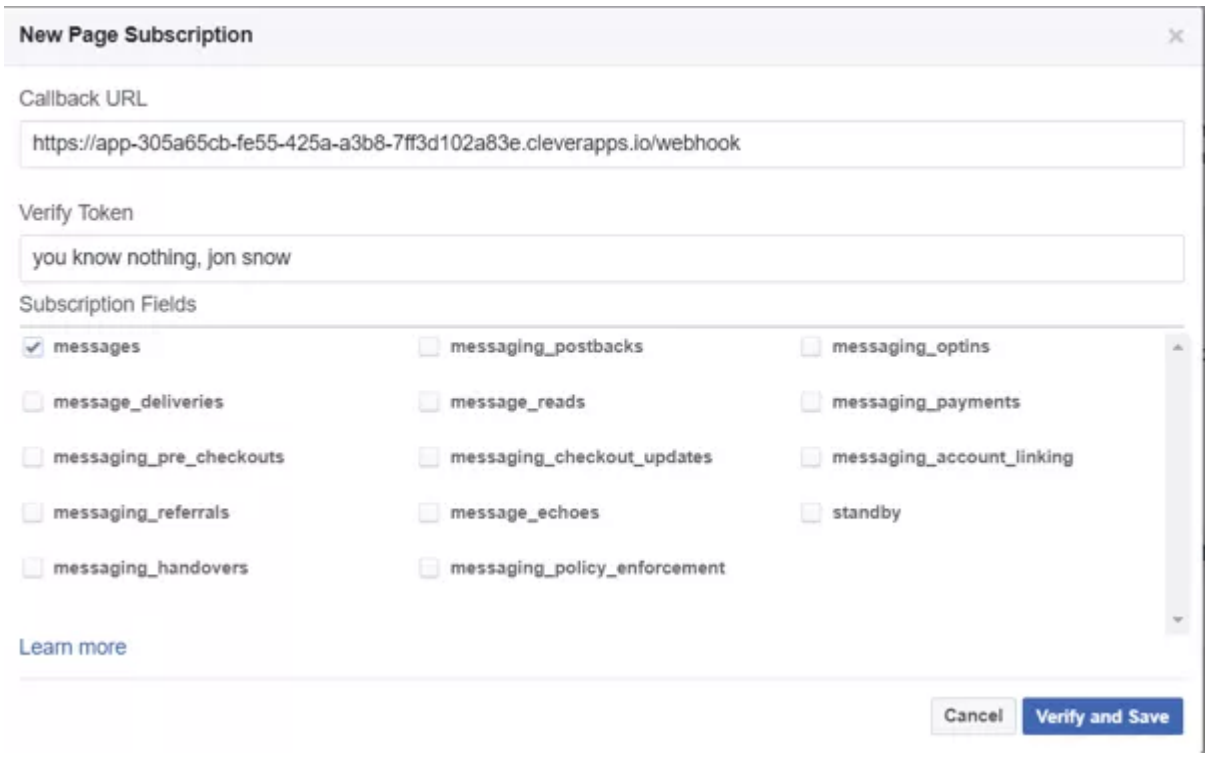
Using the page we created earlier, you'll get a random "**Page Access Token**"



You need to copy the token to your clipboard. We'll need it as an environment variable (**PAGE_ACCESS_TOKEN**) for our server.

3.2.2 – Setup subscription

Next, we click on "**Setup webhook**" button on "**Webhooks**" section, a new popup will show up:

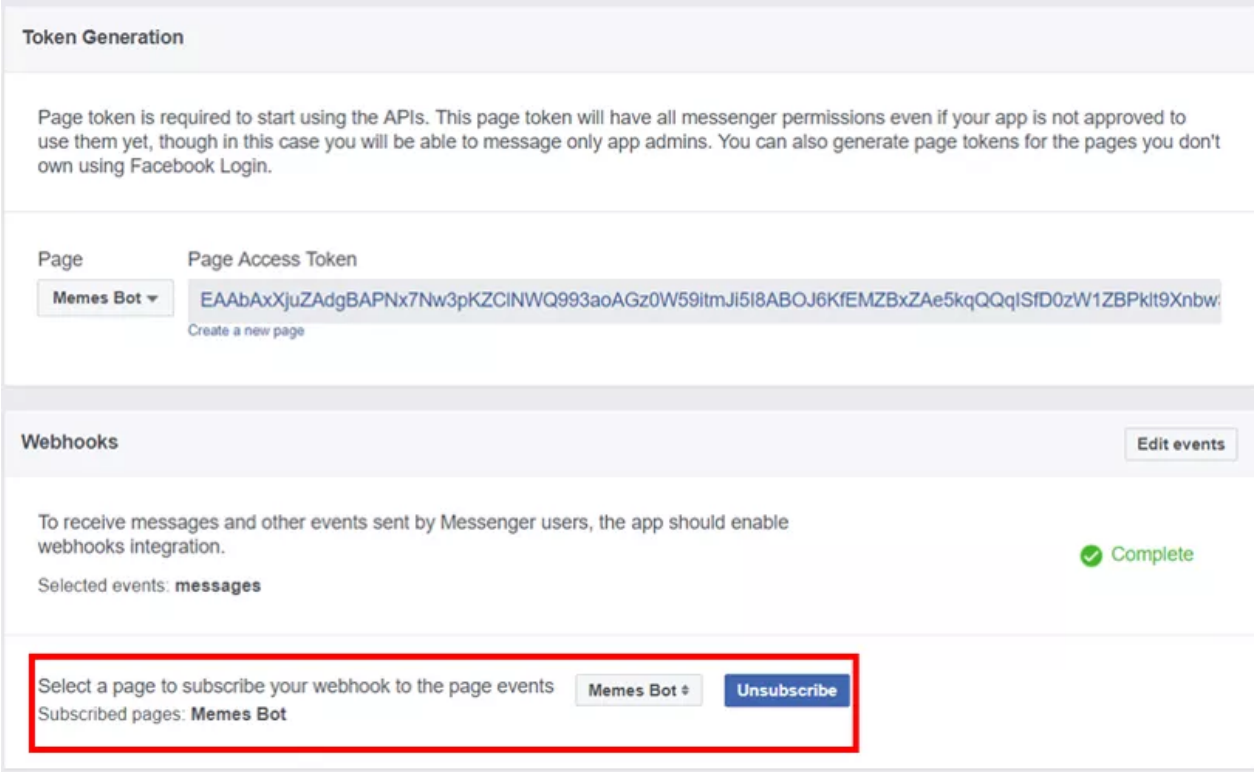


- **Callback URL** : Clever Cloud we set up earlier
- **Verify Token**: A secret token that will be sent to your bot, in order to verify the request is coming from Facebook, Make sure to remember the value because we will

need it as a second environment variable (**VERIFY_TOKEN**) for the server

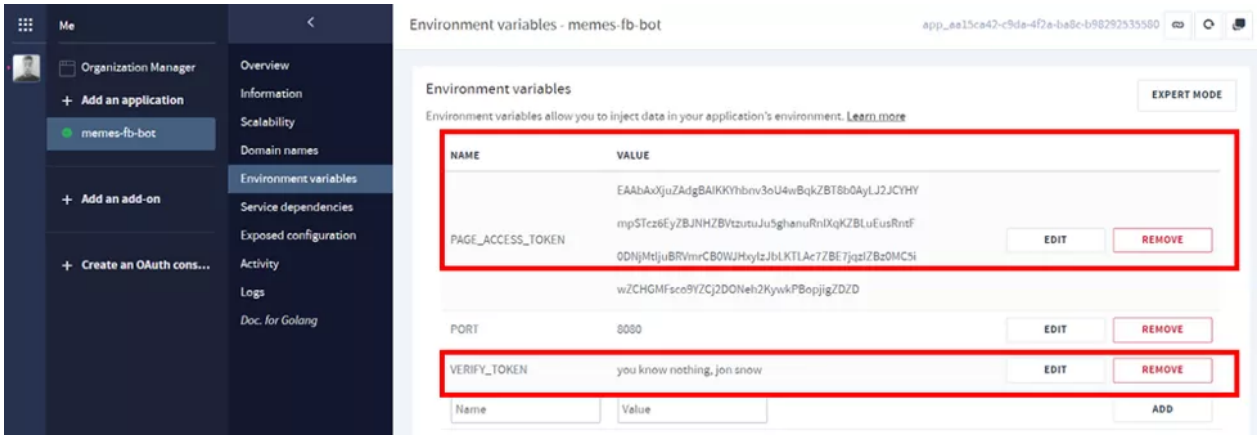
- **Subscription Fields:** represents which events you want Facebook to notify your webhook about, in this case, we will choose "**messages**"

After you've configure your subscription, you'll need to subscribe to the specific page, you want to receive message notification for



3.2.3 – Set environment variables

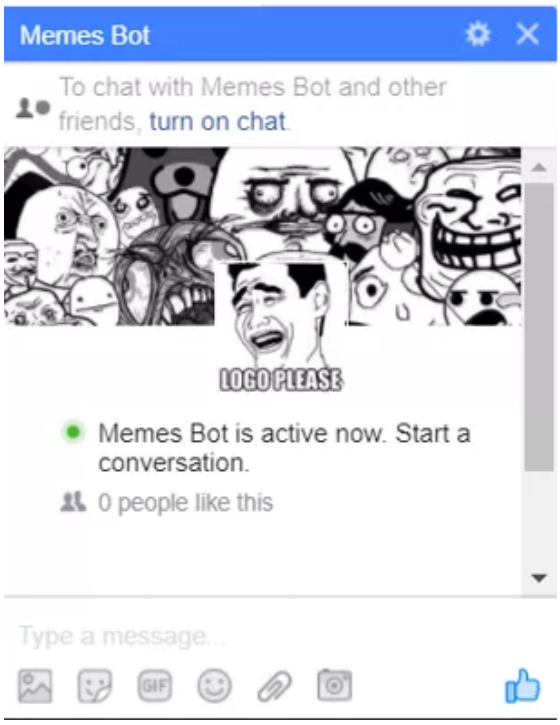
Once you've gotten your **PAGE_ACCESS_TOKEN** and **VERIFY_TOKEN**, make sure you add those two as environment variables for the server on **clever cloud dashboard**

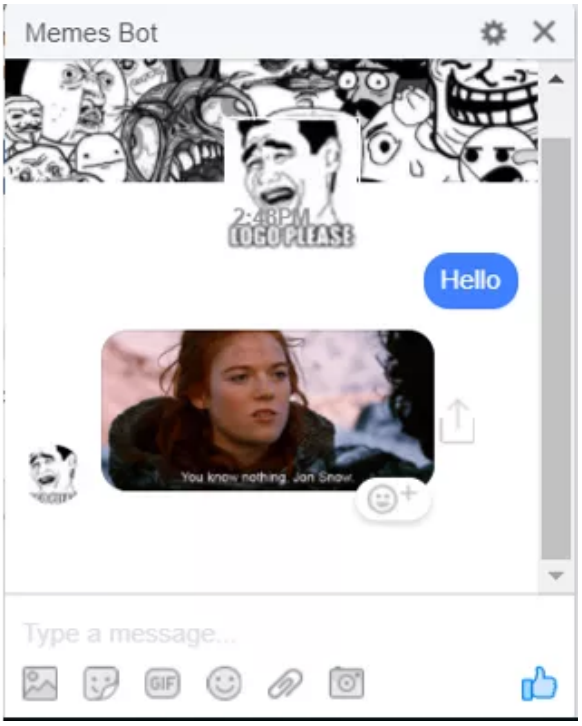


Then restart the application and you should be good to go !

4 – Test the Bot

Go to your Facebook Page and send a message to it. You should see a gif back to you



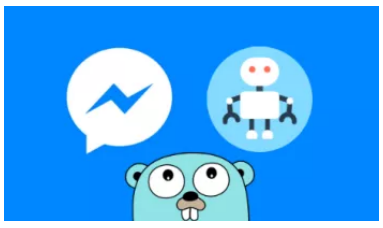


5 – Customize your Bot's behavior

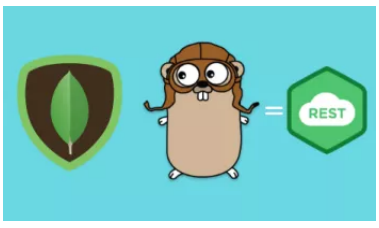
In this quick tutorial I showed you how to build a simple & dumb bot for facebook messenger, to make it smarter and have more interactions with the user. We need to use a **NLP** backend like **api.ai** (Google), **wit.ai** (Facebook) or **motion.ai**. And that will be the subject of my upcoming tutorial ❤

SHARE THIS:

RELATED



Messenger Bot with DialogFlow & Golang
In "Bot"



Build RESTful API in Go and MongoDB
In "AWS"

Create 9Gag Android Application
In "Android"

Comments

2 comments

2 Comments Sort by **Oldest**

Add a comment...

Rania Zyane

Many techies in one tutorial. In addition, this can be used as a getting started bot application with Golang since there are very few chatbots written in this lang.
Thanks !

Like · Reply · 2 · 1y

Michelle Chartrand

There is github.com/abourget/slick

Like · Reply · 1 · 1y

Minh Hiếu

I did follow instructions but it did not work at step setup webhook.
It informed " This url contains an invalid domain " (https://app_c088f42b-6cf5-4c83-ac2c-6755fd62d172.cleverap...)
And i change " app_ " to " app- " , it throws http code 404 not found.
Did i have to start it in Overview ? (I tried but it failed to deploy)

Like · Reply · 1 · 43w

Facebook Comments Plugin

August 8, 2017 Mohamed Labouardy Bot, ChatOps, Cloud, Go

Designed by Mohamed Labouardy