

Manual para instalar contenedores

¿Que es docker?

La idea detrás de Docker es crear contenedores ligeros y portables para las aplicaciones software que puedan ejecutarse en cualquier máquina con Docker instalado, independientemente del sistema operativo que la máquina tenga por debajo.

Para más información, ingresa a los siguientes links:

<https://www.javiergarzas.com/2015/07/que-es-docker-sencillo.html>

<https://www.redhat.com/es/topics/containers/what-is-docker>

¿Que es un contenedor?

Los contenedores son un modo estándar de empaquetar el código, las configuraciones y las dependencias de un proyecto en un único objeto.

Si los vemos de otra manera, un contenedor es un entorno en el cual instalas todos los programas y ficheros que son necesarios para un proyecto, de este proyecto puedes crear diferentes versiones, restaurarlo en caso de ser necesario, además de compartirlo e implementarlo fácilmente en otras computadoras independientemente del sistema operativo con el que se esté trabajando, basta con tener docker instalado.

Propósito de este manual

Instalar contenedor para realizar pruebas desde dialogflow usando una base de datos.

Instalación del contenedor

1. Instalar docker, esta información cambia dependiendo del sistema operativo que se usa, basate en la documentación oficial de docker o sigue algún videotutorial, es necesario que mediante la terminal se puedan correr los comandos docker, para estar seguro, corre el siguiente comando en la terminal.

docker

Debería de aparecer una lista de opciones de todo lo que se puede hacer con dicho comando, como se muestra a continuación:

```
Archivo  Editar  Ver  Búsqueda  Terminal  Ayuda
jazz@jazz-XPS-8700:~$ docker

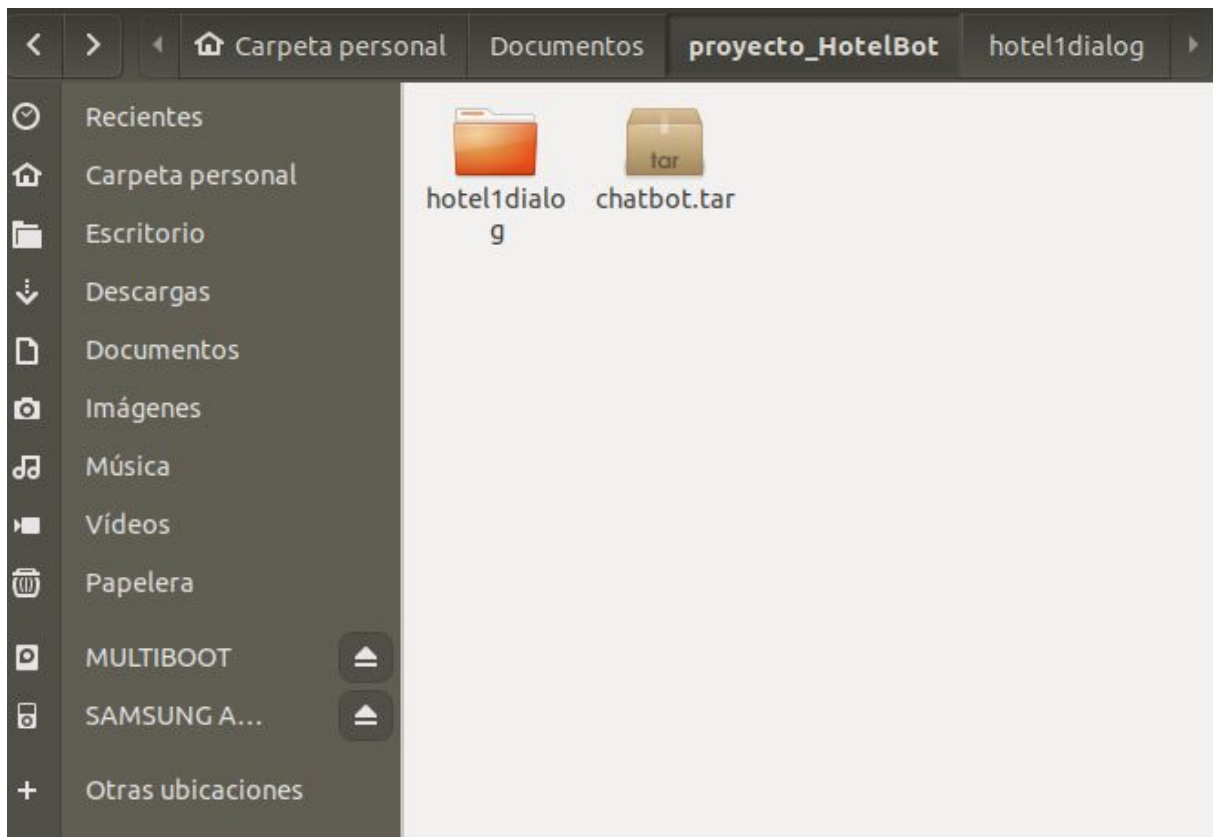
Usage: docker [OPTIONS] COMMAND

A self-sufficient runtime for containers

Options:
  --config string      Location of client config files (default
                        "/home/jazz/.docker")
  -c, --context string  Name of the context to use to connect to the
                        daemon (overrides DOCKER_HOST env var and
                        default context set with "docker context use")
  -D, --debug           Enable debug mode
  -H, --host list       Daemon socket(s) to connect to
  -l, --log-level string Set the logging level
                        ("debug"|"info"|"warn"|"error"|"fatal")
                        (default "info")
  --tls                Use TLS; implied by --tlsverify
  --tlscacert string    Trust certs signed only by this CA (default
                        "/home/jazz/.docker/ca.pem")
  --tlscert string      Path to TLS certificate file (default
                        "/home/jazz/.docker/cert.pem")
  --tlskey string       Path to TLS key file (default
                        "/home/jazz/.docker/key.pem")
  --tlsverify           Use TLS and verify the remote
  -v, --version         Print version information and quit

Management Commands:
  builder      Manage builds
  config       Manage Docker configs
  container    Manage containers
  context      Manage contexts
  engine       Manage the docker engine
  image        Manage images
  network      Manage networks
  node         Manage Swarm nodes
  plugin       Manage plugins
  secret       Manage Docker secrets
  service      Manage services
  stack        Manage Docker stacks
  swarm       Manage Swarm
  system       Manage Docker
  trust        Manage trust on Docker images
  volume       Manage volumes
```

2. Coloca la carpeta llamada "proyecto_HotelBot" en documentos.



3. El siguiente paso es cargar la imagen (el archivo llamado chatbot.tar) de nuestro contenedor, para ello usamos el siguiente comando.

`docker load -i ruta_donde_esta_chatbot.tar`

```
jazze@jazze-HP-455-Notebook-PC:~$ docker load -i /home/jazze/Documentos/proyecto_HotelBot/chatbot.tar
fbb641a8b943: Loading layer 105.5MB/105.5MB
ed8673d4b40c: Loading layer 265MB/265MB
d2d8922feed1: Loading layer 373.1MB/373.1MB
Loaded image: chatbot:v1
```

4. Para asegurarnos de que tenemos esta imagen cargada, corremos el comando

docker images

Nos mostrará una lista de imágenes que tenemos cargadas en nuestra máquina como en la siguiente imagen.

```
jazz@jazz-HP-455-Notebook-PC:~$ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
chatbot	v1	e3f24cb4647f	5 days ago	717MB
rails	6_v2	7c3b6de37ec0	2 weeks ago	2.46GB
rails_c	v1	05a31e53b1c7	2 months ago	1.28GB
cicese/suicidio_postgres	latest	fd80abc3ef1a	10 months ago	430MB
hello-world	latest	fce289e99eb9	12 months ago	1.84kB

- Después crearemos el contenedor de esa imagen, para ello es necesario correr el siguiente comando

```
docker run -itp 8080:8080 --name chatbot -v
/ruta_a_la_carpeta_hotel1dialog:/nombre_de_ruta_de_contenedor chatbot:v1
/bin/bash
```

***nombre_de_ruta_de_contenedor** : Es una ruta que se crea dentro del contenedor, se recomienda que uses **/nombre_cualquiera**, en el caso del ejemplo, la carpeta se llama **/jazz**.

Cuando el comando se ejecute nos aparecerá lo siguiente:

```
jazz@jazz-HP-455-Notebook-PC:~$ docker run -itp 8080:8080 --name chatbot -v /home/jazz/Documentos/proyecto_HotelBot/museoidialog /jazz chatbot:v1 /bin/bash
root@b2dab2730ddb:/#
```

- Lo que aparece en la siguiente imagen nos indica que nuestro contenedor está creado y estamos dentro de él.

```
root@b2dab2730ddb:/#
```

Comandos docker

Antes de seguir, se mostrarán los comandos necesarios para poder utilizar el contenedor:

- Ver los comandos que docker ofrece

docker

```
jazz@jazz-XPS-8700:~$ docker

Usage:  docker [OPTIONS] COMMAND

A self-sufficient runtime for containers

Options:
  --config string      Location of client config files
  -c, --context string  Name of the context to use to connect to the daemon
                       and its APIs; this override any Docker configuration
                       on the host
  -D, --debug           Enable debug mode
  -H, --host list       Daemon socket(s) to connect to
  -l, --log-level string Set the logging level: "debug", "info", "warn",
  --tls                Use TLS; implied by --tlsverify
  --tlscacert string   Trust certs signed under the given path or by all
  --tlscert string     Path to TLS certificate file
  --tlskey string       Path to TLS key file
  --tlsverify           Use TLS and verify the remote
  -v, --version         Print version information and quit

Management Commands:
  builder      Manage builds
  config       Manage Docker configs
  container    Manage containers
  context       Manage contexts
  engine       Manage the docker engine
  image        Manage images
  network      Manage networks
  node         Manage Swarm nodes
  plugin       Manage plugins
  secret       Manage Docker secrets
  service      Manage services
  stack        Manage Docker stacks
  swarm        Manage Swarm
  system       Manage Docker
  trust        Manage trust on Docker images
  volume       Manage volumes
```

- Ver documentación de cada comando

docker comando --help


```
jazz@jazz-XPS-8700:~$ docker config --help

Usage:  docker config COMMAND

Manage Docker configs

Commands:
  create      Create a config from a file or STDIN
  inspect     Display detailed information on one or more configs
  ls          List configs
  rm          Remove one or more configs

Run 'docker config COMMAND --help' for more information on a command.
```

- ver contenedores que están corriendo

docker ps

```
jazz@jazz-XPS-8700:~$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED
NAMES			

*En este caso o nos muestra ningún contenedor por que no esta corriendo ninguno

- muestra todos los contenedores estén corriendo o no

docker ps -a

```
jazz@jazz-XPS-8700:~$ docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
5b1c8a215674	conda/miniconda3	"/bin/bash"	6 days ago	Exited (0) 20 hours ago	
30cc81aa2307	cicese/suicidio_postgres	"/bin/bash"	5 weeks ago	Exited (0) 5 weeks ago	
427fb8d1d375	rails:6	"/bin/bash"	5 weeks ago	Exited (0) 7 days ago	

- Iniciar un contenedor

docker start nombre_contenedor

```
jazz@jazz-XPS-8700:~$ docker start chatbot
chatbot
```

*si no recuerdas el nombre del contenedor, ejecuta **docker ps -a** y en la información que te brinda este comando se encuentra el nombre.

```
NAMES
chatbot
db
dorian
```

- Comprobar con docker ps

```
jazz@jazz-XPS-8700:~$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
5b1c8a215674	conda/miniconda3	"/bin/bash"	6 days ago	Up 7 minutes

*podemos ver que está corriendo, ahora

- Entrar al contenedor

docker attach nombre_contenedor

```
jazz@jazz-XPS-8700:~$ docker attach chatbot
root@5b1c8a215674:/#
```

- Salir del contenedor

Hay dos formas de salir del contenedor:

- **Salir y cerrar el contenedor**

se escribe **exit**

```
root@5b1c8a215674:/# exit
exit
```

si revisamos los contenedores que están corriendo

```
jazz@jazz-XPS-8700:~$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED
5b1c8a215674	conda/miniconda3	"/bin/bash"	6 days ago

- **Salir sin cerrar el contenedor**

combinando las teclas ctrl + p + q

```
jazz@jazz-XPS-8700:~$ docker attach chatbot
root@5b1c8a215674:/# read escape sequence
```

si revisamos los contenedores que están corriendo

```
jazz@jazz-XPS-8700:~$ docker ps
```

CONTAINER ID	IMAGE	COMMAND
5b1c8a215674	conda/miniconda3	"/bin/bash"

Dirígete a la carpeta que creamos dentro de nuestro contenedor, en mi caso está en la ruta /jazz, si no recuerdas el nombre, puedes usar el comando

ll

para que te liste los ficheros de tu contenedor

```
root@5b1c8a215674:/# ll
total 96
drwxr-xr-x  1 root root 4096 Jan 17 19:04 .
drwxr-xr-x  1 root root 4096 Jan 17 19:04 ..
-rwxr-xr-x  1 root root    0 Jan 17 19:04 .dockerenv
drwxr-xr-x  2 root root 4096 Apr 12  2019 .empty
drwxr-xr-x  1 root root 4096 Jan 17 19:08 bin
drwxr-xr-x  2 root root 4096 Feb  3  2019 boot
drwxr-xr-x  5 root root  360 Jan 23 19:47 dev
drwxr-xr-x  1 root root 4096 Jan 17 19:08 etc
drwxr-xr-x  2 root root 4096 Feb  3  2019 home
drwxrwxrwx  4 1000 1000 4096 Jan 22 23:14 jazz
drwxr-xr-x  1 root root 4096 Mar 26  2019 lib
drwxr-xr-x  2 root root 4096 Mar 26  2019 lib64
drwxr-xr-x  2 root root 4096 Mar 26  2019 media
drwxr-xr-x  2 root root 4096 Mar 26  2019 mnt
drwxr-xr-x  2 root root 4096 Mar 26  2019 opt
dr-xr-xr-x 346 root root    0 Jan 23 19:47 proc
drwx----- 1 root root 4096 Jan 17 22:02 root
```

para acceder al directorio /jazz se utiliza el siguiente comando

cd jazz

```
root@5b1c8a215674:/# cd jazz
root@5b1c8a215674:/jazz#
```

si usamos de nuevo el comando **ll** podemos ver los archivos que estaban en nuestra carpeta hotel1dialog.


```

^Croot@5b1c8a215674:/jazz# ll
total 61280
drwxrwxrwx 4 1000 1000      4096 Jan 22 23:14 .
drwxr-xr-x 1 root root      4096 Jan 17 19:04 ..
-rwxrwxrwx 1 1000 1000       162 Jan 17 17:37 ~/.nual_museo_instalacion1.docx
-rwxrwxrwx 1 1000 1000       2331 Jul  9 2019 Bot-Demo-f535b0c412fa.json
-rwxrwxrwx 1 1000 1000    1434973 Jul  9 2019 FORMATO_DE_ASIGNACION.pdf
-rwxrwxrwx 1 1000 1000   137907 Jul  9 2019 Registro-erika-jazmin-urciel-hernandez.pdf
-rwxrwxrwx 1 1000 1000        97 Jul  9 2019 _env
-rwxrwxrwx 1 1000 1000        41 Jul  9 2019 _flaskenv
-rwxrwxrwx 1 1000 1000     22331 Jul  9 2019 arbol_conversacion.docx
-rwxrwxrwx 1 1000 1000     26624 Jan 20 21:58 bd_hotel.s3db
-rwxrwxrwx 1 1000 1000       2884 Jul  9 2019 expresiones_regulares.py
-rwxrwxrwx 1 1000 1000        672 Jul  9 2019 index.py
-rwxrwxrwx 1 1000 1000   653481 Jul  4 2019 manual_museo_bot.docx
-rwxrwxrwx 1 1000 1000  1001123 Jul 11 2019 manual_museo_documentacion2.docx
-rwxrwxrwx 1 1000 1000  2222758 Jul 11 2019 manual_museo_instalacion1.docx
-rwxr-xr-x 1 1000 1000 26683198 Oct  8 19:55 ngrok
-rwxrwxrwx 1 1000 1000  30477824 Jul  9 2019 ngrok.exe
-rw-r--r-- 1 root root        643 Jan 17 18:57 requirements.txt
drwxrwxrwx 2 1000 1000      4096 Jul  9 2019 static
drwxrwxrwx 2 1000 1000      4096 Jul  9 2019 templates
-rwxrwxrwx 1 1000 1000     11449 Jan 17 20:09 webhook.py
-rw-rw-r-- 1 1000 1000       2914 Jan 24 19:08 webhook_new.py
-rwxrwxrwx 1 1000 1000      8727 Jul  9 2019 webhooks1.py

```

Por último es necesario habilitar los permisos de ejecución de un programa llamado ngrok(servidor), para esto tenemos que correr el siguiente comando.

```

root@5b1c8a215674:/jazz# chmod +x ngrok

```

Trabajando con el contenedor y DialogFlow

1. Abrir diálogo flow y entrar a cualquier intent

En este caso abrimos el intent “habitaciones”, en el apartado “Action and parameters”, agregamos un nombre a la acción “action1.habitacion”, esta configuración es necesaria para poder usar los intentos en nuestro webhook, mediante código, esto se verá más a profundidad más adelante.

Action and parameters

action1.habitacion

REQUIRED ?	PARAMETER NAME ?	ENTITY ?	VALUE	IS LIST ?
<input type="checkbox"/>	Enter name	Enter entity	Enter value	<input type="checkbox"/>

+ New parameter

En el apartado “Responses” no agregamos ninguna respuesta y en el apartado “Fulfillment” activamos la opción “Enable webhook call for this intent”

Fulfillment ?

☒ Enable webhook call for this intent

☐ Enable webhook call for slot filling

Por último guardamos los cambios realizados a nuestro intent.

• habitaciones

SAVE

2. Abrir terminal, iniciar y entrar al contenedor, una vez dentro accedemos al directorio /jazz.

```
root@5b1c8a215674:/jazz#
```

Correr el siguiente comando

python webhook_new.py &

Este comando correrá el webhook en segundo plano, te debe de aparecer lo siguiente:

```
root@5b1c8a215674:/jazz# python webhook_new.py &
[1] 6
root@5b1c8a215674:/jazz# * Serving Flask app "webhook_new" (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://0.0.0.0:8080/ (Press CTRL+C to quit)
```

Después correremos el siguiente comando

./ngrok http puerto_de_flask

puerto_de_flask: Es el puerto que se obtiene al correr el servicio de flask, lo puedes observar en el recuadro rosa de la imagen anterior, por lo tanto el comando queda así:

```
root@5b1c8a215674:/jazz# ./ngrok http 8080
```

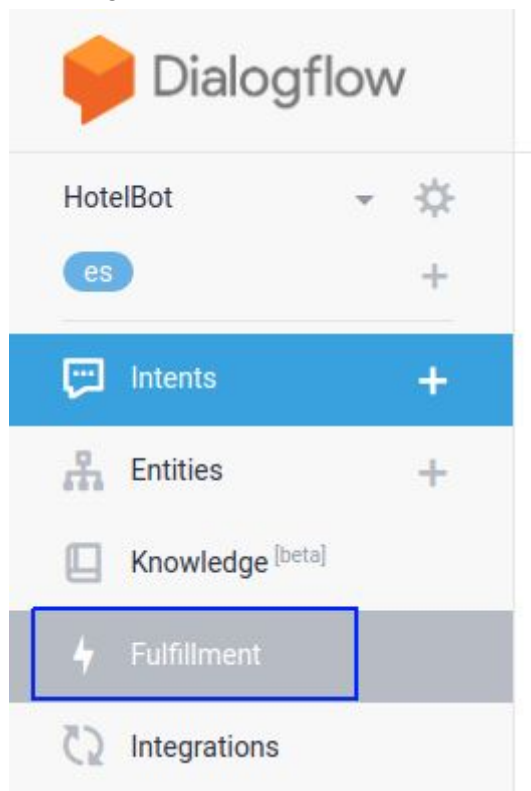
copiamos la dirección que está encerrada en el recuadro azul

```
ngrok by @inconshreveable

Session Status      online
Account             19uhej@gmail.com (Plan: Free)
Version             2.3.35
Region              United States (us)
Web Interface       http://127.0.0.1:4040
Forwarding           http://1d162e5d.ngrok.io -> http://localhost:8080
Forwarding           https://1d162e5d.ngrok.io -> http://localhost:8080

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

Nos dirigimos al apartado fulfillment de dialogflow





Pega la dirección que obtuvimos de ngrok en el recuadro rojo agregando /webhook como se muestra a continuación:

Webhook

ENABLED 

Your web service will receive a POST request from Dialogflow in the form of the response to a user query matched by intents with webhook enabled. Be sure that your web service meets all the [webhook requirements](#) specific to the API version enabled in this agent.

URL*	<input type="text" value="https://1d162e5d.ngrok.io/webhook"/>	
BASIC AUTH	<input type="text" value="Enter username"/>	<input type="text" value="Enter password"/>
HEADERS	<input type="text" value="Enter key"/>	<input type="text" value="Enter value"/>
	<input type="text" value="Enter key"/>	<input type="text" value="Enter value"/>
	 Add header	



Guarda los cambios



Fulfillment

Webhook

ENABLED



Your web service will receive a POST request from Dialogflow in the form of the response to a user query matched by intents with webhook enabled. Be sure that your web service meets all the [webhook requirements](#) specific to the API version enabled in this agent.

URL*

BASIC AUTH

HEADERS



[+ Add header](#)

SMALL TALK

Inline Editor (Powered by Cloud Functions for Firebase)

DISABLED



Build and manage fulfillment directly in Dialogflow via Cloud Functions for Firebase. [Docs](#)

[index.js](#) [package.json](#)

```
1 // See https://github.com/dialogflow/dialogflow-fulfillment-nodejs
2 // for Dialogflow fulfillment library docs, samples, and to report issues
3 'use strict';
4
5 const functions = require('firebase-functions');
6 const {WebhookClient} = require('dialogflow-fulfillment');
7 const {Card, Suggestion} = require('dialogflow-fulfillment');
8
9 process.env.DEBUG = 'dialogflow:debug'; // enables lib debugging statements
10
11 exports.dialogflowFirebaseFulfillment = functions.https.onRequest((request, response) => {
12   const agent = new WebhookClient({ request, response });
13   console.log('Dialogflow Request headers: ' + JSON.stringify(request.headers));
14   console.log('Dialogflow Request body: ' + JSON.stringify(request.body));
15
16   // Handle the request here
17
18   // Return the response here
19 });
```

SAVE

3. Ahora prueba el intento y veras que ya tienes información obtenida de la base de datos.

Try it now




 See how it works in [Google Assistant](#). 

Agent

USER SAYS

COPY CURL

habitaciones

 DEFAULT RESPONSE

▼

estndar ,superior,

INTENT

habitaciones

ACTION

action1.habitacion

DIAGNOSTIC INFO

Cuando hagas cambios en dialogflow o el webhook, es necesario que repitas todo el proceso, desde iniciar webhook con el comando **python webhook_new.py &**.

para cerrar la consola de ngrok basta con que presiones las teclas ctrl+c

para cerrar el servicio de flask necesitamos escribir **kill -15**

num_recuadro_naranja(ya que este va cambiando cada que reinicias el servicios)

```

root@5b1c8a215674:/jazz# python webhook_new.py &
[1] 6
root@5b1c8a215674:/jazz# * Serving Flask app "webhook_new" (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://0.0.0.0:8080/ (Press CTRL+C to quit)

```

El comando queda de la siguiente manera

```

root@5b1c8a215674:/jazz# kill -15 6
root@5b1c8a215674:/jazz#
[1]+  Terminated                  python webhook_new.py

```

Pruebas

Como vimos, el proceso anterior es muy tardado, es por eso que hay un metodo mas rapido para que hagas peticiones desde la consola, y solo pruebes en dialogflow cuando sea necesario.

1. Abrir terminal y escribir el siguiente comando

python webhook_new.py

```

root@5b1c8a215674:/jazz# python webhook_new.py
* Serving Flask app "webhook_new" (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://0.0.0.0:8080/ (Press CTRL+C to quit)

```

2. Abrir una segunda terminal y correr el siguiente comando

curl -X POST -d '{"queryResult": {"action":

"action_del_intento_que_quieres_probar"}}' -H "Content-Type: application/json"

<http://172.17.0.2:8080/webhook>

```

jazz@jazz-XPS-8700:~$ curl -X POST -d '{"queryResult": {"action": "action1.habitacion"}}'
-H "Content-Type: application/json" http://172.17.0.2:8080/webhook

```

El resultado es :

```

{"fulfillmentText": "estndar ,superior,"}

```

Cuando hagas cambios en dialogflow o el webhook, es necesario que reinicies el servicio de flask, para ello solo presiona **ctrl+c** y corre el comando del paso número 1.

Webhook y BD

La estructura de la base de datos se encuentra dentro de la carpeta “proyecto_HotelBot”, en un archivo llamado “BD.docx”, para que te puedas basar y hagas tus propias consultas.

En el webhook se tienen 3 ejemplos de diferentes consultas que se pueden obtener de la bd, solo son un ejemplo para que despues tu comiences a armar tus propias consultas con base en tus intentos.

Abre el archivo “Webhook_new.py” desde el editor de texto de tu preferencia.

Todo el código está documentado, en caso de dudas, escríbelas al correo “19uhej@gmail.com”.