# Celery powered BOTS

Real examples for Telegram and Slack

**Xavier Orduña**
Python Meetup
March 2017

# Presentation





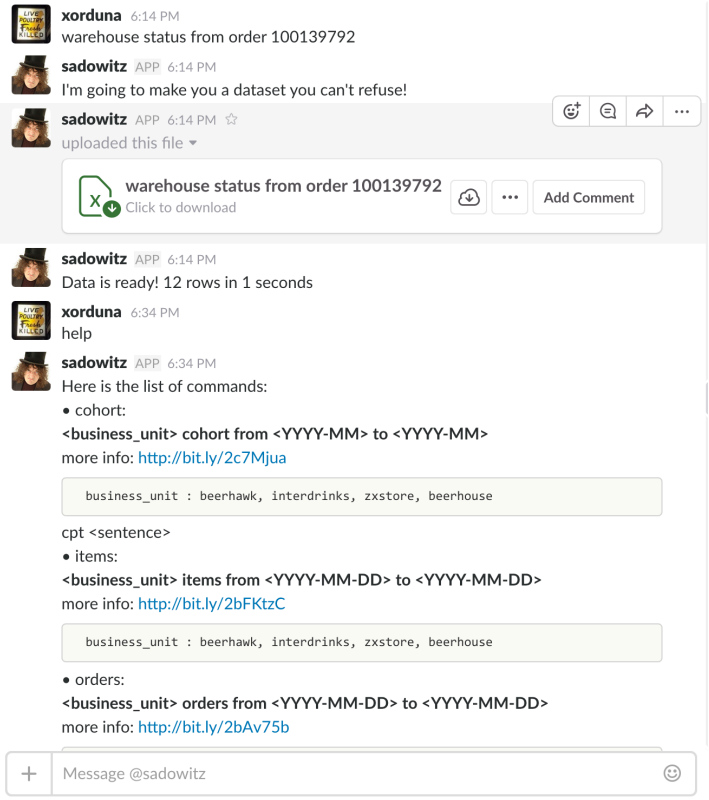
**Get more beers at www.saveur-biere.com!**

# Bot Facts

- Bots != AI
- Bots can be fully conversational or command based
- Non invasive user interface
- Bots can a replacement of Mobile and Web applications
- Bot enabled platforms: Telegram, Slack, Facebook Messenger,

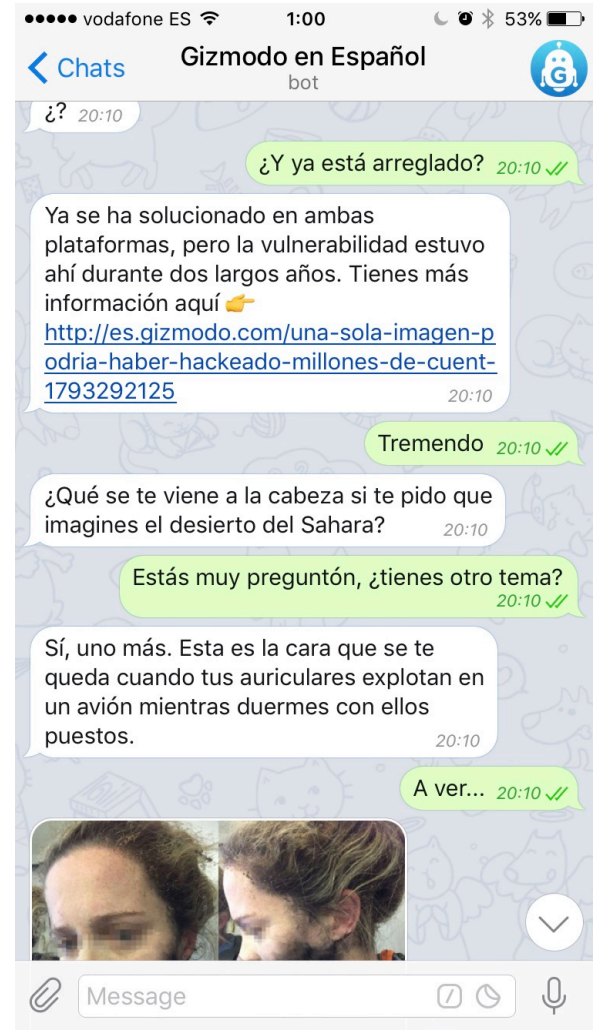
# Business Case

Deliver reporting



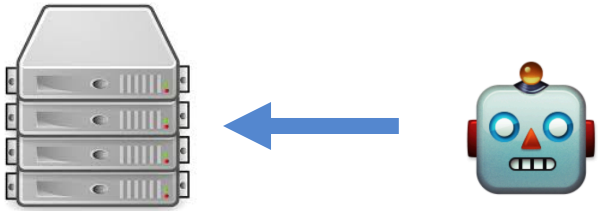

GIZMODO

Explore new ways of publishing articles



# Bot Architecture

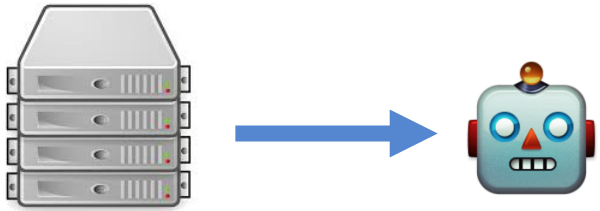
**Long Polling**



Platform API

+ perfect for test bots
+ less open connections
+ ordered messages
- Not scalable
- Not suitable for multibots

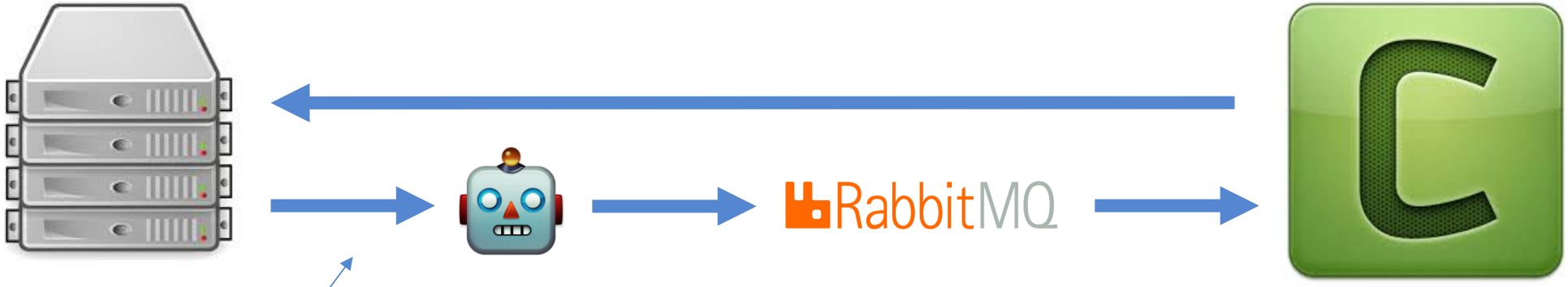
**Webhooks**



Platform API

+ scales
+ perfect for multibot
- Non ordered messages
- Requires open port with SSL -> (ngrok)
- More complex to build

# Celery based architecture




Use ngrok for local development: https://ngrok.com/

# Why Celery

- Fast (or at least fast enough for Humans)
- Easy task routing
- Compatible with **Rabbitmq**, Amazon SQS or Redis
- Flexible
- LOTS of documentation
- Flower: Dashboard
- Rabbit management Tool is priceless
- Free rabbitmq: https://www.cloudamqp.com/plans.html

# Example (Flask webapp)

Import celery task

Set webhook

Get event

Launch task

```python
from flask import Flask
from flask import request, redirect, flash, url_for, render_template
import telepot
from tasks import reply_message

ENDPOINT = "https://celerybot.ngrok.io"
app = Flask(__name__)
app.secret_key = 'this_is_my_no_so_secret_please_change_it'

@app.route("/set_webhook", methods=['GET', 'POST'])
def set_webhook():
    if request.method == 'POST':
        token = request.form['token']
        callback = "{endpoint}/telegram/{token}".format(
            endpoint=ENDPOINT,
            token=token
        )
        bot = telepot.Bot(token)
        bot.setWebhook(callback)
        flash('webhook set!')
        return redirect(url_for('set_webhook'))
    else:
        return render_template('set_webhook.html')


@app.route("/telegram/<token>", methods=['POST'])
def on_message(token):
    event = request.json
    chat_id = event['message']['chat']['id']
    # Maybe we want to store token and chat_id to contact this user later
    reply_message.delay(token, chat_id, event['message']['text'])
    return 'OK'

if __name__ == "__main__":
    app.run(host="0.0.0.0", port=3333, debug=True)
```

# Example (Celery worker)

```python
from celery import Celery, Task
import telepot
from celery.utils.log import get_task_logger
from datetime import datetime, timedelta
from celery.signals import worker_init
import time

LOCAL_BROKER = 'amqp://guest:guest@localhost/celerybot'

celery_app = Celery('tasks', broker=LOCAL_BROKER)
celery_app.conf.CELERY_ROUTES = {
        'reply_message': {
            'queue': 'bot_messages',
        }
}
logger = get_task_logger(__name__)


@worker_init.connect
def worker_start(sender, **kwargs):
    print('worker started!!!')


class MyTask(Task):
    """An abstract Celery Task to perform any action on task completion"""
    abstract = True

    def after_return(self, status, retval, task_id, args, kwargs, einfo):
        # to something
        logger.info('task completed!')
        pass


@celery_app.task(name="reply_message", bind=True, default_retry_delay=0.1, base=MyTask, rate_limit="30/s")
def reply_message(self, token, chat_id, message):
    bot = telepot.Bot(token)
    try:
        bot.sendChatAction(chat_id, "typing")
        time.sleep(0.5)
        bot.sendMessage(chat_id, u"That was she said:" + message)
    except Exception as err:
        self.retry(exc=err, max_retries=3, eta=datetime.now()+timedelta(minutes=1))
    logger.info("replied to {chat_id}".format(chat_id=chat_id))
    return 'OK'

```

Routing

Logging

Signals

Base tasks

Retry strategy

Retry strategy

# How to run

```
Webapp I:       python webapp.py
Webapp II:      uwsgi --module=webapp:app --http :3333 --threads=5 --workers=5 --master
Ngrok:          ngrok http 3333 --subdomain celerybot
Worker:         celery worker -A tasks --loglevel=INFO --queues=bot_messages –c 4
```

How to choose the number of workers?
- By default concurrency is the number of CPU's in the machine. If you add more workers, performance will be better , there is a cut-off point where performance dramatically decreases. You should find a balance.

Eventlet as Pool
- If you want to use eventlet instead of prefork (which monkey patches all standard lib blocking calls) you can run:

```
celery worker -A tasks --loglevel=INFO --queues=bot_messages –c 100 –P eventlet
```

In Amazon EC2 you should always use eventlet, because performance is very bad for –c > 2

# Libraries and Workflow

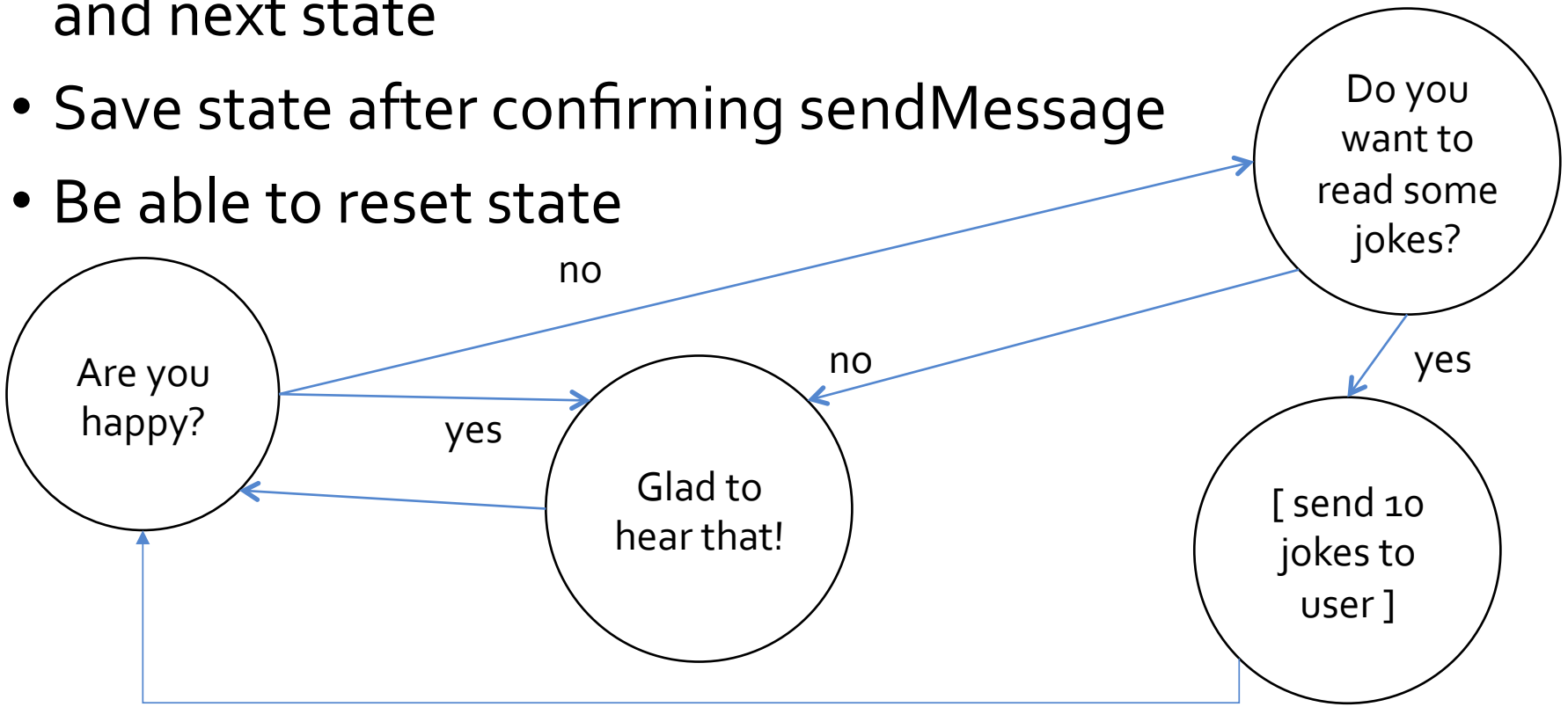
- Telepot: https://github.com/nickoala/telepot (sync and async)
- Slacker: https://github.com/os/slacker

- Get a token using @botfather
- Single token for each bot
- Webhook is set using API
- Endpoint requires SSL (not self signed)
- When webhook is set, no long polling available.

- Get token for a single bot or create an App to get key and secret and install a bot in many teams (you need a token for each team)
- Apps are associated to a team and user.
- Slacker includes Oauth helper to get credentials.
- Webhook (for events and oauth) and scope is set in https://api.slack.com/apps
- Endpoints require SSL (not self signed)

# State based bots

- Store state in database -> REDIS
- Create a function that given a message and a state, computes output and next state
- Save state after confirming sendMessage
- Be able to reset state



# Command based bots (stateless)

- Regular expressions

```
@botutils.listen_to('(.*) orders from (.*) to (.*)$', re.IGNORECASE)
def export_orders(slack_client, message, enabled_bus="", business_unit=None, day_from=None, day_to=None):
    epoch = time.time()

    logger.info('REQUEST from {user}: {req}:'.format(user=find_user_name(slack_client, message['user']), req=message['text']))
```

```
def listen_to(matchstr, flags=0):
    def wrapper(func):
        commands['listen_to'][re.compile(matchstr, flags)] = func
        return func
    return wrapper
```

```
def dispatcher(text):
    for matcher in commands['listen_to']:
        m = matcher.search(text)
        if m:
            args = m.groups()
            return commands['listen_to'][matcher], args
    return None, None
```

Tips:

- Store regular expressions in database
- Associate regular expressions with SQL queries or API endpoints (for example using AWS Lambda or Heroku)

# Telemock: Load testing

- Telemock: https://github.com/xorduna/telemock
- Emulates Telegram API
- Docker ready!
- User answers randomly in 1-9 seconds
- REST API to create users, conversations, block users, …
- Tested with 30K fake users -> in Amazon ECS
- Requires fork of telepot (https://github.com/xorduna/telepot) that enables Bot(endpoint="telemockendpoint')

# Telemock example

```python
import requests

names = ['xavi', 'elies', 'joan', 'nuria', 'diana', 'arnau']
numeric_lastnames = range(101, 500)
botname = '@telemock'
telemock_endpoint = "http://localhost:5000"

for name in names:
    for lastname in numeric_lastnames:
        username = '@'+name+str(lastname)
        print(username)
        requests.post(telemock_endpoint+'/user', json={'username': username, 'first_name': name, 'last_name': str(lastname)})
        requests.post(telemock_endpoint+'/chat', json={'botname': botname, 'username': username})
```

# Telegram tricks

- Chat_id identifies a unique conversation for each bot (bots can be added to groups)
- Send images using public URL -> is much faster
- SetWebhook for each bot (example format yourapi.com/telegram/<token>)
- Use inline keyboards and keyboards to direct user answers
- Be carefull with Telegram rate limits (celery Task supports rate_limit)
- A special message is sent at start: “/start” <- you build a link with arguments

# Slack tricks

- Single endpoint for an application (an application can include a bot)
- Set presence = True when configuring bot
- Set as_user=True (or messages will come from Application)
- Channel is the identifier of the conversation
- A bot can be installed in many teams, using Oauth you will get tokens for each team.
- For each user store channel, user id and team.

```
if 'message' in answer:
    slack_client.chat.post_message(message['channel'], answer['message'], as_user=True)
else:
    slack_client.chat.post_message(message['channel'], response.text, as_user=True)
```

# Some bots tricks

- Adding “typing” and pauses adds a human touch
- Give your bot a name and a nice picture -> Sadowitz
- Give feedback very fast, and then launch another task to do the job
- Use arrays of random sentences (“Yes”, “Alright”, “Let’s go”, “That’s all”)
- Provide always default answer that can reset state or provide help!

# Questions?

Thanks a lot!

xavier.orduna@gmail.com
http://github.com/xorduna