mattermost_bot Documentation

Release 1.0.20

Aug 03, 2017

Contents

A chat bot for Mattermost

CHAPTER 1

Contents

Installation

Compatibility

- Python: 2.6, 3.5+
- Mattermost: 3.0+

Recommended way to install is via pip:

```
pip install -U mattermost_bot
```

Check *mattermost_bot* version:

```
import mattermost_bot
print(mattermost_bot.get_version())
```

Usage

Basic

Register new user on Mattermost. Copy email/password/team and url into `mattermost_bot_settings.py` file:

```
BOT_URL = 'http://<mm.example.com>/api/v3'  # with 'http://' and with '/api/v3' path
BOT_LOGIN = '<bot-email-address>'
BOT_PASSWORD = '<bot-password>'
BOT_TEAM = '<your-team>'
```

Run the bot:

```
$ MATTERMOST_BOT_SETTINGS_MODULE=mattermost_bot_settings matterbot
```

Integration with Django

Create bot_settings on your project and after you can create `django` command:

```
import logging
import sys

from django.core.management.base import BaseCommand
from django.conf import settings

from mattermost_bot import bot, settings


class Command(BaseCommand):

    def handle(self, **options):

        logging.basicConfig(**{
            'format': '[%(asctime)s] %(message)s',
            'datefmt': '%m/%d/%Y %H:%M:%S',
            'level': logging.DEBUG if settings.DEBUG else logging.INFO,
            'stream': sys.stdout,
        })

        try:
            b = bot.Bot()
            b.run()
        except KeyboardInterrupt:
            pass
```

Modify `manage.py`:

```
#!/usr/bin/env python

import os
import sys


if __name__ == "__main__":
    os.environ.setdefault("DJANGO_SETTINGS_MODULE", "project.settings")
    os.environ.setdefault("MATTERMOST_BOT_SETTINGS_MODULE", "project.mattermost_bot_
↪settings")

    from django.core.management import execute_from_command_line

    execute_from_command_line(sys.argv)
```

Settings

`mattermost_bot` has some configuration:

```
DEBUG = False

PLUGINS = [
    'mattermost_bot.plugins',
]

# Docs + regexp or docs string only
PLUGINS_ONLY_DOC_STRING = False

# Basic configuration
BOT_URL = 'http://mm.example.com/api/v3'
BOT_LOGIN = 'bot@example.com'
BOT_PASSWORD = None
BOT_TEAM = 'devops'

# Ignore broadcast message
IGNORE_NOTIFIES = ['@channel', '@all']

# Threads num
WORKERS_NUM = 10

'''
# Custom default reply module

Example:
filename:
    my_default_reply.py
code:
    def default_reply(dispatcher, raw_msg):
        dispatcher._client.channel_msg(
            raw_msg['channel_id'], dispatcher.get_message(raw_msg)
        )
settings:
    DEFAULT_REPLY_MODULE = 'my_default_reply'
'''
DEFAULT_REPLY_MODULE = None

# or simple string for default answer
DEFAULT_REPLY = None

'''
If you use Mattermost Web API to send messages (with send_webapi()
or reply_webapi()), you can customize the bot logo by providing Icon or Emoji.
If you use Mattermost API to send messages (with send() or reply()),
the used icon comes from bot settings and Icon or Emoji has no effect.
'''
BOT_ICON = 'http://lorempixel.com/64/64/abstract/7/'
BOT_EMOJI = ':godmode:'
```

Plugins

Plugins

A chat bot is meaningless unless you can extend/customize it to fit your own use cases.

To write a new plugin, simply create a function decorated by `mattermost_bot.bot.respond_to` or `mattermost_bot.bot.listen_to`:

- A function decorated with `respond_to` is called when a message matching the pattern is sent to the bot (direct message or @botname in a channel/group chat)
- A function decorated with `listen_to` is called when a message matching the pattern is sent on a channel/group chat (not directly sent to the bot)

```
import re

from mattermost_bot.bot import listen_to
from mattermost_bot.bot import respond_to


@respond_to('hi', re.IGNORECASE)
def hi(message):
    message.reply('I can understand hi or HI!')


@respond_to('I love you')
def love(message):
    message.reply('I love you too!')


@listen_to('Can someone help me?')
def help_me(message):
    # Message is replied to the sender (prefixed with @user)
    message.reply('Yes, I can!')

    # Message is sent on the channel
    # message.send('I can help everybody!')
```

To extract params from the message, you can use regular expression:

```
from mattermost_bot.bot import respond_to


@respond_to('Give me (.*)')
def give_me(message, something):
    message.reply('Here is %s' % something)
```

If you would like to have a command like 'stats' and 'stats start_date end_date', you can create reg ex like so:

```
from mattermost_bot.bot import respond_to
import re


@respond_to('stat$', re.IGNORECASE)
@respond_to('stat (.*) (.*)', re.IGNORECASE)
def stats(message, start_date=None, end_date=None):
    pass
```

And add the plugins module to `PLUGINS` list of mattermost_bot settings, e.g. `mattermost_bot_settings.py`:

```
PLUGINS = [
    'mattermost_bot.plugins',
    'devops.plugins',           # e.g. git submodule: domain:devops-plugins.git
    'programmers.plugins',      # e.g. python package: package_name.plugins
```

```
    'frontend.plugins',         # e.g. project tree: apps.bot.plugins
]
```

For example you can separate git repositories with plugins on your team.

Attachment Support

```
from mattermost_bot.bot import respond_to


@respond_to('webapi')
def webapi_reply(message):
    attachments = [{
        'fallback': 'Fallback text',
        'author_name': 'Author',
        'author_link': 'http://www.github.com',
        'text': 'Some text here ...',
        'color': '#59afe1'
    }]
    message.reply_webapi(
        'Attachments example', attachments,
        username='Mattermost-Bot',
        icon_url='https://goo.gl/OF4DBq',
    )
    # Optional: Send message to specified channel
    # message.send_webapi('', attachments, channel_id=message.channel)
```

Deploy

Ubuntu 14.04

Install supervisor and virtualenv:

```
$ sudo apt-get update
$ sudo apt-get install supervisor python-virtualenv git
```

Add `matterbot` user:

```
$ useradd --shell /bin/bash -m -d /home/matterbot matterbot
```

Login as `matterbot`:

```
sudo -i -u matterbot
```

Clone your project (before create git repository with `settings.py`):

```
$ git clone https://github.com/USER/REPO.git ~/mybot
```

Create virtual environment:

```
$ cd ~/
$ virtualenv mm-env
```

Install project requirements:

```
$ cd ~/mybot
$ . ~/mm-env/bin/activate
$ pip install mattermost_bot
$ pip install -r requirements.txt
```

Logout:

```
$ exit
```

Configure supervisor:

```
$ sudo vi /etc/supervisor/conf.d/matterbot.conf
```

Add following config:

```
[program:matterbot]
command=/home/matterbot/mm-env/bin/matterbot
user=matterbot
directory=/home/matterbot/mybot
# should be a real settings file on MATTERMOST_BOT_SETTINGS_MODULE
environment=MATTERMOST_BOT_SETTINGS_MODULE="settings"
redirect_stderr=true
stdout_logfile=/var/log/matterbot.log
numprocs=1
autostart=true
autorestart=true
startretries=25
logfile_maxbytes=50MB
logfile_backups=3
killasgroup=true
stopasgroup=true
priority=998
```

Restart supervisor:

```
$ service supervisor restart
```

Check status:

```
$ supervisorctl status
```

Track bot logs:

```
$ tail -f /var/log/matterbot.log
```

Enjoy:)

Use cases

Standard cases for web organizations.

Support | QA

`callto John|Alex|All Your text ...` - Emergency call

`smsto John|Alex|All Your text ...` - Emergency sms

`task|bug|feature @username <low|normal|default|high|urgent> Task subject ... \nDescription` - create task on bug tracker/etc

`contact info <@username>` - Show everyone's/username emergency contact info

`remind me to TEXT at|on|in TIME` - Set a reminder for a thing, at a time

`remind @user1,@user2 to TEXT at|on|in TIME` - Set a reminder for a thing, at a time for somebody else

Development

Standard

`build VERSION` - Run build

`test CURRENT|COMMIT` - Run tests

Stage servers

`ls` - List all available staging servers

`make COMMIT <fake> or <-1h|d>` - Create stage server. optional: hour|day ago

`rm ID` - Remove server by ID

DevOps

Standard

`maintenance on|off` - Show maintenance page on website

AWS

`ls ami server1|server2|all` - Show aws AMI's

`make ami server1|server2` - Create a new ami

`upgrade|downgrade server1|server2|all <ami>` - upgrade|downgrade AMI's

`stop upgrade|downgrade` - Stop upgrade|downgrade

Distributed version control system

`update|rollback server1|server2|all <commit>` - Update code on servers

`stop update|rollback` - Stop update|rollback

Configuration management system

`cms update repo` - Update own repository (such git pull)

`cms update server1|sever2|all` - Run server provision (e.g. puppet/chef/ansible)

Internal commands for Bot

`version` - Get current revision and matterbot version

`selfupdate` - Update bot and plugins from your CVS repo

`busy` - Get num of busy workers. Used for monitoring and for check before update

`uptime` - MatterBot uptime

`ping` - Ping bot and waiting `pong`

Everything is limited only by your imagination :)

Built-in decorators

Allowed users:

```
from mattermost_bot.utils import allowed_users
from mattermost_bot.bot import respond_to


@respond_to('^is-allowed$')
@allowed_users('admin', 'root')
def access_allowed(message):
    message.reply('Access is allowed')
```

Direct messages:

```
from mattermost_bot.utils import allow_only_direct_message
from mattermost_bot.bot import respond_to


@respond_to('^direct-msg$')
@allow_only_direct_message()
def direct_msg(message):
    message.reply('Message is direct')
```

Real case

For example we have some users on our chat. We want allow some functionality to some users. We can create constants with allowed users on bot settings:

```
ADMIN_USERS = ['admin', 'root']
SUPPORT_USERS = ['mike', 'nick']
```

So now we can close access to some functions on plugins:

```
from mattermost_bot.utils import allow_only_direct_message
from mattermost_bot.utils import allowed_users
from mattermost_bot.bot import respond_to
from mattermost_bot import settings


@respond_to('^selfupdate')
@allow_only_direct_message()
```

```
@allowed_users(*settings.ADMIN_USERS)
def self_update(message):
    # some code
    message.reply('Update was done')


@respond_to('^smsto (.*)')
@allowed_users(*settings.SUPPORT_USERS)
def sms_to(message, name):
    # some code
    message.reply('Sms was sent to %s' % name)
```

Demo installation

Install OS specific packages:

```
# Ubuntu
sudo apt-get install python-virtualenv python-pip

# CentOS
sudo yum install python-virtualenv python-pip

# OS X
brew install python
```

Create project directory and virtual environment:

```
mkdir ~/my-bot && cd ~/my-bot
virtualenv venv
. venv/bin/activate
```

Install stable package from PyPi:

```
pip install mattermost_bot
```

Create settings file *mattermost_bot_settings.py* with the following lines:

```
DEBUG = True
BOT_URL = 'http://mm.example.com:8065/api/v3'
BOT_LOGIN = '<BOT_EMAIL>'
BOT_PASSWORD = '<BOT_PASSWORD>'
BOT_TEAM = '<TEAM>'
```

Run your bot by following command:

```
MATTERMOST_BOT_SETTINGS_MODULE=mattermost_bot_settings matterbot
```

Installation for development

```
$ sudo apt-get install virtualenvwrapper
$ mkvirtualenv mattermost_bot
$ git clone https://github.com/LPgenerator/mattermost_bot.git
```

```
$ cd mattermost_bot
$ python setup.py develop
$ pip install -r requirements.txt
$ pip install -r docs/requirements.txt
$ touch local_settings.py               # configure your local settings
$ matterbot                             # run bot
```

```
>>> import mattermost_bot
>>> print(mattermost_bot.get_version())
```

Contributing

If you wish to contribute, please add corresponding tests.

Running tests:

```
make test
```

Checking coverage (requires `coverage` package):

```
make coverage
```

Run tests for all python-django combinations

```
tox
```

Credits

Development Lead

- GoTLiuM InSPiRiT <gotlium@gmail.com>

Contributors

- Grokzen <grokzen@gmail.com>
- Philipp Schmitt <philipp@schmitt.co>
- Tom Horlock <thomashorlock1993@gmail.com>
- Evgeniy Poturaev <gig177@ya.ru>

CHAPTER 2

Contributing

You can grab latest code on `master` branch at Github.

Feel free to submit issues, pull requests are also welcome.

Good contributions follow *simple guidelines*