ChatOps Documentation

IT-Projects LLC

Jan 28, 2020

Contents

Note: The project is moved to https://itpp.dev/chat/ and will be shutdowned here soon

CHAPTER 1

ToDo bot

1.1 Description

Allows to create TODOs for a small group of users.

Tasks can have on of the following states:

- TODO – to be done
- DONE – done
- CANCELED – nothing was done and not going to be done
- RELOCATED – task is moved to another task management tool

Warning: Bot keeps only message IDS and task's description (which is text of the first message). It means that if chat history is deleted, then some task information could be lost. Chat history could be deleted either by user or by telegram itself. What we know about the latter case is that "Bot storage is limited", though it's unknow how much or how long messages are kept in telegram servers.

1.2 Technical specification

- `/mytasks`, `/tasks_from_me` – shows tasks. By default it shows only WAITING and TODO tasks
  - Prints all tasks in a single message with button "Load more Done", "Load more WAITING", "Load more Canceled"
- **`/t123` – shows specific task.**
  - Prints original forwarded messages
  - You can change status from here
- `/attach123` – attach new messages to a task
- `/stop_attaching` – treat next messages as new task
- `/assign123` – assign a task to another person
- `/update_id` – current update_id. Can be used to set `MIN_UPDATE_ID` (see below)
- `/myid` – Id and Name of current user

To create new task:

- Forward message to the bot
- Assign to a user from the list

1.3 Roadmap

1.4 Deployment

1.4.1 Create a bot

https://telegram.me/botfather – follow instruction to set bot name and get bot token

1.4.2 Prepare zip file

To make a deployment package execute following commands:

```
mkdir /tmp/todo-bot
cd /tmp/todo-bot

pip2 install pyTelegramBotAPI -t .
wget https://gitlab.com/itpp/chatops/raw/master/todo-bot/lambda_function.py -O lambda_
↪function.py
zip -r /tmp/todo_bot_package.zip *
```

1.4.3 Create DynamoDB tables

In *AWS (Amazon Web Services): DynamoDB service*

Tasks table

It's used to save tasks

- *Partition key:* `id` (number)
- Unmark `[ ] Use default settings` checkbox

Add Secondary index:

- *Partition key:* `from_id` (number)
- *Sort key:* `task_state` (number)
- *Index name:* `from_id-task_state-index`
- *Projected attributes:* `Include` – then add field `to_id`, `description`, `telegram_unixtime`, `msg_num`

Add another Secondary index:

- *Partition key:* `to_id` (number)
- *Sort key:* `task_state` (number)
- *Index name:* `to_id-task_state-index`
- *Projected attributes:* `Include` – then add field `from_id`, `description`, `telegram_unixtime`, `msg_num`, `next_reminder`

Users table

It's used to save current user activity. For example, if user sends batch of forwarded message, we need to change user status to save all messages to a single task.

- *Partition key:* `user_id` (number)

1.4.4 Create Lambda function

Runtime

In *AWS: Lambda service*

Use `Python 2.7`

Environment variables

In *AWS: Lambda service*

- `BOT_TOKEN` – the one you got from BotFather
- `USERS` – Dictionary of users who can be assigned to a task. Format: `{"USER_ID": "USER_NAME"}`. At this moment there is no API to get list of members. As a workaround you can ask users to send /myid command to get name and id and prepare the dictionary manually. To use emoji in user names to as following:
  - Get emoji code via http://www.webpagefx.com/tools/emoji-cheat-sheet/
  - Install python lib: https://pypi.python.org/pypi/emoji
  - Prepare json in python console:

```
import emoji
import json
d = {"123": ":thumbsup: Ivan"}
print(json.dumps(dict([(k, emoji.emojize(v, use_aliases=True)) for k, v in d.
↪items()])))
```

- `DYNAMODB_TABLE_TASK` – table with tasks (name of the table)
- `DYNAMODB_TABLE_USER` – table with users (name of the table)
- `LOG_LEVEL` – `DEBUG` or `INFO`
- `MIN_UPDATE_ID` – Number to distract from update_id in task's id computation. Use `/update_id` to get value.
- `FORWARDING_DELAY` – max seconds to wait for next forwarded message. It's a workaround for limitation of telegram API – it sends forwarded messages one by one and never in a single event. Default is 3 sec.
- `REMINDER_DAYS` – how much days to wait before remind a user about open task

Trigger

In *AWS: Lambda service*

- **API Gateway**. Once you configure it and save, you will see `Invoke URL` under Api Gateway **details** section
- **CloudWatch Events**. Create new rule for reminders, for example set
  - *Rule name* – `boto-todo-reminder`
  - *Schedule expression* – `rate(1 day)`

Role

In *AWS: IAM (Identity and Access Management) service: Policies*

- Create policy of actions for DynamoDB:
  - *Service* – `DynamoDB`
  - *Action* – `All DynamoDB actions`
  - *Resources* – `All Resources`

In *AWS: IAM service: Roles*

In list of roles choose the role, which was named in process of creating lambda function, and attach to it recently created policy for DynamoDB

- The role must allow access to lambda and dynamodb services.

By the final, role should look something like this:

In *AWS: Lambda service: Designer: View Permissions (Key-Icon)*

```
{

     "roleName": "{ROLE_NAME}",
      "policies": [
        {
          "document": {
            "Version": "2012-10-17",
            "Statement": [
              {
                "Effect": "Allow",
                "Action": [
                  "logs:CreateLogGroup",
                  "logs:CreateLogStream",
                  "logs:PutLogEvents"
                ],
                "Resource": [
                  "arn:aws:logs:*:*:*"
                ]
              }
            ]
          },
          "name": "AWSLambdaEdgeExecutionRole-daf8b371-4fc9-4e1a-9809-fcd44b96d4f2",
          "id": "ANPAX7765LQXBC72HXN4W",
          "type": "managed",
          "arn": "arn:aws:iam::549753543726:policy/service-role/
↪AWSLambdaEdgeExecutionRole-daf8b371-4fc9-4e1a-9809-fcd44b96d4f2"
          },
        {
          "document": {
            "Version": "2012-10-17",
            "Statement": [
              {
                "Sid": "VisualEditor0",
                "Effect": "Allow",
                "Action": "dynamodb:*",
                "Resource": "*"
              }
            ]
          },
          "name": "{NAME_OF_POLICY_FOR_DYNAMODB}",
          "id": "ANPAX7765LQXJUGC2FXMV",
          "type": "managed",
          "arn": "arn:aws:iam::549753543726:policy/{NAME_OF_POLICY_FOR_DYNAMODB}"
        }
      ],
      "trustedEntities": [
        "edgelambda.amazonaws.com",
        "lambda.amazonaws.com"
```

(continues on next page)

(continued from previous page)

```
        ]

}
```

Timeout

in *AWS: Lambda service*

Execution time depends on telegram server and amount of requests there. So, think about 30 seconds for limit.

Concurrency

in *AWS: Lambda service*

You may need to disable concurrency (i.e. set **Reserve concurrency** to value **1**) as a workaround for following issue: on resending batch of messages, those might be processed by several workers, so you might get several messages instead of a single one.

1.4.5 Register webhook at telegram

via python lib

Execute once in python console:

```
BOT_TOKEN = "PASTETHETOKEN"
WEB_HOOK = "PASTEAWSWEBHOOK"

import telebot  # https://github.com/eternnoir/pyTelegramBotAPI
bot = telebot.TeleBot(BOT_TOKEN, threaded=False)
bot.set_webhook(WEB_HOOK)
```

via curl

```
# TODO pass allowed_updates arg
curl -XPOST https://api.telegram.org/bot<YOURTOKEN>/setWebhook\?url\=YOURAPIGATEWAYURL
```

via url

Type the following in your browser and hit enter. (Make sure to substitute the place holder text):

```
https://api.telegram.org/bot<your-bot-token>/setWebHook?url=<your-API-invoke-URL>
```

CHAPTER 2

Resend bot

2.1 Description

The general idea is to ask questions within the Telegram group (let's call it *Target group*) and get answers.

The group might be:

- Support Team
- IT Department of your company
- etc.

2.2 Technical specification

2.2.1 Chat ID

`/thischat` – returns id of current chat. It's used for the identification of the Telegram group. `/myid` – returns id of a user

2.2.2 Send message

The are following ways for sending *message-reqest* to the bot:

- In **private chat** with the bot: any message
  - e.g. *hello, how are you?, etc*
- In **another** Telegram group (different from *Target group*): any message that starts with / and ends with @<*name_of_the_*>*bot* is used as a response to the Bot's message
  - e.g. */hey@super_bot*, */please@request_bot*, etc.

2.2.3 Get message

The *Target group* receives a copy of the message with a reference to the sender and the original message itself.

2.2.4 Response

Users response to the bot, in so doing:

- The copy of the response is sent back to chat, which contains the original message.
- Response from *Target group* is anonymous by default, but could be customized.

2.2.5 Examples

- In the *original chat* from Ivan `/hey Answer to the Ultimate Question of Life, the Universe, and Everything?`
- In the *Target group* from @name_of_the_bot: `<a href="userlink">Ivan</a>: Answer to the Ultimate Question of Life, the Universe, and Everything? *msg:<message1>:<chat>*`
- In the *Target group* from @Deep_thought1:`Anybody knows the answer?`

- In the *Target group* from @Deep_thought2:`Let me think a little bit?`
- In the *Target group* from @Deep_thought1:`?`
- In the *Target group* from @Deep_thought2:`Ok, I found and checked the answer!`
- In the *Target group* from @Deep_thought2: *In reply to Ivan: Answer to the Ultimate Question ...* `The answer is 42!`
- In the *Original chat* from @name_of_the_bot: `The answer is 42! *msg:<message2>*`

2.3 Deployment

2.3.1 Create a bot

https://telegram.me/botfather – follow instruction to set bot name and get bot token.

Check your steps:

- Use the /newbot command to create a new bot first.
- The name of the bot must be end witn "bot" (e.g. TetrisBot or tetris_bot).
- Keep your token secure and store safely, it can be used by anyone to control your bot.

2.3.2 Prepare zip file

To make deployment package execute following commands:

```
mkdir /tmp/resend-bot
cd /tmp/resend-bot

pip2 install pyTelegramBotAPI -t .
wget https://gitlab.com/itpp/chatops/raw/master/resend-bot/lambda_function.py -O␣
↪lambda_function.py
zip -r /tmp/resend_bot_package.zip *
```

2.3.3 Create Lambda function

Runtime

Use `Python 2.7`

Environment variables

- `BOT_TOKEN` – the one you got from BotFather
- `TARGET_GROUP` – put here Chat ID from the Target group using `/thischat` command
  - Note: ID number may contains the "-" before number
- `ANONYMOUS_REPLY` – whether to send replies anonymously. Default True.
- `AANONYMOUS_REQUEST_FROM_GROUPS` – whether to show author name on requesting from another group. Default True.

- `ACCESS_BOT_LIST` – List of ID's (users) which can use the bot. If empty - everyone can.
- `LOGGING_LEVEL` – Level of loger. (Allowed values: DEBUG, INFO, CRITICAL, ERROR, WARNING), by default: INFO

Trigger

- **API Gateway**. Once you configure it and save, you will see `Invoke URL` under Api Gateway **details** section
- Set the security mechanism for your API endpoint as Open

2.3.4 Register webhook at telegram

- Replace "PASTETHETOKEN" with your Telegram HTTP API access token.
- Replace "PASTEAWSWEBHOOK" with your Invoke URL obtained in the previous section.
- Run following command

via python lib

Execute once in python console:

```
BOT_TOKEN = "PASTETHETOKEN"
WEB_HOOK = "PASTEAWSWEBHOOK"

import telebot  # https://github.com/eternnoir/pyTelegramBotAPI
bot = telebot.TeleBot(BOT_TOKEN, threaded=False)
bot.set_webhook(WEB_HOOK, allowed_updates=['message'])
```

via curl

```
# TODO pass allowed_updates arg
curl -XPOST https://api.telegram.org/bot<YOURTOKEN>/setWebhook\?url\=YOURAPIGATEWAYURL
```

CHAPTER 3

Posting notifications in telegram via bot

This telegram bot substitutes ifttt's applet "Webhook to Telegram", which may work slowly.

3.1 Deployment

3.1.1 Create a bot

- In telegram client open BotFather
- Send `/newbot` command to create a new bot
- Follow instruction to set bot name and get bot token
- Keep your token secure and store safely, it can be used by anyone to control your bot

3.1.2 Prepare zip file

To make deployment package execute following commands:

```
mkdir /tmp/bot
cd /tmp/bot

pip3 install pyTelegramBotAPI -t .
wget https://gitlab.com/itpp/chatops/raw/master/ifttt-to-telegram/lambda_function.py
zip -r /tmp/bot.zip *
```

3.1.3 Create Lambda function

- Navigate to https://console.aws.amazon.com/lambda/home
- Click *Create function*
- Configure the function as described below

Runtime

Use `Python 3.6`

Function code

- Set **Code entry type** to *Upload a .zip file*
- Select `bot.zip` file you made

Environment variables

- `BOT_TOKEN` – the one you got from BotFather
- `TELEGRAM_CHAT` – where to send notification. You can chat id by sending any message to Get My ID bot
- `EVENT_<EVENT_NAME>` – set response value. Use IFTTT syntax. For example:

  `EVENT_RED_PULL_REQUEST` set to value `PR TESTS are failed: {{Value1}}<br> {{Value2}}.`

Trigger

- **API Gateway**. Once you configure it and save, you will see `Invoke URL` under Api Gateway **details** section
- Set the security mechanism for your API endpoint as *Open*

3.2 Try it out

Use URL of the following format to replace with your ifttt webhook URL:

`<INVOKE_URL>?event=<EVENT_NAME>`, for example `https://9ltrkrik2l.execute-api.eu-central-1.amazonaws.com/default/MyLambda?event=RED_PULL_REQUEST`