Chat ’n’ Hook Documentation

brantje

Mar 19, 2018

Menu

A **service** represents any web service that sends a webhook.

A **event** is identified from the given webhook (like a push on Github, for example).

A **comm** is then used to publish information from this hook in messaging services.

A **project** is used to contain multiple configurations.

CHAPTER 1

Requirements

- **Python 2.7 / 3**
  - pip setuptools
  - pip wheel
- A hostname + valid tld.
- A reverse proxy (Apache / Nginx)
- For telegram bots: A valid SSL cert

CHAPTER 2

Configuration

2.1 Setting up the bot

Configuring this bot is simple. Copy `config.example.yml` to `config.yml`.

After that edit the file, below more information is given.

`global.bot_url` - The bot url, this url will be used for creating redirects

The `comms` section allows you to set global configuration for that comm. A comm is used to send information about a webhook. For example, if you have configured **slack** and **telegram**, and Github would send a push event. Then that event is published to slack and telegram. The configuration options should be self-explaining, if not, feel free to open an issue.

The `services` section allows to set global configuration for that service. For more details see *Service Configuration*.

The `hooks` section is where is real fun begins. This holds all information that will send every webhook in to the right direction.

2.2 Webserver configuration

Generally it's advised to use a reverse proxy in front of the flask application. Below you can find configs for various webservers:

- *NGINX*
- *Apache2*

2.2.1 NGINX

```
server {
        listen 80;
        listen [::]:80;

        server_name your.bot.url;
        location / {
             proxy_pass  http://127.0.0.1:5000;
             proxy_next_upstream error timeout invalid_header http_500 http_502 http_
↪503 http_504;
             proxy_redirect off;
             proxy_buffering off;
             proxy_set_header        Host            $host;
             proxy_set_header        X-Real-IP       $remote_addr;
             proxy_set_header        X-Forwarded-For $proxy_add_x_forwarded_for;
             proxy_set_header        X-Forwarded-Proto $scheme;
         }

        location ~* \.(ico|css|js|gif|jpeg|jpg|png|woff|ttf|otf|svg|woff2|eot)$ {
            expires 1d;
            access_log off;
            add_header Pragma public;
            add_header Cache-Control "public, max-age=86400";
            add_header X-Asset "yes";
            proxy_pass http://192.168.1.16:5000;
            proxy_next_upstream error timeout invalid_header http_500 http_502 http_
↪503 http_504;
            proxy_redirect off;
            proxy_buffering off;
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header X-Forwarded-Proto $scheme;
            gzip on;
            gzip_disable "msie6";

            gzip_vary on;
            gzip_proxied any;
            gzip_comp_level 5;
            # gzip_buffers 16 8k;
            gzip_http_version 1.1;
            gzip_types text/plain text/css application/json application/javascript␣
↪text/xml application/xml application/xml+rss text/javascript;

        }


    listen 443 ssl; # managed by Certbot
    ssl_certificate /etc/letsencrypt/live/your.bot.url/fullchain.pem; # managed by␣
↪Certbot
    ssl_certificate_key /etc/letsencrypt/live/your.bot.url/privkey.pem; # managed by␣
↪Certbot
    include /etc/letsencrypt/options-ssl-nginx.conf; # managed by Certbot
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem; # managed by Certbot
```

(continues on next page)

(continued from previous page)

```
    if ($scheme != "https") {
        return 301 https://$host$request_uri;
    } # managed by Certbot

}
```

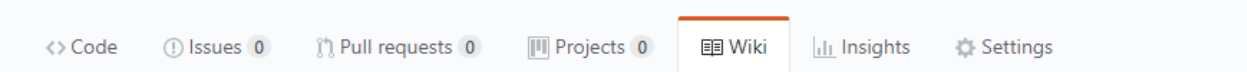
2.2.2 Apache2

```
<IfModule mod_ssl.c>
<VirtualHost *:443>
        #server information
        ServerName your.bot.url
        ServerAdmin webmaster@example.com
        RequestHeader set X-Forwarded-Proto "https"

        ProxyPreserveHost Off
        ProxyPass / http://127.0.0.1:5000/
        ProxyPassReverse / http://127.0.0.1:5000/

        ErrorLog ${APACHE_LOG_DIR}/error.log
        CustomLog ${APACHE_LOG_DIR}/access.log combined

        #SSL Settings Managed by certbot
        SSLCertificateFile /etc/letsencrypt/live/your.bot.url/fullchain.pem
        SSLCertificateKeyFile /etc/letsencrypt/live/your.bot.url/privkey.pem
        Include /etc/letsencrypt/options-ssl-apache.conf
</VirtualHost>
</IfModule>
```

CHAPTER 3

Service Configuration

3.1 Github



Browse to the repository you want to add, and click on settings in the menu.

In the left menu choose **Webhooks**.
Then click on **Add webhook**.

Under **Payload URL** enter the following: `https://your.bot.url/<project>/github`.
In this case the `/github` tells chat 'n' hook that the webhooks are coming from github.
The `/<project>` tells chat 'n' hook what project it is.

Make sure you have `/<project>/github` **on the end of your url**

After you've set the Payload url, set **Content type** to `application/json`.
And if you like to protect your endpoint, you can enter a `secret`.
Make sure to enter it in the config too.

Last and final: **Which events would you like to trigger this webhook?** select all events

Last one, click **Add webhook**

3.1.1 Example config

```
services:
  github:
    token: 'xxxxxxxx'

hooks:
  <project>:
    github:
      enabled: true
      secret: ''
      token: ''
      scripts:
        push:
          - python /path/to/script.py
      notify_branches:
        - master
      disabled_events: # Allow all but those here
        - create
      notify_events: # Deny all but those here
        - push
        - commit_commment
        - watch
      send_to:
        telegram:
          enabled: true
          token: "xx:xxx"
          channels:
            - "-xxxxx"
        slack:
          enabled: false
          bot_name: 'Bot'
          webhooks:
            - 'https://hooks.slack.com/services/xxxx/xxxxx/xxxx'
        mattermost:
          enabled: false
          bot_name: 'Bot'
          webhooks:
            - 'https://mattermost.xxxxx.com/hooks/xxxx'
        discord:
          enabled: false
          bot_name: 'Bot'
          webhooks:
            - 'https://discordapp.com/api/webhooks/xxxx/xxxxxxx'
```

About the services.github config
The `services.github.token` is used for api communication.
You don't need to supply a token, but you will hit the rate limits of github quite fast.
You can generate a token here, no need to check any checkboxes, public access is good enough.

About the hooks.<project>.github config
The `secret` is the secret key in the github webhook. For more info see above.

3.2 Bitbucket

Browse to the repository you want to add, and click on settings in the menu.

In the left menu choose **Webhooks**. Then click on **Add webhook**.

Under **URL** enter the following: `https://your.bot.url/<project>/bitbucket`.
In this case the `/bitbucket` tells chat 'n' hook that the webhooks are coming from bitbucket.
The `/<project>` tells chat 'n' what project it is.

Make sure you have `<project>/bitbucket` **on the end of your url**

Last and final: **Triggers -> Choose from a full list of triggers** and select the events you want.

Last one, click **Save**

3.2.1 Example config

```
services:
  bitbucket:
    token: 'xxxxxxxx'

hooks:
  <project>:
    github:
      enabled: true
      secret: ''
      token: ''
      scripts:
        push:
          - python /path/to/script.py
      notify_branches:
        - master
      disabled_events: # Allow all but those here
        - create
      notify_events: # Deny all but those here
        - push
        - commit_commment
        - watch
      send_to:
        telegram:
          enabled: true
          token: "xx:xxx"
          channels:
            - "-xxxxx"
        slack:
          enabled: false
          bot_name: 'Bot'
          webhooks:
            - 'https://hooks.slack.com/services/xxxx/xxxxx/xxxx'
        mattermost:
          enabled: false
          bot_name: 'Bot'
```

(continues on next page)

(continued from previous page)

```
          webhooks:
            - 'https://mattermost.xxxxx.com/hooks/xxxx'
        discord:
          enabled: false
          bot_name: 'Bot'
          webhooks:
            - 'https://discordapp.com/api/webhooks/xxxx/xxxxxxx'
```

The `token` is used for api communication. You don't need to supply a token, but you will hit the rate limits of bitbucket quite fast.

3.3 Patreon

Go to here.

In **Create a new webhook by pasting your URL here** enter the following: `https://your.bot.url/<project>/patreon`.

In this case the `<project>/patreon` tells chat 'n' hook that the webhooks are coming from patreon.
The `/<project>` tells chat 'n' what project it is.
Make sure you have `/<project>/patreon` **on the end of your url**

3.3.1 Example config

```
hooks:
  <project>:
    patreon:
      enabled: true
      scripts:
        pledges.create:
          - python /path/to/script.py
      send_to:
        telegram:
          enabled: true
          token: "xx:xxx"
          channels:
            - "-xxxxx"
        slack:
          enabled: false
          bot_name: 'Bot'
          webhooks:
            - 'https://hooks.slack.com/services/xxxx/xxxxx/xxxx'
        mattermost:
          enabled: false
          bot_name: 'Bot'
          webhooks:
            - 'https://mattermost.xxxxx.com/hooks/xxxx'
        discord:
          enabled: false
```

(continues on next page)

(continued from previous page)

```
          bot_name: 'Bot'
          webhooks:
            - 'https://discordapp.com/api/webhooks/xxxx/xxxxxx'
```

3.4 Docker

Login to the Docker Hub.
Pick the repo you want to add the webhook to.
In the menu you see Webhooks, click on it.
Enter a name and enter `https://your.bot.url/<project>/docker` as url.

In this case the `/docker` tells chat 'n' hook that the webhooks are coming from Docker.
The `/<project>` tells chat 'n' hook what project it is.
Make sure you have `/<project>/docker` on the end of your url

3.4.1 Example config

```
hooks:
  <project>:
    docker:
      enabled: true
      scripts:
        push:
          - python /path/to/script.py
      send_to:
        telegram:
          enabled: true
          token: "xx:xxx"
          channels:
            - "-xxxxx"
        slack:
          enabled: false
          bot_name: 'Bot'
          webhooks:
            - 'https://hooks.slack.com/services/xxxx/xxxxx/xxxx'
        mattermost:
          enabled: false
          bot_name: 'Bot'
          webhooks:
            - 'https://mattermost.xxxxx.com/hooks/xxxx'
        discord:
          enabled: false
          bot_name: 'Bot'
          webhooks:
            - 'https://discordapp.com/api/webhooks/xxxx/xxxxxx'
```

3.5 Slack

Add the **Outgoing Webhooks** integration or click here to add it.

Use the following configuration under **Integration Settings**.

Channel: pick the channel you send webhooks from or use all.
Trigger word(s): Up to you
URL(s): `https://your.bot.url/<project>/slack`
Token: Recommend to use prefilled token.

The rest is entirely up to you to configure (label, name, picture, etc)

In this case the `/slack` tells chat 'n' hook that the webhooks are coming from Slack.
The `/<project>` tells chat 'n' hook what project it is.
Make sure you have `/<project>/slack` **on the end of your url**

3.5.1 Example config

```
hooks:
  <project>:
    slack:
      token: "your_outgoing_hook_token"
      scripts:
        message:
          - python /path/to/script.py
      send_to:
        telegram:
          enabled: true
          token: "xx:xxx"
          channels:
            - "-xxxxx"
        slack:
          enabled: false
          bot_name: 'Bot'
          webhooks:
            - 'https://hooks.slack.com/services/xxxx/xxxxx/xxxx'
        mattermost:
          enabled: false
          bot_name: 'Bot'
          webhooks:
            - 'https://mattermost.xxxxx.com/hooks/xxxx'
        discord:
          enabled: false
          bot_name: 'Bot'
          webhooks:
            - 'https://discordapp.com/api/webhooks/xxxx/xxxxxxx'
```

Replace `your_outgoing_hook_token` with the token.

3.6 CircleCI

CircleCI supports sending webhooks when your build completes.
More information can be found here.
Add the following to `circle.yml`

```
notify:
  webhooks:
    - url:  https://your.bot.url/<project>/circleci
```

In this case the `/circleci` tells chat 'n' hook that the webhooks are coming from CircleCI.
The `/<project>` tells chat 'n' hook what project it is.
Make sure you have `/<project>/circleci` on the end of your url

3.6.1 Example config

```
hooks:
  <project>:
    circleci:
      token: ''
      scripts:
        build_complete:
          - python /path/to/script.py
      send_to:
        telegram:
          enabled: true
          token: "xx:xxx"
          channels:
            - "-xxxxx"
        slack:
          enabled: false
          bot_name: 'Bot'
          webhooks:
            - 'https://hooks.slack.com/services/xxxx/xxxxx/xxxx'
        mattermost:
          enabled: false
          bot_name: 'Bot'
          webhooks:
            - 'https://mattermost.xxxxx.com/hooks/xxxx'
        discord:
          enabled: false
          bot_name: 'Bot'
          webhooks:
            - 'https://discordapp.com/api/webhooks/xxxx/xxxxxxx'
```

3.7 TravisCI

TravisCI supports sending webhooks when your build completes.
More information can be found here

Add the following to *.travis.yml*

```
notifications:
  webhooks: https://your.bot.url/<project>/travis
```

In this case the `/travis` tells chat 'n' hook that the webhooks are coming from Travis.
The `/<project>` tells chat 'n' hook what project it is.
Make sure you have `/<project>/travis` **on the end of your url**

3.7.1 Example config

```
hooks:
  <project>:
    docker:
      enabled: true
      scripts:
        build:
          - python /path/to/script.py
      send_to:
        telegram:
          enabled: true
          token: "xx:xxx"
          channels:
            - "-xxxxx"
        slack:
          enabled: false
          bot_name: 'Bot'
          webhooks:
            - 'https://hooks.slack.com/services/xxxx/xxxxx/xxxx'
        mattermost:
          enabled: false
          bot_name: 'Bot'
          webhooks:
            - 'https://mattermost.xxxxx.com/hooks/xxxx'
        discord:
          enabled: false
          bot_name: 'Bot'
          webhooks:
            - 'https://discordapp.com/api/webhooks/xxxx/xxxxxxx'
```

3.8 Scrutinizer

Browse to the repository you want to add, and click on settings in the menu.

In the left menu choose **Service Hooks**. Then click on **Add Web-Hook**.

Under **URL** enter the following: `https://your.bot.url/<project>/scrutinizer`.

In this case the `/scrutinizer` tells chat 'n' hook that the webhooks are coming from Scrutinizer.
The `/<project>` tells chat 'n' what project it is.
Make sure you have `/<project>/scrutinizer` **on the end of your url**

Last one, click **Add Webhook**

3.8.1 Example config

```
hooks:
  <project>:
    scrutinizer:
      enabled: true
      events:
        - inspection.created
        - inspection.completed
        - inspection.canceled
        - inspection.failed
      notify_branches:
        - master
      scripts:
        build_complete:
          - python /path/to/script.py
      send_to:
        telegram:
          enabled: true
          token: "xx:xxx"
          channels:
            - "-xxxxx"
        slack:
          enabled: false
          bot_name: 'Bot'
          webhooks:
            - 'https://hooks.slack.com/services/xxxx/xxxxx/xxxx'
        mattermost:
          enabled: false
          bot_name: 'Bot'
          webhooks:
            - 'https://mattermost.xxxxx.com/hooks/xxxx'
        discord:
          enabled: false
          bot_name: 'Bot'
          webhooks:
            - 'https://discordapp.com/api/webhooks/xxxx/xxxxxxx'
```

3.9 Codacy

Browse to the repository you want to add, and click on settings in the menu.

In the menu on top, click on **Integrations**
Then click on **Add integration**.

A popup shows, click on **WebHook**.

Under **Endpoint Address** enter the following: `https://your.bot.url/<project>/codacy`.

In this case the `/codacy` tells chat 'n' hook that the webhooks are coming from scrutinizer.
The `/<project>` tells chat 'n' hook what project it is.
Make sure you have* ''/<project>/codacy'' on the end of your url

Last one, click **Save**

3.9.1 Example config

```
hooks:
  <project>:
    codacy:
      enabled: true
      scripts:
        review_complete:
          - python /path/to/script.py
      send_to:
        telegram:
          enabled: true
          token: "xx:xxx"
          channels:
            - "-xxxxx"
        slack:
          enabled: false
          bot_name: 'Bot'
          webhooks:
            - 'https://hooks.slack.com/services/xxxx/xxxxx/xxxx'
        mattermost:
          enabled: false
          bot_name: 'Bot'
          webhooks:
            - 'https://mattermost.xxxxx.com/hooks/xxxx'
        discord:
          enabled: false
          bot_name: 'Bot'
          webhooks:
            - 'https://discordapp.com/api/webhooks/xxxx/xxxxxxx'
```

3.10 Telegram

Chat 'n' can be used as a Telegram bot.
The bot is very extensible, you can add your own commands.
For more information about adding commands see *Adding telegram command*.

3.10.1 Requirements

- A domain
- A valid SSL certificate (Let's encrypt for example)
- A valid bot token
- Common sense

3.10.2 Example config

```
services:
  telegram:
    enabled: true
    server_key: '' # Only when using a custom ssl cert
    server_cert: '' # Only when using a custom ssl cert
    hostname: 'your.bot.url'
    port: 443
    permissions:
        admins:
            - username1
        moderators:
            - username2
    token: "xxxx:xxxx"
    plugins:
      giphy:
        apiKey: 'xxxx'
```

Use `server_key` and `server_cert` only when you use an self signed certificate.
Set `hostname` to your bots hostname.
Under `permissions` add as many admins / moderators as you like.
Paste your token under `token`

3.11 Zabbix

Chat 'n' hook has the ability to receive reports from Zabbix.

To receive hooks from zabbix you need to install the Chat 'n' Hook script and configure it.

3.11.1 Example config

```
hooks:
  <project>:
    zabbix:
      scripts:
        issue:
          - python /path/to/script.py
      send_to:
        telegram:
          enabled: true
          token: "xx:xxx"
          channels:
            - "-xxxxx"
        slack:
          enabled: false
          bot_name: 'Bot'
          webhooks:
            - 'https://hooks.slack.com/services/xxxx/xxxxx/xxxx'
        mattermost:
          enabled: false
          bot_name: 'Bot'
          webhooks:
            - 'https://mattermost.xxxxx.com/hooks/xxxx'
        discord:
          enabled: false
          bot_name: 'Bot'
          webhooks:
            - 'https://discordapp.com/api/webhooks/xxxx/xxxxxxx'
```

CHAPTER 4

Comm Configuration

4.1 Slack

Add the **Incoming Webhooks** integration or click here to add it.

Use the following configuration under **Integration Settings**.

Channel: pick the channel to publish events to

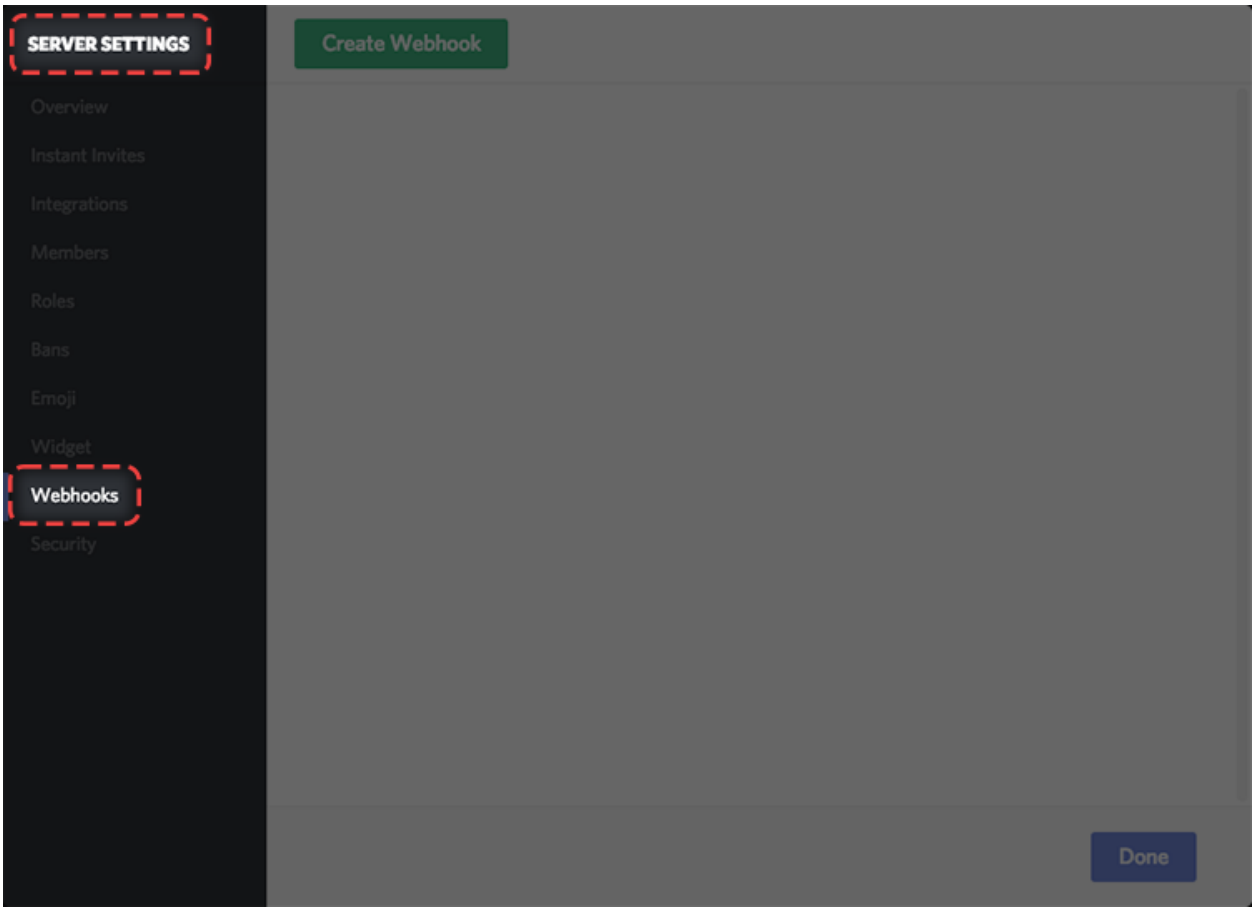
4.1.1 Example config

```
comms:
  slack:
    enabled: true
    hook_url: "slack_webhook_here"
```

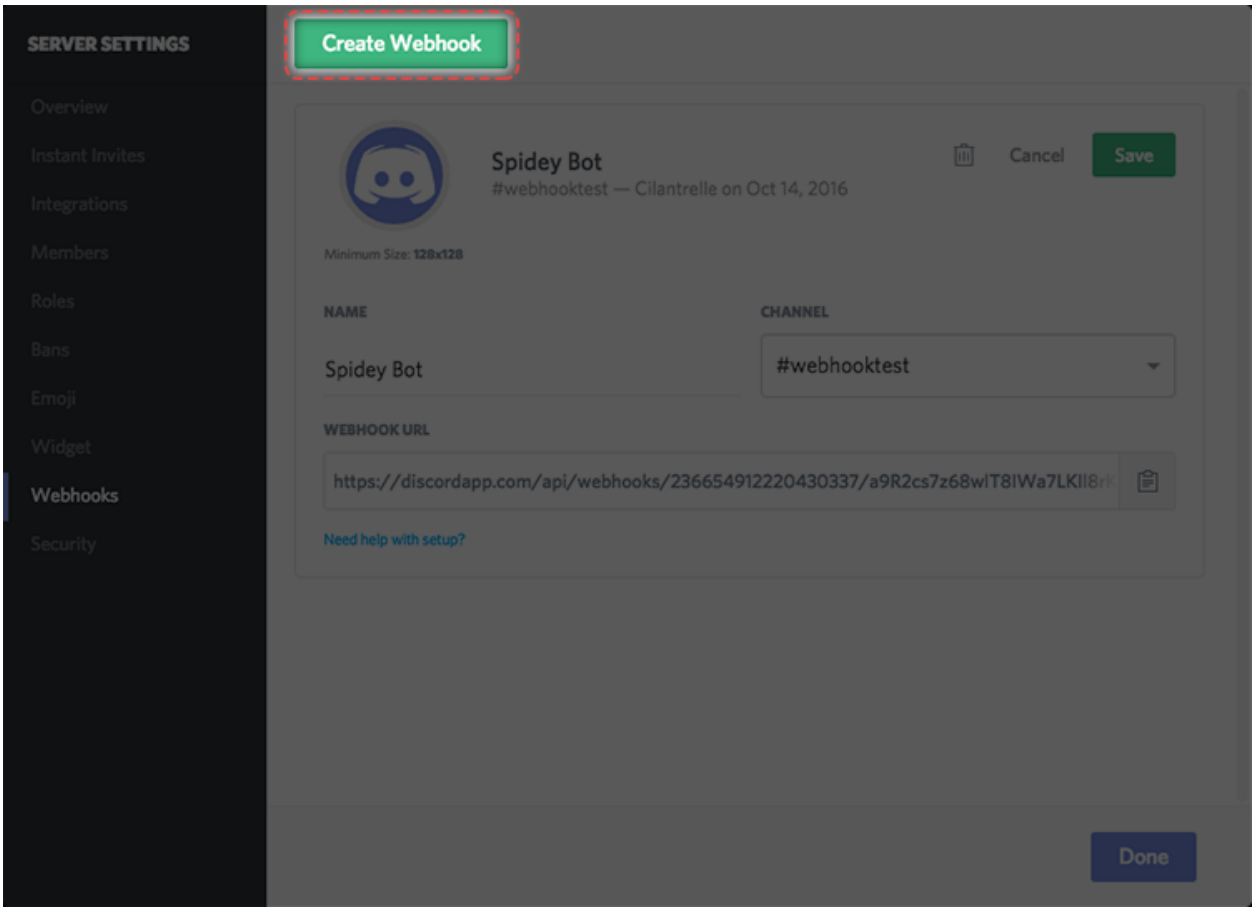
Replace `slack_webhook_here` with your slack webhook url.

4.2 Discord

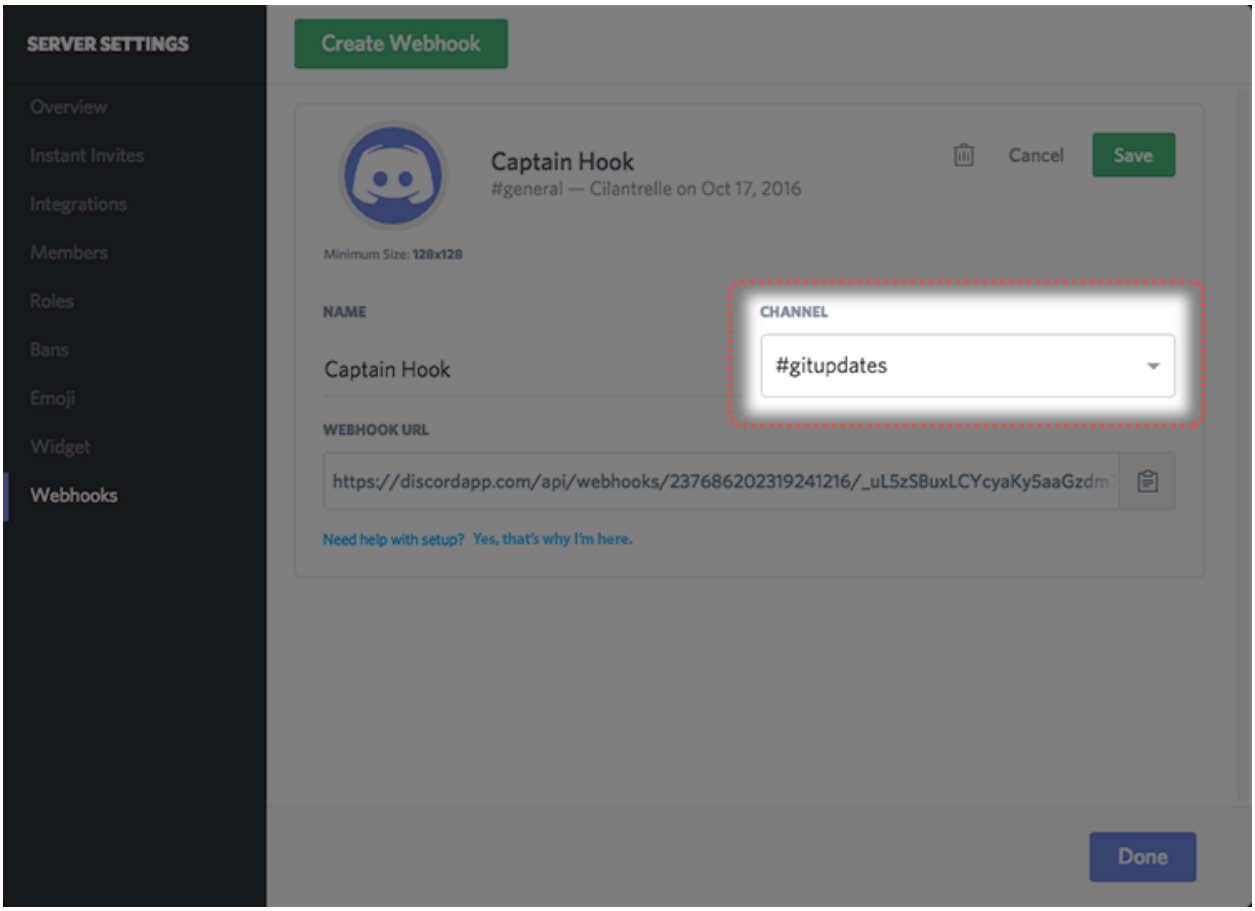
1. Open your Server Settings Webhook tab:



2. Click the green button to create a new webhook!



3. Enter the required information, for example:



Use the url in `Webhook URL` for the bot.

4.2.1 Example config

```
comms:
  discord:
    enabled: false
    hook_url: 'https://discordapp.com/api/webhooks/...../.....'
    bot_name: 'Bot'
```

4.3 Mattermost

Add the **Incoming Webhooks** integration

Use the following configuration under **Integration Settings**.

Channel: pick the channel to publish events to

4.3.1 Example config

```
comms:
  mattermost:
    enabled: false
    webhook_url: 'mattermost_webhook_url'
    channel: 'test'
    bot_name: 'Bot'
```

Replace `mattermost_webhook_url` with your mattermost webhook url.

4.4 Telegram

4.4.1 Requirements

- NewBot with ID
- Channel ID

Create BOT

For this you need to make a Bot with BotFather see BotFather. If you made the bot you can copy the id Given by botfather and put it between the quotes at token inside config.yml.

Find Channel ID

You need a channel ID of the channel you want the bot to report to, you can find a way to this at Stackoverflow.

Or if you did setup the *Telegram Service*. you can use the following command as admin: `/chatid`

If you found the ID you can copy the id and put it between the quotes at channel inside config.yml.

4.4.2 Example global config

If a service doesn't define a channel or token, then the global settings will be used. Meaning, you can have different bots for different services / channels.

```
global:
  bot_url: "https://your.bot.url"
comms:
  telegram:
    enabled: true
    token: "000000000:XXXXXXXXXXXXXXXXXXXXXXXX"
    channel: "-100000000"
```

CHAPTER 5

Webinterface

5.1 Authentication

Chat 'n' Hook uses external providers to authenticate and authorize access to the web interface. This is awesome, because users can re-use their Github / Google / Twitter / etc account when logging in.

5.1.1 Configuration

Without any auth providers the webinterface will be inaccessible. This is done because of security reasons. Leaving the interface open, unprotected gives everyone the possibility to edit your configuration. Therefore Chat 'n' Hook requires you to configure the auth before you can access the web interface.

Specifying auth providers is done in the `auth` section of the `config.yaml`. Below you can find some configuration examples for popular services.

5.1.2 Github authentication

To authorize users to Chat 'n' Hook, add their username to the `allowed_users` list.

To make use of the Github login please follow those steps:

- Visit https://github.com/settings/applications/new
- Name your application
- Make sure the redirect url is `https://<bot_domain>.com/admin/login/github/authorized`
- Click `Register application`
- Copy the `client_id` and `client_secret` to your config.yml

```
auth:
  github:
    client_id:
    client_secret:
    allowed_users:
      - brantje
      - a-github-username
```

5.1.3 Google authentication

To authorize users to Chat 'n' Hook, add their e-mail to the ''allowed_users' list.

Todo: steps to obtain `client_id` & `secret` (+ config redirect url)

```
auth:
  github:
    client_id: ''
    client_secret: ''
    allowed_users:
      - some_user@gmail.com
```

CHAPTER 6

Customization

6.1 Adding a com

tbd

6.2 Adding a event

Let's say, for example, you want to process hooks from a service called `Foobar`.

Inside the `foobar` folder, there is a folder called `events` (create if not exist). Inside it create a script with the name of an event that will be identified by the `event` property of the service class. For instance, let's say the `Foobar` service only sends one event: `bark`. So you create `bark.py` with these contents:

```
from ...base.events import BaseEvent


class BarkEvent(BaseEvent):

    def process(self):
        return {'default': str(self.event)}
```

Modify the `process` method as desired. In the event class you have access to the variables `self.event`, `self.body` and `self.request`. You need to return a dict with **comm** names as keys, like `telegram`, and the values are the messages each **comm** will publish for that event. If you want the same message for all **comms** that do not have a specific one, use the `default` key.

6.3 Adding a service

Let's say, for example, you want to process hooks from a service called `Foobar`.

1. Create a `foobar` folder inside the `services` folder

2. Inside it, create a file named `foobar.py` with the following contents:

```
from ..base import BaseService


class FoobarService(BaseService):

    @property
    def event(self):
        return self.request.headers['X-GITHUB-EVENT']
```

3. Modify the `event` property as necessary to identify the the event sent by the service. Later you're gonna create a script to handle this event. Inside this class you have access to `self.request` and `self.body` to do whatever you want.

4. Still in the `foobar` folder, create a `__init__.py` with the following contents: `from .foobar import FoobarService`

6.4 Adding telegram command

Let's say, for example, you want to add a command called `Foobar`.

Inside the `services/telegram/commands` folder, there is a folder called `custom`. Create a new folder here, called `foobar`. Inside the folder `foobar` create 2 files a script with the name of the command and a init file. So for example you create `foobar.py` with the following contents:

```
# -*- coding: utf-8 -*-
from __future__ import absolute_import
from ...base import BaseCommand


class FoobarCommand(BaseCommand):
    def get_description(self):
        return "Foo.....bar!"

    def run(self, messageObj, config):
        self.send_message(chat_id=messageObj.get('chat').get('id'),
              text='Foobar')
```

And a file called `__init__.py` wich is empty.

Each command file needs to have 2 functions: **run** and **get_description**. `run()` will be ran when the command is called and `get_description()` is called when the command `/start` is ran.

In the command class you have access to the following variables / functions:

- `messageObj` - Telegram message object, see below
- `self.send_message()`, `self.send_document()`, `self.send_photo()` are aliases to the functions in `self.telegram_bot`
- `self.telegram_bot`- is an instance of python-telegram-bot.

`messageObj` is a telegram message object, which looks like the following:

```
{
   "date":1441645532,
   "chat":{
      "last_name":"Test Lastname",
      "id":1111111,
      "type":"private",
      "first_name":"Test Firstname",
      "username":"Testusername"
   },
   "message_id":1365,
   "from":{
      "last_name":"Test Lastname",
      "id":1111111,
      "first_name":"Test Firstname",
      "username":"Testusername"
   },
   "text":"/testing 1 2 3",
   "command":"testing",
   "args":[
      "1",
      "2",
      "3"
   ]
}
```


6.5 Calling custom scripts

CHAPTER 7

Help

7.1 Troubleshooting