botbot Documentation

Release 1.0

Lincoln Loop

Mar 16, 2023

Contents

Contents:

CHAPTER 1

Installation

1.1 Pre-requisites

Some of the suggested commands that follow may require root privileges on your system.

1.1.1 Python

- Python 2.7

1.1.2 Postgresql with hStore extension

- **OS X**:
  - Homebrew: installed by default
  - Postgres.app: installed by default
- **Ubuntu**: `apt-get install postgresql-contrib-9.1 postgresql-server-dev-9.1 python-dev virtualenv`

1.1.3 Go

Version 1.2 or higher required

- **OS X**: `brew install go`
- **Ubuntu**: `apt-get install golang-go`

1.1.4 Redis

- **OS X**: `brew install redis`
- **Ubuntu**: `apt-get install redis-server`

1.2 Install

Run in a terminal:

```
virtualenv brainzbot && source brainzbot/bin/activate
pip install -e git+https://github.com/metabrainz/brainzbot-core.git#egg=brainzbot-core
cd $VIRTUAL_ENV/src/brainzbot-core

# This builds the project environment and will run for at least several minutes
make dependencies

# Adjust ``.env`` file if necessary. Defaults are chosen for local debug environments.
# If your Postgres server requires a password, you'll need to override STORAGE_URL
# The default database name is 'brainzbot'
$EDITOR .env

# Make the variables available to subprocesses
export $(cat .env | grep -v ^# | xargs)

createdb brainzbot
echo "create extension hstore" | psql brainzbot
manage.py migrate

# You'll need a staff account for creating a bot and registering channels
manage.py createsuperuser
```

Redis needs to be running prior to starting the BotBot services. For example:

```
redis-server
```

Then, to run all the services defined in `Procfile`:

```
honcho start
```

Note: foreman will also work if you have the gem or Heroku toolbelt installed.

You should now be able to access the site at `http://localhost:8000`. Log in with the username you created.

See *Getting Started* for instructions on configuring a bot.

If you plan make code changes, please read through the *Developing with BrainzBot* doc.

If you plan to run BotBot in a production environment please read the production doc.

1.3 Running Tests

The tests can currently be run with the following command:

```
manage.py test accounts bots logs plugins
```

1.4 Building Documentation

Documentation is available in `docs` and can be built into a number of formats using Sphinx:

```
pip install Sphinx
cd docs
make html
```

This creates the documentation in HTML format at `docs/_build/html`.

CHAPTER 2

Getting Started

2.1 Create a Bot and Connect to a Freenode Channel

2.1.1 Add a new Bot

A bot acts as an IRC client. It connects to one or more IRC servers and joins one or more channels per server.

1. Go to `http://localhost:8000/admin/bots/chatbot/add/`
2. Log in with your username. If you didn't create a superuser during installation, you can create one via:

```
manage.py createsuperuser
```

3. Select **'Is active'**
4. **Server:** `chat.freenode.net:6667`
5. **Nick:** You'll need to create a unique nick for your bot.

Note: Freenode registration guidelines suggest creating a separate nick if you plan to run a bot.

It isn't necessary to register your bot's nick right away if you're just trying things out. However, you'll want to choose a nick that isn't already in use. Also, be aware that many channels will require registered nick to join (the +r flag). See IRC Resources for information about valid Freenode nicks. If you're planning on running a bot for real-world or regular use, definitely register your nick. You can check if a nick is registered in your IRC client via:

```
/msg NickServ info <desired_nick>
```

6. **Real Name:** This should be a readable identifier related to your bot: a URL, project name, etc. The spec leaves a lot of room for interpretation about what this value should be. It could probably be any random value but other members of the Freenode community would likely appreciate if you use a sensible name. See **4.1.3 User message** in the IRC protocol spec for more info.

7. **Save** the Bot. Check the output in the console you started `honcho` in. You should see a number of messages indicating the bot has connected to Freenode and identified itself.

2.1.2 Add a Channel

Now that your bot is connected to a network or server, you can start having it join channels:

1. Go to `http://localhost:8000/admin/bots/channel/add/`
2. Select your bot from the dropdown
3. **Channel**: `#brainzbot`. This is a channel where we test channel bots.
4. If you'd like the channel to be listed on the site home page, Select **'Is public'**
5. Several useful plugins will already be configured. At a minimum, `ping` and `logger` will be helpful for testing the bot.
6. Save your channel. You should be directed back to the Channel list view.
7. On the Channel list view, select your channel. From the `Actions` dropdown select "Reload botbot-bot configuration" and press "Go". You should see something similar in the `honcho` console output (edited for brevity):

```
14:15:07 bot.1      | I0711 14:15:07.557470 61493 botbot.go:67] Command:  REFRESH
14:15:07 bot.1      | I0711 14:15:07.557546 61493 botbot.go:124] HandleCommand:␣
↪REFRESH
14:15:07 bot.1      | I0711 14:15:07.557557 61493 botbot.go:153] Reloading␣
↪configuration from database
14:15:07 bot.1      | I0711 14:15:07.557564 61493 network.go:49] Entering in␣
↪NetworkManager.RefreshChatbots
14:15:07 bot.1      | I0711 14:15:07.558013 61493 storage.go:121] config.Id: 1
14:15:07 bot.1      | I0711 14:15:07.558753 61493 storage.go:145] config.Channel:␣
↪[#botbot-warmup]
...
14:15:07 bot.1      | I0711 14:15:07.559072 61493 irc.go:460] [Info] The channels␣
↪the bot is connected to need to be updated
14:15:07 bot.1      | I0711 14:15:07.559083 61493 irc.go:473] [Info] Joining new␣
↪channel:  #botbot-warmup
14:15:07 bot.1      | I0711 14:15:07.559096 61493 network.go:98] Exiting␣
↪NetworkManager.RefreshChatbots
14:15:07 bot.1      | I0711 14:15:07.559111 61493 irc.go:228] [RAW thahslkd334558␣
↪on chat.freenode.net:6667 (0xc208028750) ] --> JOIN #botbot-warmup
```

8. In your IRC client, join #botbot-warmup. Try issuing a *ping* command (using your bot's nick in place of "mybot"). The bot should respond with a friendly message.
9. Go back to the home page `http://localhost:8000`, you should see the channel listed as a public channel.
10. **Add another Active Plugin** and this time select **Logger**.
11. **Save** and "Reload botbot-bot configuration" as before. Your `honcho` console should once again show a refresh
12. In your IRC client, go to #botbot-warmup and post a message. You should now have a log available at `http://localhost:8000/freenode/botbot-warmup`. Each message you post in the channel shows up in the `honcho` console.

You're ready to configure your own channels and utilize other plugins.

CHAPTER 3

Managing Bots

3.1 Multiple Bots

Bots can be connected to multiple networks. For example you can have a bot that connects to Freenode, and another that connects to Mozilla's IRC network.

Multiple bots for the same network / server is not supported at this time.

3.2 Public, Private, and Featured Channels

These are primarily distinctions for the Django site and the display of logs.

Public Logs for public channels will be available on a public URL like *http://example.com/freenode/django*

Featured Featured channels are public channels. Used by chatlogs.metabrainz.org for highlighting some public channels. May be deprecated in the future.

Private Logs for private channels are only availale to authenticated users of the site. They will have URLs that are not easy to guess.

3.3 Freenode

3.3.1 Policy

Before logging any public channels, take a couple simple steps to ensure no misunderstandings occur.

1. Have the consent of a channel operator.
2. Ask the operator to make it clear in the channel topic that it is being logged.

Freenode's channel guidelines don't seem to address non-operator users who want to run bots. (see final bullet point) Our preference is to favor honesty and transparency.

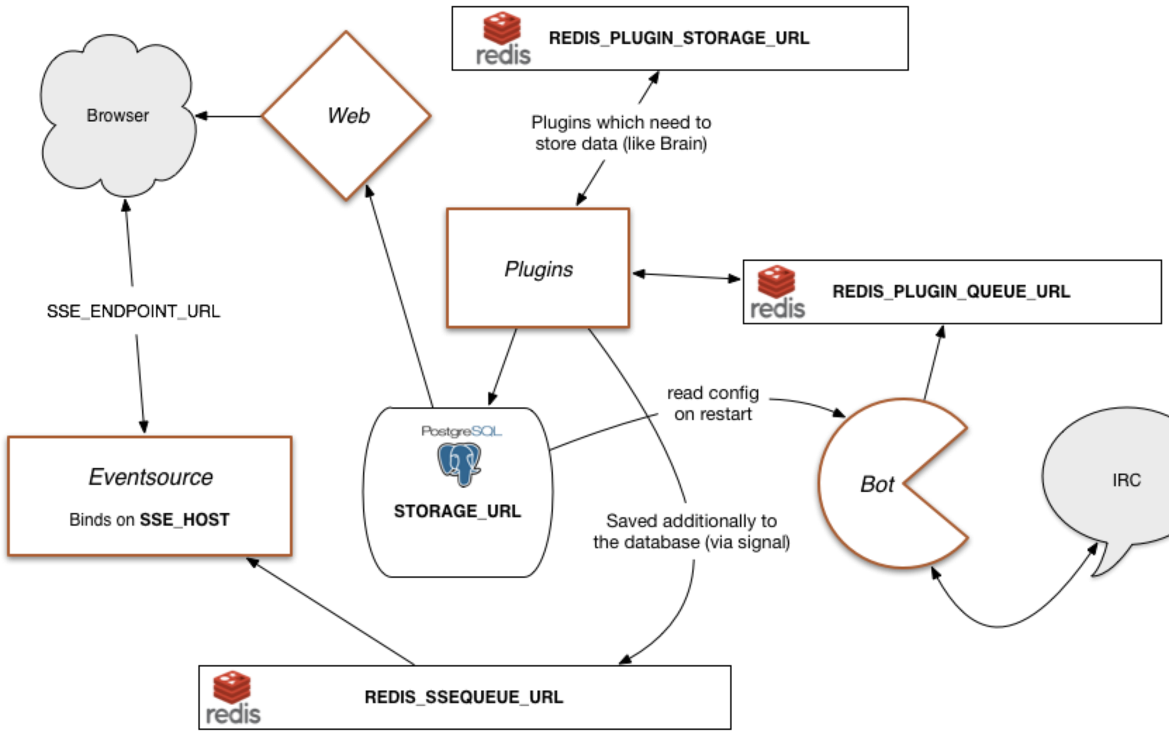
CHAPTER 4

Developing with BrainzBot

4.1 Architecture

Several loosely coupled pieces make up the whole of BotBot:

1. **brainzbot-bot:** An IRC client capable of connecting to multiple IRC networks, and connecting multiple channels and nicks per network. (Go)
2. **brainzbot-plugins:** A plugin framework - plugins receive messages from IRC channels and can respond within the channel. (Python)
3. **brainzbot-core:** A web site for managing bots/channels as well as a beautiful public interface for channel logs. (Python/Django)
4. **nginx + push-stream-module:** An SSE provider that the web site can connect to for real-time logs.



4.1.1 Configuration / Environment Variables

At Lincoln Loop we've found the 12-factor style to be helpful and use those guidelines. As recommended, we rely on environment variables whenever possible for determining site-specific configuration values.

The .env file

We've adopted a convention used by Heroku and others of creating a `.env` file in the project root that can be used as a base for managing environment variables. Tools like Honcho and Foreman will use the `.env` file to bootstrap the environment for each process it starts up. If you prefer to use these tools, all you have to worry about is editing the `.env` file as needed. The `.env` file can also be used as a template for other methods of populating environment variables.

We don't want make assumptions about how developers prefer to configure their environments. There are some handy open source libs out there that we could use to easily make each service utilize the `.env` file, but this won't work well for everyone or every environment. If you need to run a service individually, you'll need to make sure the proper environment variables are configured. The next section describes some easy ways to handle this.

4.1.2 Running Services

Run All Services

Honcho is great for getting everything started quickly. Running this command will load all services defined in the `Procfile`:

```
honcho start
```

Run Services Individually

When working on code changes or debugging, it is often desirable to manage one or more of the services (bot, plugin runner, runserver, nginx) independently.

If you'd like to make use of the `.env` file, you can still start services individually using Honcho:

```
honcho start core  # Starts Django site
honcho start bot  # Starts IRC client
honcho start plugins  # Starts plugin runner
```

If you would prefer not to use Honcho, you'll need to manage the environment variables. Many developers use the virtualenv `postactivate` feature to set required environment variables whenever a virtualenv is activated. An alternative approach could be to attempt to set variables directly from the `.env` file. As an example, you could put the following into a `set_env.sh` file:

```
export $(cat .env | grep -v ^# | xargs)
```

Then you could invoke commands and individual services like:

```
source set_env.sh && manage.py runserver
source set_env.sh && botbot-bot -v=2 -logtostderr=true
nginx -c `pwd`/nginx.conf.example
source set_env.sh && manage.py run_plugins
```

If you've explicitly set the environment through your own methods, services can be invoked like usual:

```
manage.py runserver
brainzbot-bot
botbot-eventsource
manage.py run_plugins
```

Go IRC Client (bot)

Execution starts in main.go, in function "main". That starts the chatbots (via NetworkManager), the goroutine which listens for commands from Redis, and the mainLoop goroutine, then waits for a Ctrl-C or kill to quit.

The core of the bot is in mainLoop (main.go). That listens to two Go channels, fromServer and fromBus. fromServer receives everything coming in from IRC. fromBus receives commands from the plugins, sent via a Redis list.

A typical incoming request to a plugin would take this path:

> IRC -> TCP socket -> ChatBot.listen (irc.go) -> fromServer channel -> mainLoop (main.go) -> Dispatcher (dispatch.go) -> redis PUBLISH -> plugin

A reply from the plugin takes this path:

> plugin -> redis LPUSH -> listenCmd (main.go) -> fromBus channel -> mainLoop (main.go) -> Network-Manager.Send (network.go) -> ChatBot.Send (irc.go) -> TCP socket -> IRC

And now, in ASCII art:

```
plugins <--> REDIS -BLPOP-> listenCmd (main.go) --> fromBus --> mainLoop (main.go) <--
↪ fromServer <-- n ChatBots (irc.go) <--> IRC
               ^                                                  | |                ␣
↪                      ^
```

(continues on next page)

(continued from previous page)

```
                | PUBLISH                                                   | |                    ␣
↪                        |
                ------------ Dispatcher (dispatch.go) <----------    ---->␣
↪NetworkManager (network.go) ----
```

Django Site

You can run commands within the Honcho environment using the `run` command:

```
honcho run manage.py dbshell
honcho run manage.py syncdb
```

If you're using the `set_env` method:

```
source set_env.sh && manage.py dbshell
source set_env.sh && manage.py syncdb
```

If you've explicitly set the environment variables, run commands like usual:

```
manage.py dbshell
manage.py syncdb
```

Working with LESS

LESS requires Node.js. There are shortcuts in the Makefile for installing everything necessary:

```
make less-install
```

From this point forward, if you need to compile LESS run:

```
make less-compile
```

To automatically compile whenever you save a change:

```
make less-watch
```

4.1.3 Plugins

You can optionally run the plugins under gevent (`pip install gevent`) which will parallelize them when running the plugins under load:

```
manage.py run_plugins --with-gevent
```

CHAPTER 5

Plugin API Documentation

You can write your own Botbot plugin by extending the core plugin class and providing one or more message handlers. A message handler is a method on the plugin class that receives an object representing a user message that has been posted to the IRC channel the plugin is associated with. The existing plugins in `botbotme_plugins/plugins` serve as good examples to follow. **ping** and **brain** are good ones to start with due to their simplicity.

5.1 Plugin Capabilities

Plugins provide three basic capabilities:

1. Parse messages and optionally respond with an output message.
2. Associate configuration variables. Useful if your plugin needs to connect to external services.
3. Store and retrieve key/value pairs.

All plugins extend the BasePlugin class, providing them with the ability to utilize these capabilities.

5.2 Parsing and responding to messages

In the simplest case, a plugin will receive a message from an IRC channel and parse it based on a rule. When the parsed input matches a rule, the plugin may return a response.

Additional methods should be defined on your `Plugin` class that will listen and optionally respond to incoming messages. They are registered with the app using one of the following decorators from `botbotme_plugins.decorators`:

- `listens_to_mentions(regex)`: A method that should be called only when the bot's nick prefixes the message and that message matches the regex pattern. For example, `[o__o]: What time is it in Napier, New Zealand?`. The nick will be stripped prior to regex matching.
- `listens_to_all(regex)`: A method that should be called on any line that matches the regex pattern.

- `listens_to_command(cmd)`: A method that should be called on any line that starts with the command prefix, followed by `cmd`. All further arguments are passed through in a list. For example, `!list ops`.
- `listens_to_regex_command(cmd, regex)`: `listens_to_command`, with a regex check on all arguments. For a good example of this, see the `metabrain` plugin, which can do things like `!remember jeff bezos=filthy rich` by matching against `some_key=some_value`.

The method should accept a `line` object as its first argument and any named matches from the regex as keyword args. Any text returned by the method will be echoed back to the channel.

The `line` object has the following attributes:

- `user`: The nick of the user who wrote the message
- `text`: The text of the message (stripped of nick if addressed to the bot)
- `full_text`: The text of the message

5.3 Configuration Metadata

Metadata can be associated with your plugin that can be referenced as needed in the message handlers. A common use case for this is storing authentication credentials and/or API endpoint locations for external services. The `github` plugin is an example that uses configuration for the ability to query a Github repository.

To add configuration to your plugin, define a config class that inherits from `config.BaseConfig`. Configuration values are declared by adding instances of `config.Field` as attributes of the class. You can validate that all required fields have a value using the `is_valid` method.

Once your config class is defined, you associate it with the plugin via the `config_class` attribute:

```python
class MyConfig(BaseConfig):
    unwarranted_comments = Field(
        required=False,
        help_text="Responds to every message with sarcastic comment",
        default=True)

class Plugin(BasePlugin):
    config_class = MyConfig

    @listens_to_all
    def peanut_gallery(self, line):
        if self.config.unwarranted_comments:
            return "Good one!"
```

5.4 Storage / Persisting Data

BasePlugin provides a simple wrapper around the Redis API that Plugins should use for storage - it'll handle namespacing the key in the format `<bot_id>:<channel_id>:<plugin_slug>:<key_name_stripped>`. This ensures that there are no collisions. This also means plugins can't access data from other plugins or other channels.

There are four methods:

- `store(key, value)`: Sets `key` to `value`. Importantly, `value` is encoded in `utf-8` before being stored.
- `retrieve(key)`: Gets the value corresponding to `key`. Returns a `utf-8`-encoded string.
- `delete(key)`: Removes the stored `key`.

- `incr(key)`: Increments the counter specified by `key`. This is really just a special-case version of `set`, but the counter is set to `0` if it does not exist, and there's no need to `retrieve` the existing value beforehand.

5.5 Testing Your Plugins

In order to simulate the plugin running in its normal environment, an app instance must be instantiated. See the current tests for examples. This may change with subsequent releases.

CHAPTER 6

IRC Resources

Internet Relay Chat Protocol

6.1 Freenode

- Registering Channels
- FAQ

6.1.1 Being a Good Citizen

- Channel Guidelines
- Being a Catalyst
- Policy

6.1.2 Nicks

It was surprisingly difficult to find a desciption of what constitutes a valid nick on Freenode. Asked the friendly folks on #freenode and had the answer a few minutes later.

> nickname = (letter / special) *8(letter / digit / special / “-”) and special is “[”, “]”, “”, ““”, “_”, “^”, “{”, “|”, “}”

Nicks can be from 2 - 16 ASCII characters long. These characters are allowed:

RFC 1459 - Internet Relay Chat Protocol http://www.ietf.org/rfc/rfc1459.txt

CHAPTER 7

Troubleshooting

7.1 Log messages

plugins.1 | DoesNotExist: Channel matching query does not exist.

If you edit or add channels this condition can occur. It is due to a bug where stale config data is in Redis. This bug will be resolved in a future release.

CHAPTER 8

Indices and tables

- genindex
- modindex
- search