Sputnik Documentation

Release 0.0.0

Kevin Fung

December 04, 2014

CHAPTER 1

Summary

Sputnik is a Python IRC bouncer written using asyncio and backed by Redis. It is intended as a lightweight, zero-configuration bouncer capable of deployment on cloud providers such as Heroku. Sputnik is written in pure Python, so adding custom functionality is relatively straightforward.

Features

- Automatic Network Reconnection
- Channel Saver
- Buffered Message History
- Multi-Client Connections

CHAPTER 2

Getting Started

You can easily deploy a Sputnik instance on Heroku using the button below.

Alternately, you can manually create and deploy your own Heroku app, or run Sputnik on your own computer or server. To do so requires a Python 3.4 interpreter and Redis (optional), if you want persistence between restarts or crashes.

CHAPTER 3

Documentation

Sputnik documentation is built using Sphinx and publicly hosted at http://sputnik.readthedocs.org/. You can also build and serve the documentation locally.

```
git submodule update --init --recursive
cd docs && make dirhtml && cd _build/dirhtml
python -m SimpleHTTPServer
```

Then visit http://localhost:8000 in a browser.

CHAPTER 4

License

CHAPTER 5

Modules

5.1 bouncer module

Sputnik Bouncer Implementation

This module provides the Sputnik Bouncer implementation. As the primary entry point, the Bouncer is responsible for bootstrapping the entire program.

class `bouncer.`**`Bouncer`**

Bases: `object`

A singleton that manages connected devices.

The Bouncer provides the base functionality needed to instantiate a new Client or Network. It also acts as a bridge between connected Clients and Networks by maintaining an authoritative record of each connected device.

`clients`

set of sputnik.Client

A set of connected Clients.

`datastore`

sputnik.Datastore

A Redis interface.

`networks`

dict of sputnik.Network

A dictionary of connected Networks.

`add_network`(*network*, *hostname*, *port*, *nickname*, *username*, *realname*, *password=None*, *usermode=0*)

Connects the Bouncer to an IRC network.

This forms the credentials into a dictionary. It then registers the network in the datastore, and connects to the indicated IRC network.

Parameters

- **`network`** (*str*) – The name of the IRC network to connect to.
- **`hostname`** (*str*) – The hostname of the IRC network to connect to.
- **`port`** (*int*) – The port to connect using.
- **`nickname`** (*str*) – The IRC nickname to use when connecting.

- **`username`** (*str*) – The IRC ident to use when connecting.
- **`realname`** (*str*) – The real name of the user.
- **`password`** (*str, optional*) – Bouncer password. Defaults to `None`.
- **`usermode`** (*int, optional*) – The IRC usermode. Defaults to `0`.

`remove_network` (*network*)

Removes a network from the Bouncer.

This disconnects the Bouncer from the indicated network and unregisters the network from the datastore.

Parameters **`network`** (*str*) – the name of a network.

`start` (*hostname='', port=6667*)

Starts the IRC and HTTP listen servers.

This creates the IRC server-portion of the Bouncer, allowing it to accept connections from IRC clients. It also starts the HTTP server, enabling browsers to connect to the web interface.

Note: This is a blocking call.

Parameters

- **`hostname`** (*str, optional*) – Hostname to use. Defaults to `""`.
- **`port`** (*int, optional*) – The port to listen on. Defaults to 6667.

5.2 client module

Sputnik Client Implementation

This module provides the Sputnik Client implementation. This is a subclass of a Connection, and defines an interface to IRC client applications implementing _RFC 2812: https://tools.ietf.org/html/rfc2812 .

class `client.`**`Client`** (*bouncer*)

Bases: `connection.Connection`

An instance of a connection from an IRC client.

A Client is the product of an asyncio protocol factory, and represents an instance of a connection from an IRC client to the listen server. It does not implement an actual IRC client, as defined in _RFC 2812: https://tools.ietf.org/html/rfc2812 .

`bouncer`

sputnik.Bouncer

A reference to the Bouncer singleton.

`broker`

sputnik.Network

The connected Network instance.

`network`

str

The name of the IRC network to connect to.

ready
bool

Indicates if the Client has connected to a Network.

connection_lost(*exc*)
Unregister the connected Client from the Bouncer.

Removes the Client from the set of connected Clients in the Bouncer before the connection is terminated. After this point, there should be no remaining references to this instance of the Client.

connection_made(*transport*)
Registers the connected Client with the Bouncer.

Adds the Client to the set of connected Clients in the Bouncer and saves the transport for later use.

data_received(*data*)
Handles incoming messages from connected IRC clients.

Messages coming from IRC clients are potentially batched, and need to be parsed into individual lines before any other operation may occur. Afterwards, we split lines according to the IRC message format and then perform actions as appropriate.

forward(**args*)
Writes a message to the Network.

Because the Client represents an instance of a connection from an IRC client, we instead need to write to the transport associated with the connected network.

Parameters **args** (*list of str*) – A list of strings to concatenate.

5.3 connection module

Sputnik Connection Implementation

This module provides the Sputnik Connection implementation. This is a base class that defines several helper functions for common tasks related to transport-level interactions, such as message encoding and message passing.

class `connection.`**Connection**
Bases: `asyncio.protocols.Protocol`

A generic instance of a network connection.

A Connection is a base class that represents an instance of a network connection. The Connection implements commonly used actions that may be performed on messages.

decode(*line*)
Attempts to decode a line as UTF-8, with fallback to Latin-1.

We try to maintain a full-Unicode presence where possible. However, not all IRC servers are encoding using UTF-8, so we shadow *str.decode()* and provide a fallback to Latin-1 when needed.

Parameters **line** (*str*) – A byte-string message to decode.

Returns A decoded message.

Return type str

normalize(*line*, *ending='\r\n'*)
Ensures that a line is terminated with the correct line endings.

The IRC protocol specifies that line endings should use CRLF line endings. This ensures that lines conform to this standard. In the event of a server that does not conform to the specification, we preserve the ability to provide an alternative line ending character sequence.

Args: line (str): A message to be sent to the IRC network. ending (str, optional): The line ending. Defaults to ““

““.

send(**args*)

Writes a message to the connected interface.

Messages are typically of the form `<command> <message>`. This encapsulates the IRC messaging protocol by concatenating messages and checking their line endings before encoding the message into raw bytes, as part of the asyncio transport mechanism.

Parameters **args** (*list of str*) – A list of strings to concatenate.

5.4 datastore module

Sputnik Datastore Implementation

This module provides the Sputnik Datastore implementation. It implements a thin wrapper around Redis, which is required in order to persist data across Bouncer restarts or network disconnections. This functionality is required due to the ephemeral filesystems typical to most Platform-as-a-Service Providers (PaaS).

class `datastore.`**Datastore**(*hostname*, *port*)

Bases: `object`

A singleton that provides a thin wrapper to Redis.

The Datastore is responsible for persisting networks and channels in the event of an unexpected crash by either the Bouncer or a connected network. It also holds persistent, shared variables, such as the Bouncer password.

database

redis.Redis

A Redis database connection.

add_channel(*network*, *channel*, *password=’‘*)

Adds a channel to the Redis.

Parameters

- **network** (*str*) – The name of a network.
- **channel** (*str*) – The name of a channel.
- **password** (*str, optional*) – The channel password. Defaults to `""`.

add_network(*network*, *hostname*, *port*, *nickname*, *username*, *realname*, *password=None*, *usermode=0*)

Adds a network to the Redis instance.

Parameters

- **network** (*str*) – The name of the IRC network to connect to.
- **hostname** (*str*) – The hostname of the IRC network to connect to.
- **port** (*int*) – The port to connect using.
- **nickname** (*str*) – The IRC nickname to use when connecting.

- **username** (*str*) – The IRC ident to use when connecting.
- **realname** (*str*) – The real name of the user.
- **password** (*str, optional*) – Bouncer password. Defaults to `None`.
- **usermode** (*int, optional*) – The IRC usermode. Defaults to `0`.

check_password(*password_attempt*)
Checks a password attempt against the Bouncer password.

Parameters **password_attempt** (*str*) – The password attempt.

Returns Whether the password matched.

Return type bool

get_channels(*network=''*)
Retrieves all connected channels from Redis.

This gets credentials for all connected channels, where credentials are of the form *{ "<network/channel>" : "<password>" }*. If the network argument is specified, then the output is filtered to only include channels from the indicated network.

Parameters **network** (*str, optional*) – The name of a network. Defaults to `""`.

Returns A dictionary of channel credentials.

Return type dict

get_networks()
Retrieves all connected networks from Redis.

This gets credentials for all connected networks, where credentials contain all values necessary to reconstruct a network connection, where networks are of the form *{ "<network_name>" : "<credentials>" }*.

Returns A dictionary of network credentials.

Return type dict

get_password()
Retrieves the Bouncer password from Redis.

Returns The encrypted Bouncer password.

Return type str

remove_channel(*network*, *channel*)
Removes a channel from Redis.

Parameters

- **network** (*str*) – The name of a network.
- **channel** (*str*) – The name of a channel.

remove_network(*network*, *hard=True*)
Removes a network from Redis.

Parameters

- **network** (*str*) – The name of a network to remove.
- **hard** (*bool*) – When True, clears all associated channels.

set_password(*password='cosmonaut'*)
Saves a new Bouncer password to Redis.

Parameters **`password`** (*str, optional*) – The new password for the Bouncer.

5.5 handlers module

Sputnik Request Handlers

This module provides Tornado Request Handlers for the Sputnik Web Interface.

class `handlers.`**`AddHandler`**(*application*, *request*, ***kwargs*)

Bases: `handlers.BaseHandler`

The RequestHandler that serves the add network page.

The add network page uses a form to receive new network settings. If a network already exists using the provided name, the network is not added.

`get`()

Renders the add network page.

The add network page provides a form for adding a new network, complete with placeholder settings.

`post`()

Handles add network requests.

If a network already exists using the provided name, the network is not added.

class `handlers.`**`BaseHandler`**(*application*, *request*, ***kwargs*)

Bases: `tornado.web.RequestHandler`

A base handler that stores the Bouncer singleton.

`get_current_user`()

`initialize`(*bouncer*)

Initializes the RequestHandler and stores the Bouncer.

Parameters **`bouncer`** (*sputnik.Bouncer*) – The singleton Bouncer instance.

class `handlers.`**`DeleteHandler`**(*application*, *request*, ***kwargs*)

Bases: `handlers.BaseHandler`

The RequestHandler that handles delete network requests.

`get`(*network_name*)

Handles delete network requests.

Parameters **`network_name`** (*str*) – Network name of the network to delete.

class `handlers.`**`EditHandler`**(*application*, *request*, ***kwargs*)

Bases: `handlers.BaseHandler`

The RequestHandler that serves the edit network page.

The edit network page uses a form to receive updated settings from users. When a network is editted, it is disconnected and then recreated using the new settings.

`get`(*network_name*)

Renders the edit network page.

The edit network page shows current settings for a network and provides a form for submitting changes to that network.

Parameters **`network_name`** (*str*) – Network name of the network to edit.

post(*network_name*)
Handles edit network requests.

The existing network is disconnected and a new connection is started using the new settings.

Parameters **network_name** (*str*) – Network name of the network to edit.

class handlers.**LoginHandler**(*application*, *request*, ***kwargs*)
Bases: `handlers.BaseHandler`

The RequestHandler that serves the login page.

The login page prompts the user for their password and authenticates them when the password matches the one stored by the bouncer in its database.

get()
Renders the login page.

The login page uses a form to ask the user for their password.

post()
Handles login requests.

Checks the password against the stored password and authenticates.

class handlers.**LogoutHandler**(*application*, *request*, ***kwargs*)
Bases: `handlers.BaseHandler`

The RequestHandler that handles log out requests.

Redirects to the homepage after clearing authentication.

get()
Handles log out requests.

Redirects to the homepage after clearing authentication.

class handlers.**MainHandler**(*application*, *request*, ***kwargs*)
Bases: `handlers.BaseHandler`

The main RequestHandler that serves the home page.

The home page displays the current list of networks.

get()
Renders the home page.

The home page displays the current list of networks.

class handlers.**SettingsHandler**(*application*, *request*, ***kwargs*)
Bases: `handlers.BaseHandler`

The RequestHandler that serves the settings page.

Allows users to change their password.

get()
Renders the settings page.

The settings page uses a form to allow users to change their password.

post()
Handles settings requests.

Change password requests require the current password to match and two entries of the new password to match.

5.6 network module

Sputnik Network Implementation

This module provides the Sputnik Network implementation. This is a subclass of a Connection, and defines an interface to IRC server networks implementing _RFC 2813: https://tools.ietf.org/html/rfc2813 .

class `network.`**`Network`**(*bouncer*, *network*, *hostname*, *port*, *nickname*, *username*, *realname*, *password=None*, *usermode=0*)

Bases: `connection.Connection`

An instance of a connection to an IRC network.

A Network is the product of an asyncio protocol factory, and represents an instance of a connection from an IRC client to an IRC server. This could be either a single IRC server, or more likely, a network of servers behind a load balancer. It does not implement an actual IRC server, as defined in

`??? (revisit this later)`

`attempt_reconnect`(*attempt=0*, *retries=5*)

Attempts to reconnect to a network that unexpectedly disconnected.

This is only called if we drop the connection to a network and the connected flag is set, to distinguish from intentional disconnects.

Parameters

- **`attempt`** (*int*) – The current attempt count.
- **`retries`** (*int*) – The number of times to attempt to reconnect.

`connection_lost`(*exc*)

Unregisters the connected Network from the Bouncer.

Removes the Network from the dictionary of connected Clients in the Bouncer before the connection is terminated. After this point, there should be no remaining references to this instance of the Network.

`connection_made`(*transport*)

Registers the connected Network with the Bouncer.

Adds the Network to the set of connected Networks in the Bouncer and saves the transport for later use. This also creates a collection of buffers and logging facilities, and initiates the authentication handshake, if applicable.

`data_received`(*data*)

Handles incoming messages from connected IRC networks.

Messages coming from IRC networks are potentially batched and need to be parsed into individual lines before any other operation may occur. On certain occasions, incoming data may overflow the transport buffer, requiring additional logic to reconstitute the messages into a single stream. Afterwards, we split lines according to the IRC message format and perform actions as appropriate.

`forward`(**args*)

Writes a message to all connected CLients.

Because the Network represents an instance of a connection to an IRC network, we instead need to write to the transports of all clients.

Parameters **`args`** (*list of str*) – A list of strings to concatenate.

5.7 server module

Sputnik HTTPServer Implementation

This module provides the Sputnik HTTPServer implementation. It is responsible for serving the web interface, and interfaces with the Bouncer singleton to connect to and disconnect from networks.

class `server.`**`HTTPServer`**`(`*bouncer*`)`

Bases: `tornado.web.Application`

An Asynchronous HTTP Server that diplays the frontend.

The HTTPServer renders the frontend and accepts commands used to control the Bouncer singleton. For development purposes, it may be helpful to set the DEBUG environment variable. e.g. *export DEBUG=True*

`start``(`*port=8080*`)`

Starts the HTTP listen server.

This loads the Tornado HTTPServer on the specified port.

Parameters **`port`** (*int, optional*) – The port to listen on. Defaults to 8080

CHAPTER 6

Indices and Tables

- *genindex*
- *modindex*

Index