discordproxy

Erik Kalkoken

Aug 01, 2023

CONTENTS:

CHAPTER

ONE

OPERATIONS GUIDE

1.1 Installation

1.1.1 Alliance Auth installation

This section describes how to install Discord Proxy into an existing Alliance Auth installation.

Install Discord Proxy

Note: This guide assumed a default installation according to the official Auth installation guide.

Login as root user, activate your venv and navigate to your Auth main folder:

```
cd /home/allianceserver/myauth
```

Install discordproxy from PyPI into the venv:

```
pip install discordproxy
```

Add Discord Proxy to your supervisor configuration for Auth.

Edit supervisor.conf in your current folder and add the below section. Make sure to replace `YOUR-BOT-TOKEN` with your current Discord bot token:

```
[program:discordproxy]
command=/home/allianceserver/venv/auth/bin/discordproxyserver --token "YOUR-BOT-TOKEN"
directory=/home/allianceserver/myauth/log
user=allianceserver
numprocs=1
autostart=true
autorestart=true
stopwaitsecs=120
stdout_logfile=/home/allianceserver/myauth/log/discordproxyserver.out
stderr_logfile=/home/allianceserver/myauth/log/discordproxyserver.err
```

Note: We do not recommend adding discordproxy to your myauth group, since it does not require to be restarted after myauth configuration changes like the other programs in that group.

Reload supervisor to activate the changes and start Discord Proxy:

```
supervisorctl reload
```

To verify Discord Proxy is up and running you can check it's status:

```
supervisorctl status discordproxy
```

It should say "RUNNING".

To verify your installation was successful we recommend to *test your server*.

1.1.2 Stand-alone installation

This section describes how to install Discord Proxy as standalone server.

Create a Discord bot account

Follow this guide to create your Discord bot:

1. Create a Discord application for your bot
2. Invite your bot to your Discord server

Install discordproxy on your server

Create a service user and switch to that user:

```
sudo adduser --disabled-login discordproxy
sudo su discordproxy
```

Setup a virtual environment for the server, activate it and update key packages:

```
cd /home/discordproxy
python3 -m venv venv
source venv/bin/activate
```

Update and install basic packages:

```
pip install -U pip
pip install wheel setuptools
```

Install discordproxy from PyPI into the venv:

```
pip install discordproxy
```

Add discordproxy to supervisor

Create a supervisor configuration file - /home/discordproxy/discordproxyserver.conf - with the below template:

```
[program:discordproxy]
command=/home/discordproxy/venv/bin/discordproxyserver --token "YOUR-BOT-TOKEN"
directory=/home/discordproxy
user=discordproxy
numprocs=1
autostart=true
autorestart=true
stopwaitsecs=120
stdout_logfile=/home/discordproxy/discordproxyserver.out
stderr_logfile=/home/discordproxy/discordproxyserver.err
```

Add discordproxy to your supervisor configuration and restart supervisor to activate the change:

```
ln -s /home/discordproxy/discordproxyserver.conf /etc/supervisor/conf.d
supervisorctl reload
```

To verify your installation was successful we recommend to *test your server*.

1.1.3 Test Discord Proxy server

To verify your Discord Proxy server is up and running send a direct message to yourself with the CLI tool (replace the number below with your own user ID):

```
discordproxymessage direct 12345678 "test"
```

Hint: Here is how you can find IDs on your Discord server: Where can I find my User/Server/Message ID?

1.2 Server configuration

Discord Proxy is designed to run via supervisor and can be configured with the below arguments. It comes with sensible defaults and will in most cases only need the Discord bot token to operate.

To configure your server just add/modify one of the below parameters in the respective command line of your supervisor configuration:

```
usage: discordproxyserver [-h] [--token TOKEN] [--host HOST] [--port PORT]
                          [--log-console-level {DEBUG,INFO,WARN,ERROR,CRITICAL}]
                          [--log-file-level {DEBUG,INFO,WARN,ERROR,CRITICAL}]
                          [--log-file-path LOG_FILE_PATH] [--version]

Server with HTTP API for sending messages to Discord

optional arguments:
  -h, --help            show this help message and exit
  --token TOKEN         Discord bot token. Can alternatively be specified as
```

(continues on next page)

(continued from previous page)

```
                        environment variable DISCORD_BOT_TOKEN. (default:
                        None)
  --host HOST           server host address (default: 127.0.0.1)
  --port PORT           server port (default: 50051)
  --log-console-level {DEBUG,INFO,WARN,ERROR,CRITICAL}
                        Log level of log file (default: CRITICAL)
  --log-file-level {DEBUG,INFO,WARN,ERROR,CRITICAL}
                        Log level of log file (default: INFO)
  --log-file-path LOG_FILE_PATH
                        Path for storing the log file. If no path if provided,
                        the log file will be stored in the current working
                        folder (default: None)
  --version             show the program version and exit
```

1.3 Tools

Discord Proxy comes with a simple tool for sending messages to your Discord server from the command line. The main purpose of this tool is to check that the server is functioning correctly.

Here is how to send a direct message to a user:

```
discordproxymessage direct 12345678 "hi!"
```

The number is the user ID of the user to sent a message to. For more information run the command with the `-h` option.

CHAPTER

TWO

DEVELOPER GUIDE

There are two different approaches on how to interact with Discord via Discord Proxy:

- DiscordClient
- gRPC

2.1 DiscordClient

`DiscordClient` is a class provided by Discord Proxy which aims to make it easy to interact with Discord in your code. Most of the complexity of the underlying gRPC protocol is hidden behind a simple API.

See also:

For the documentation of the DiscordClient class see: *Client*

2.1.1 Sending a message

Here is a simple example for sending a direct message to a user:

```
from discordproxy.client import DiscordClient

client = DiscordClient()
client.create_direct_message(user_id=123456789, content="Hello, world!")
```

Hint: To test this script please replace the user ID with your own. Here is how you can find IDs on your Discord server: Where can I find my User/Server/Message ID?

2.1.2 Sending a message with an embed

Messages often include attachments called embeds. You can construct your own embeds with the provided `Embed` class.

```
from discordproxy.client import DiscordClient
from discordproxy.discord_api_pb2 import Embed

client = DiscordClient()
embed = Embed(description="Hello, world!")
client.create_direct_message(user_id=123456789, embed=embed)
```

See also:

All common Discord objects are defined as Protobuf classes and can be found here: *Protocol Documentation*. These classes are used for creating objects for to methods of the *DiscordClient* and are returned by them.

2.1.3 Error handling

There are two classes of errors which can occur:

First, errors from the Discord API, e.g. that a user can not be found. These errors will generate a `DiscordProxyHttpError` exception.

Second, errors from the network connection with the Discord Proxy server, e.g. when the server is not up or a timeout occurs. These errors will generate a `DiscordProxyGrpcError` exception.

Both exceptions objects include additional details, e.g. the HTTP error code or the Discord specific JSON error code. Both exceptions are also inherited from `DiscordProxyException`, so for a very simple error handling you can just catch the top level exception.

Here is the same example from before, but now with some rudimentary error handling:

```python
from discordproxy.client import DiscordClient
from discordproxy.exceptions import DiscordProxyException

client = DiscordClient()
try:
    client.create_direct_message(user_id=123456789, content="Hello, world!")
except DiscordProxyException as ex:
    print(f"An error occured when trying to create a message: {ex}")
```

2.1.4 Timeouts

All requests are synchronous and will usually complete almost instantly. However, sometimes it can take a few seconds longer for a request to complete due to Discord rate limiting, especially if you are running multiple request to Discord in parallel. There also might be issues with the network or the Discord API, which might case requests to go on for a long time (the hard timeout on the client side is about 30 minutes). In order to build a robust application we recommend to use sensible timeouts with all requests. Note that this timeout must cover the complete duration it takes for a request to compete and should therefore not be set too short.

You can define custom timeouts with when instanciating your client. Further, when a timeout occures the specical exception `DiscordProxyTimeoutError` will be raised. Here is an example:

```python
from discordproxy.client import DiscordClient, DiscordProxyTimeoutError

client = DiscordClient(timeout=30)  # Defines a timeout of 30 seconds for all methods
try:
    client.create_direct_message(user_id=123456789, content="Hello, world!")
except DiscordProxyTimeoutError as ex:
    # handle timeout
    ...
```

2.2 gRPC

Alternatively you can use the gRPC protocol directly in your code to interact with Discord. This approach is more complex and requires a deeper understanding of gRPC. But it will also give you the most flexibility with full access to all gRPC features.

2.2.1 gRPC client example

Here is a hello code example for a gRPC client that is sending a direct "hello world" message to a user:

```
import grpc

from discordproxy.discord_api_pb2 import SendDirectMessageRequest
from discordproxy.discord_api_pb2_grpc import DiscordApiStub

# opens a channel to Discord Proxy
with grpc.insecure_channel("localhost:50051") as channel:
    # create a client for the DiscordApi service
    client = DiscordApiStub(channel)
    # create a request to use the SendDirectMessageRequest method of the service
    request = SendDirectMessageRequest(user_id=123456789, content="Hello, world!")
    # send the request to Discord Proxy
    client.SendDirectMessage(request)
```

2.2.2 gRPC error handling

If a gRPC request fails a `grpc.RpcError` exception will be raised. RPC errors return the context of the request, consisting of two fields:

- `code`: the gRPC status code
- `details`: a string with additional details about the error.

The most common errors you can except will be originating from calls to the Discord API. e.g. if a user is no longer a member of the guild the Discord API will return a http response code 404. Discord Proxy will map all HTTP response codes from Discord to a gRPC status codes and raise a gRPC error exceptions (see also *gRPC status codes*). In addition the details field of that exception will contain the full error information as JSON object (see also *gRPC details*).

Code Example

Here is an example on how to catch and process an error exception from your gRPC calls:

```
try:
    client.SendDirectMessage(request)
except grpc.RpcError as e:
    # print error code and details
    print(f"Code: {e.code()}")
    print(f"Details: {e.details()}")
```

gRPC status codes

Here is how HTTP error codes are mapped against gRPC status codes:

See also:

Status codes and their use in gRPC.

gRPC details

The Discord API will return two types of error codes:

- HTTP response code (e.g. 404 if a request user does not exist)
- JSON error code (e.g. 30007 when the maximum number of webhooks is reached)

In addition the response may contain some additional error text. All that information will be encoded as JSON in the details attribute of the gRPC error exception. Here is an example:

```
{
    "type": "HTTPException",
    "status": 403,
    "code": 50001,
    "text": "Missing Access"
}
```

Legend:

- `status`: HTTP status code
- `code`: JSON error code
- `text`: Error message

Note: For most cases it should be sufficient to deal with the status code. The JSON error code is only needed in some special cases.

To simplify dealing with the JSON error objects you can also use this helper from the djangoproxy package, which will parse the details and return them as handy named tuple:

```
from discordproxy.helpers import parse_error_details

try:
    client.SendDirectMessage(request)
except grpc.RpcError as e:
    details = parse_error_details(e)
    print(f"HTTP response code: {details.status}")
    print(f"JSON error code: {details.code}")
    print(f"Discord error message: {details.text}")
```

See also:

For the documentation of all helpers see: *Helpers*

2.2.3 gRPC timeouts

Here is how to use timeout with requests to the Discord Proxy. All timeouts are in seconds:

```
try:
    client.SendDirectMessage(request, timeout=10)
except grpc.RpcError as e:
    # handle timeouts
```

Should a timeout be triggered the client will receive a `grpc.RpcError` with status code `DEADLINE_EXCEEDED`.

2.3 Rate limiting

The Discord API is imposing rate limiting to all requests. Discord Proxy will automatically adhere to those rate limits by suspending a request until it can be sent. This can in certain situations result in requests taking a longer time to complete. If you need to complete your Discord request within a certain time, please see the sections about how to set custom timeouts.

CHAPTER THREE

API

The API is defined with Google's Protocol Buffers (aka protobuf). It consists of the service *DiscordApi* and it's methods.

Please see *Protocol Documentation* for a complete documentation of the service with it's methods and messages.

Or if you want to generate your own gPRC client you can find the protobuf definition files in the `/protobufs` folder.

In addition the Python pacakge contains an overview of the gRPC classes and helpers useful for gRPC clients.

3.1 Protocol Documentation

3.1.1 Table of Contents

3.1.2 discord_api.proto

Discord API service

This file contains all messages and services currently supported by Discord Proxy

Channel

Source: https://discord.com/developers/docs/resources/channel#channel-object

Embed

Source: https://discord.com/developers/docs/resources/channel#embed-object-embed-structure

Embed.Author

Embed.Field

Embed.Footer

Embed.Image

Embed.Provider

Embed.Thumbnail

Embed.Video

Emoji

Source: https://discord.com/developers/docs/resources/emoji#emoji-object

GetGuildChannelsRequest

GetGuildChannelsResponse

GuildMember

Source: https://discord.com/developers/docs/resources/guild#guild-member-object

Message

Source: https://discord.com/developers/docs/resources/channel#message-object

Message.Activity

Message.Application

Message.Attachment

Message.ChannelMention

Message.Reaction

Message.Reference

Message.Sticker

Role

Source: https://discord.com/developers/docs/topics/permissions#role-object

Role.Tag

SendChannelMessageRequest

SendChannelMessageResponse

SendDirectMessageRequest

SendDirectMessageResponse

User

Source: https://discord.com/developers/docs/resources/user#user-object

Channel.Type

Message.Activity.Type

Message.Sticker.Type

Message.Type

DiscordApi

Provides access to the Discord API

3.1.3 Scalar Value Types

3.2 Python package

Modules of the Discord Proxy package that are relevant for clients.

3.2.1 Client

3.2.2 Exceptions

3.2.3 gRPC classes

3.2.4 Helpers

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search