Pyrox Documentation

Release 0.4.5

John Hopper

November 12, 2014

Contents

Pyrox is a HTTP reverse proxy that can intercept requests ahead of an upstream HTTP REST service. This allows reuse of common middleware functions like: message enhancement, dynamic routing, authentication, authorization, resource request rate limiting, service distribution, content negotiation and content transformation. These services can then be scaled horizontally separate the origin REST endpoint.

Built ontop of the Tornado Async I/O library , the HTTP code inside Pyrox can scale to thousands of concurrent clients and proxy them to a similar number of upstream REST services.

CHAPTER 1

Getting Started

Below are some helpful documents to help get you started in using Pynsive.

1.1 Building Pyrox

1.1.1 Requirements

Pyrox is a complex daemon that requires a few things installed in your development environment before it can be built.

- GCC (Tested on versions **4.6.x** and **4.7.x**)
- Python Development Libaries

Installing Python Dependencies and Building Pyrox

```
# This requirements file contains requirements related only to Pyrox development
pip install -r tools/dev-requires

# This requirements file contains requirements needed to install and run Pyrox
pip install -r tools/pip-requires

# This requirements file contains requirements needed to test Pyrox
pip install -r tools/test-requires

# This script will auto-build Pyrox and then launch it
./pyrox_dev.sh start
```

1.2 Installing Pyrox

1.2.1 Debian Compatible Systems

Download the correct .deb file for your architecture.

- Debian Wheezy (No Link Available Yet)
- Ubuntu 12.04 LTS

```
dpkg -i pyrox_<version>_<arch>.deb
service pyrox start
```

1.2.2 Configuring Pyrox

CHAPTER 2

Pyrox Documentation

2.1 Using Pyrox

2.1.1 Anatomy of HTTP Message Processing

Pyrox allows a programmer to hook into the different stages of processing HTTP messages. There are four stages in total:

- Request Head
- Request Body
- Response Head
- Response Body

Filters that hook into these stages are organized into two pipelines. The upstream pipeline contains logic that needs to intercept HTTP requests being sent to an upstream origin service. The downstream pipeline contains logic that needs to intercept HTTP responses being sent to the downstream client.

2.1.2 Writing Your First Filter

As a programmer, Pyrox allows you to hook into the various processing stages of a HTTP message via python decorators. There are decorators specified for each stage.

Message Head Decorators

Message head decorators must be applied to class functions that answer to one argument that represents either the request message head or the response message head.

```
import pyrox.filtering as filtering

class FilterTest(filtering.HttpFilter):

    @filtering.handles_request_head
    def on_request_head(self, request_head):
        print('Got request head with verb: {}'.format(request_head.method))

    @filtering.handles_response_head
    def on_response_head(self, response_head):
        print('Got response head with status: {}'.format(response_head.status))
```

Message Body Decorators

Message body decorators must be applied to class functions that answer to two arguments. The first argument represents either the request message body or the response message body chunk being processed. The second argument is a writable object to which the processed content can be handed off to for transmission either upstream or downstream (depending on the stage being processed).

```
import pyrox.filtering as filtering

class FilterTest(filtering.HttpFilter):

    @filtering.handles_request_body
    def on_request_body(self, msg_part, output):
        print('Got request content chunk: {}'.format(msg_part))
        output.write(msg_part)

    @filtering.handles_response_body
    def on_response_body(self, msg_part, output):
        print('Got response content chunk: {}'.format(msg_part))
        output.write(msg_part)
```

Stacking Decorators on Common Functionality

Pyrox decorators may be stacked onto a class function that adheres to the expected interface.

```
import pyrox.filtering as filtering

class FilterTest(filtering.HttpFilter):

    @filtering.handles_request_head
    @filtering.handles_response_head
    def on_head(self, msg_head):
        print('Got msg head: {}'.format(msg_head))

    @filtering.handles_request_body
    @filtering.handles_response_body
    def on_body(self, msg_part, output):
        print('Got message content part: {}'.format(msg_part))
        output.write(msg_part)
```

2.1.3 Utilizing Pyrox's HTTP Model

The Pyrox HTTP model is quite feature-full and has many helpers designed to do their best in making your life easier. Below are come common usage patterns available.

Passing Arbitrary Data to the Next Filter

Pyrox HTTP request and response objects inherit from a common class called the HTTP message. The HTTP message has an attribute called local_data which travels along with the message object to the next filter in the pipeline. The following filter may then search this attribute for any interesting data, thus allowing filters to sparingly communicate state without having to modify the HTTP message itself.

```python
import uuid
import pyrox.filtering as filtering

class FilterTest(filtering.HttpFilter):

    @filtering.handles_request_head
    def on_request_head(self, request_head):
        request_head.local_data['transaction_id'] = str(uuid.uuid4())
        return filtering.next()
```

2.1.4 Pipeline Processing and Logic

Pyrox filters may influence how Pyrox handles further processing of the HTTP message by returning control actions. These control actions are available for import in the filtering module.

Passing a Message Stage

By default, if a filter returns None, has no return or returns a FilterAction with the kind class member set to NEXT_FILTER, Pyrox will continue handing off the HTTP message stage down its associated pipeline to the next filter.

```python
import pyrox.filtering as filtering

class FilterTest(filtering.HttpFilter):

    @filtering.handles_request_head
    def on_request_head(self, request_head):
        # Do nothing but pass the stage to the next filter in the
        # filter pipeline
        return filtering.next()

    @filtering.handles_response_head
    def on_response_head(self, response_head):
        # No return also defaults to passing the message stage to the
        # next filter in the pipeline
        pass
```

Consuming a Message Stage

Consuming a HTTP message stage tells Pyrox to continue proxying the message but to stop processing it through its associated pipeline.

```python
import pyrox.filtering as filtering

class FilterTest(filtering.HttpFilter):

    @filtering.handles_request_head
    def on_request_head(self, request_head):
        # Do nothing but consume the http message stage
        return filtering.consume()
```

Rejecting a Message

Rejecting a HTTP message stage will return to the client with the passed response message head object. This response object will be serialized and sent to the client immediately after the function returns.

Note: rejecting a message may not occur during the response body message stage.

```
import pyrox.http as http
import pyrox.filtering as filtering

class FilterTest(filtering.HttpFilter):

    @filtering.handles_request_head
    def on_request_head(self, request_head):
        # Reject the request if it is not a GET request
        if request_head.method != 'GET':
            # Create a response object - this should be a static
            # instanace set elsewhere for performance reasons
            response = http.HttpResponse()
            response.version = '1.1'
            response.status = '405 Method Not Allowed'

            return filtering.reject(response)
```

Routing a Message

Pyrox allows for a message to be routed to an upstream host target. By default, messages are proxied to upstream hosts defined in the Pyrox configuration. When more flexibility is required, a filter action may be returned that informs Pyrox of the message's intended upstream destination.

Note: routing a message is only allowed during the request message head stage.

```
import pyrox.filtering as filtering

class FilterTest(filtering.HttpFilter):

    @filtering.handles_request_head
    def on_request_head(self, request_head):
        # Do nothing but route the request
        return filtering.route('google.com:80')
```

2.2 Pyrox API Documentation

2.2.1 `pyrox` Module

`pyrox.http.model` Module

class `pyrox.http.model.`**`HttpHeader`**`(`*name*`)`

Bases: `object`

Defines the fields for a HTTP header

Attributes:

name A bytearray or string value representing the field-name of the header.

class `pyrox.http.model.`**`HttpMessage`**`(`*version='1.1'*`)`

Bases: `object`

Parent class for requests and responses. Many of the elements in the messages share common structures.

Attributes:

headers A dictionary of the headers currently stored in this HTTP message.

version A bytearray or string value representing the major-minor version of the HttpMessage.

local_data The local_data variable is a dictionary that may be used as a holding place for data that other filters may then access and utilize. Setting entries in this dictionary does not modify the HTTP model in anyway.

`get_header``(`*name*`)`

Returns the header that matches the name via case-insensitive matching. Unlike the header function, if the header does not exist then a None result is returned.

`header``(`*name*`)`

Returns the header that matches the name via case-insensitive matching. If the header does not exist, a new header is created, attached to the message and returned. If the header already exists, then it is returned.

`headers`

`remove_header``(`*name*`)`

Removes the header that matches the name via case-insensitive matching. If the header exists, it is removed and a result of True is returned. If the header does not exist then a result of False is returned.

`replace_header``(`*name*`)`

Returns a new header with a field set to name. If the header exists then the header is removed from the request first.

`set_default_headers``()`

Allows messages to set default headers that must be added to the message before its construction is complete.

class `pyrox.http.model.`**`HttpRequest`**

Bases: `pyrox.http.model.HttpMessage`

HttpRequest defines the HTTP request attributes that will be available to a HttpFilter.

Attributes:

method A bytearray or string value representing the request's method verb.

url A bytearray or string value representing the requests' uri path including the query and fragment string.

`to_bytes``()`

class `pyrox.http.model.`**`HttpResponse`**

Bases: `pyrox.http.model.HttpMessage`

HttpResponse defines the HTTP response attributes that will be available to a HttpFilter.

Attributes:

status A string representing the response's status code and potentially its human readable component delimited by a single space.

to_bytes()

`pyrox.filtering.pipeline` Module

class `pyrox.filtering.pipeline.`**FilterAction**(*kind=0*, *payload=None*)

Bases: `object`

A filter action allows us to tell upstream controls what the filter has decided as the next course of action. Certain filter actions may include a response object for serialization out to the client in the case where the action enforces a rejection.

Attributes:

kind An integer value representing the kind of action this object is intended to communicate.

payload An argument to be passed on to the consumer of this action.

breaks_pipeline()

intercepts_request(*sefl*)

is_consuming()

is_replying()

is_routing()

class `pyrox.filtering.pipeline.`**HttpFilter**

Bases: `object`

HttpFilter is a marker class that may be utilized for dynamic gathering of filter logic.

class `pyrox.filtering.pipeline.`**HttpFilterPipeline**

Bases: `object`

The filter pipeline represents a series of filters. This pipeline currently serves bidirectional filtering (request and response). This chain will have the request head and response head events passed through it during the lifecycle of a client request. Each request is assigned a new copy of the chain, meaning that state may not be shared between requests during the lifetime of the filter chain or its filters.

Parameters **chain** – A list of HttpFilter objects organized to act as a pipeline with element 0 being the first to receive events.

add_filter(*http_filter*)

intercepts_req_body()

intercepts_resp_body()

on_request_body(*body_part*, *output*)

on_request_head(*request_head*)

on_response_body(*body_part*, *output*)

on_response_head(*response_head*)

`pyrox.filtering.pipeline.`**consume**()

Consumes the event and does not allow any further downstream filters to see it. This effectively halts execution of the filter chain but leaves the request to pass through the proxy.

`pyrox.filtering.pipeline.`**handles_request_body**(*request_func*)

This function decorator may be used to mark a method as usable for intercepting request body content.

handles_request_body will intercept the HTTP content in chunks as it arrives. This method, like others in the filter class may return a FilterAction.

`pyrox.filtering.pipeline.`**`handles_request_head`**`(`*request_func*`)`

This function decorator may be used to mark a method as usable for intercepting request head content.

handles_request_head will accept an HttpRequest object and implement the logic that will define the FilterActions to be applied to the request

`pyrox.filtering.pipeline.`**`handles_response_body`**`(`*request_func*`)`

This function decorator may be used to mark a method as usable for intercepting response body content.

handles_response_body will intercept the HTTP content in chunks as they arrives. This method, like others in the filter class, may return a FilterAction.

`pyrox.filtering.pipeline.`**`handles_response_head`**`(`*request_func*`)`

This function decorator may be used to mark a method as usable for intercepting response head content.

handles_response_head will accept an HttpResponse object and implement the logic that will define the FilterActions to be applied to the request

`pyrox.filtering.pipeline.`**`next`**`()`

Passes the current http event down the filter chain. This allows for downstream filters a chance to process the event.

`pyrox.filtering.pipeline.`**`reject`**`(`*response=None*`)`

Rejects the request that this event is related to. Rejection may happen during on_request and on_response. The associated response parameter becomes the response the client should expect to see. If a response parameter is not provided then the function will default to the configured default response.

Parameters **response** – the response object to reply to the client with

`pyrox.filtering.pipeline.`**`reply`**`(`*response*, *src*`)`

A special type of rejection that implies willful handling of a request. This call may optionally include a stream or a data blob to take the place of the response content body.

Parameters **response** – the response object to reply to the client with

`pyrox.filtering.pipeline.`**`route`**`(`*upstream_target*`)`

Routes the request that this event is related to. Usage of this method will halt execution of the filter pipeline and begin streaming the request to the specified upstream target. This method is invalid for handling an upstream response.

Parameters **upstream_target** – the URI string of the upstream target to route to.

`pyrox.server.config` Module

class `pyrox.server.config.`**`CoreConfiguration`**`(`*cfg*, *defaults=None*`)`

Bases: `pyrox.util.config.ConfigurationPart`

Class mapping for the Pyrox core configuration section.

```
# Core section
[core]
```

`bind_host`

Returns the host and port the proxy is expected to bind to when accepting connections. This option defaults to localhost:8080 if left unset.

```
bind_host = localhost:8080
```

enable_profiling

Returns a boolean value representing whether or not Pyrox should use a special single-process start up and run sequence so that code may be profiled. If unset, this defaults to False.

NOTE: If enabled, the number of processess Pyrox will be allowed to spin up will be limited to **1**

```
enable_profiling = True
```

plugin_paths

Returns a list of directories to plug into when attempting to resolve the names of pipeline filters. This option may be a single directory or a comma delimited list of directories.

```
# Any of the below are acceptable
plugin_paths = /usr/share/project/python
plugin_paths = /usr/share/project/python,/usr/share/other/python
plugin_paths = /opt/pyrox/stock, /opt/pyrox/contrib
```

processes

Returns the number of processes Pyrox should spin up to handle messages. If unset, this defaults to 1.

```
processes = 75
```

class pyrox.server.config.**LoggingConfiguration**(*cfg*, *defaults=None*)

Bases: pyrox.util.config.ConfigurationPart

Class mapping for the Pyrox logging configuration section.

```
# Logging section
[logging]
```

console

Returns a boolean representing whether or not Pyrox should write to stdout for logging purposes. This value may be either True of False. If unset this value defaults to False.

```
console = True
```

logfile

Returns the log file the system should write logs to. When set, Pyrox will enable writing to the specified file for logging purposes If unset this value defaults to None.

```
logfile = /var/log/pyrox/pyrox.log
```

verbosity

Returns the type of log messages that should be logged. This value may be one of the following: DEBUG, INFO, WARNING, ERROR or CRITICAL. If unset this value defaults to WARNING.

```
verbosity = DEBUG
```

class pyrox.server.config.**PipelineConfiguration**(*cfg*, *defaults=None*)

Bases: pyrox.util.config.ConfigurationPart

Class mapping for the Pyrox pipeline configuration section.

```
# Pipeline section
[pipeline]
```

Configuring a pipeline requires the admin to first configure aliases to each filter referenced. This is done by adding a named configuration option to this section that does not match “upstream” or “downstream.” Filter aliases must point to a class or function that returns a filter instance with the expected entry points.

After the filter aliases are specified, they may be then organized in comma delimited lists and assigned to either the "upstream" option for filters that should receive upstream events or the "downstream" option for filters that should receive downstream events.

In the context of Pyrox, upstream events originate from the requesting client also known as the request. Downstream events originate from the origin service (the upstream request target) and is also known as the response.

```
[pipeline]
    filter_1 = myfilters.upstream.Filter1
    filter_2 = myfilters.upstream.Filter2
    filter_3 = myfilters.downstream.Filter3

    upstream = filter_1, filter_2
    downstream = filter_3
```

downstream
Returns the list of filters configured to handle downstream events. This configuration option must be a comma delimited list of filter aliases. If left unset this option defaults to an empty tuple.

```
downstream = filter_3
```

upstream
Returns the list of filters configured to handle upstream events. This configuration option must be a comma delimited list of filter aliases. If left unset this option defaults to an empty list.

```
upstream = filter_1, filter_2
```

use_singletons
Returns a boolean value representing whether or not Pyrox should reuse filter instances for up and downstream aliases. This means, effectively, that a filter specified in both pipelines will maintain its state for the request and response lifecycle. If left unset this option defaults to false.

```
use_singletons = True
```

class pyrox.server.config.**RoutingConfiguration**(*cfg*, *defaults=None*)
Bases: pyrox.util.config.ConfigurationPart

Class mapping for the Pyrox routing configuration section.

```
# Routing section
[routing]
```

upstream_hosts
Returns a list of downstream hosts to proxy requests to. This may be set to either a single valid URL string or a comma delimited list of valid URI strings. This option defaults to http://localhost:80 if left unset.

```
upstream_hosts = http://host:port, https://host:port
```

class pyrox.server.config.**SSLConfiguration**(*cfg*, *defaults=None*)
Bases: pyrox.util.config.ConfigurationPart

Class mapping for the Portal configuration section 'ssl'

cert_file
Returns the path of the cert file for SSL configuration within Pyrox. If left unset the value will default to None.

:: cert_file = /etc/pyrox/ssl/server.cert

key_file
Returns the path of the key file for SSL configuration within Pyrox. If left unset the value will default to None.

:: key_file = /etc/pyrox/ssl/server.key

class `pyrox.server.config.`**`TemplatesConfiguration`**(*cfg*, *defaults=None*)

Bases: `pyrox.util.config.ConfigurationPart`

Class mapping for the Pyrox teplates configuration section.

```
# Templates section
[templates]
```

`pyrox_error_sc`

Returns the status code to be set for any error that happens within Pyrox that would prevent normal service of client requests. If left unset this option defaults to 502.

```
pyrox_error_sc = 502
```

`rejection_sc`

Returns the default status code to be set for client request rejection made with no provided response object to serialize. If left unset this option defaults to 400.

```
rejection_sc = 400
```

`pyrox.server.config.`**`load_pyrox_config`**(*location*)

`pyrox.util.config` Module

class `pyrox.util.config.`**`Configuration`**(*cfg_cls_list*, *cfg*, *defaults*)

Bases: `object`

exception `pyrox.util.config.`**`ConfigurationError`**(*msg*)

Bases: `exceptions.Exception`

class `pyrox.util.config.`**`ConfigurationPart`**(*cfg*, *defaults=None*)

Bases: `object`

A configuration part is an OO abstraction for a ConfigParser that allows for ease of documentation and usage of configuration options. All subclasses of ConfigurationPart must follow a naming convention. A configuration part subclass must start with the name of its section. This must then be followed by the word "Configuration." This convention results in subclasses with names similar to: CoreConfiguration and LoggingConfiguration.

A configuration part subclass will have its section set to the lowercase name of the subclass sans the word such that a subclass with the name, "LoggingConfiguration" will reference the ConfigParser section "logging" when looking up options.

`get`(*option*)

`getboolean`(*option*)

`getint`(*option*)

`has_option`(*option*)

`name`()

`options`()

`pyrox.util.config.`**`load_config`**(*cfg_module_name*, *location*, *defaults=None*)

2.2.2 Indices

- *genindex*
- *modindex*

CHAPTER 3

That Legal Thing...

This software library is released to you under the MIT Software License . See LICENSE for more information.

Index