pyramid$_{zipkin}$

Release 0.19.2

Dec 12, 2018

Contents

This project acts as a Pyramid tween by using py_zipkin to facilitate creation of zipkin service spans.

Features include:

- Blacklisting specific route/paths from getting traced.
- `zipkin_tracing_percent` to control the percentage of requests getting sampled.
- Adds `http.uri`, `http.uri.qs`, and `status_code` binary annotations automatically for each trace.
- Allows configuration of additional arbitrary binary annotations.

CHAPTER 1

Install

```
pip install pyramid_zipkin
```

CHAPTER 2

Usage

In your service's webapp, you need to include:

```
config.include('pyramid_zipkin')
```

Contents:

2.1 Configuring pyramid_zipkin

2.1.1 Required configuration settings

zipkin.transport_handler

A callback function which takes a single *message data* parameter. A sample method can be something like this:

1. FOR *kafka* TRANSPORT:

```
from kafka import SimpleProducer, KafkaClient

def kafka_handler(stream_name, message):

    kafka = KafkaClient('{0}:{1}'.format('localhost', 9092))
    producer = SimpleProducer(kafka, async=True)
    producer.send_messages(stream_name, message)
```

The above example uses python package kafka-python.

2. FOR *scribe* TRANSPORT:

```
import base64
from scribe import scribe
from thrift.transport import TTransport, TSocket
```

(continues on next page)

(continued from previous page)

```
from thrift.protocol import TBinaryProtocol

def scribe_handler(stream_name, message):
    socket = TSocket.TSocket(host="HOST", port=9999)
    transport = TTransport.TFramedTransport(socket)
    protocol = TBinaryProtocol.TBinaryProtocol(
        trans=transport, strictRead=False, strictWrite=False)
    client = scribe.Client(protocol)
    transport.open()

    message_b64 = base64.b64encode(message).strip()
    log_entry = scribe.LogEntry(stream_name, message_b64)
    result = client.Log(messages=[log_entry])
    if result == 0:
      print 'success'
```

The above example uses python package facebook-scribe for the scribe APIs but any similar package can do the work.

2.1.2 Optional configuration settings

All below settings are optional and have a sane default functionality set. These can be used to fine tune as per your use case.

service_name

A string representing the name of the service under instrumentation. It defaults to *unknown*.

zipkin.blacklisted_paths

A list of paths as strings, regex strings, or compiled regexes, any of which if matched with the request path will not be sampled. Pre-compiled regexes will be the fastest. Defaults to *[]*. Example:

```
'zipkin.blacklisted_paths': [r'^/status/?',]
```

zipkin.blacklisted_routes

A list of routes as strings any of which if matched with the request route will not be sampled. Defaults to *[]*. Example:

```
'zipkin.blacklisted_routes': ['some_internal_route',]
```

zipkin.host

The host ip that is used for zipkin spans. If not given, host will be automatically determined.

zipkin.port

The port that is used for zipkin spans. If not given, port will be automatically determined.

zipkin.stream_name

A log name to log Zipkin spans to. Defaults to 'zipkin'.

zipkin.tracing_percent

A number between 0.0 and 100.0 to control how many request calls get sampled. Defaults to *0.50*. Example:

```
'zipkin.tracing_percent': 100.0  # Trace all the calls.
```

zipkin.trace_id_generator

A method definition to generate a *trace_id* for the request. This is useful if you, say, have a unique_request_id you'd like to preserve. The trace_id must be a 64-bit hex string (e.g. '17133d482ba4f605'). By default, it creates a random trace id.

The method MUST take *request* as a parameter (so that you can make trace id deterministic).

zipkin.create_zipkin_attr

A method that takes *request* and creates a ZipkinAttrs object. This can be used to generate span_id, parent_id or other ZipkinAttrs fields based on request parameters.

The method MUST take *request* as a parametr and return a ZipkinAttrs object.

zipkin.is_tracing

A method that takes *request* and determines if the request should be traced. This can be used to determine if a request is traced based on custom application specific logic.

The method MUST take *request* as a parameter and return a Boolean.

zipkin.set_extra_binary_annotations

A method that takes *request* and *response* objects as parameters and produces extra binary annotations. If this config is omitted, only *http.uri* and *http.uri.qs* are added as binary annotations. The return value of the callback must be a dictionary, and all keys and values must be in *str* format. Example:

```
def set_binary_annotations(request, response):
    return {'view': get_view(request)}

settings['zipkin.set_extra_binary_annotations'] = set_binary_annotations
```

zipkin.always_emit_zipkin_headers

Whether to forward the Zipkin's headers if the request is not being sampled. Defaults to True.

Set to False if you're really concerned with performance as it'll save you about 300us on every non-traced request.

zipkin.request_context

If it contains a valid request attribute, this specifies the stack for storing the zipin attributes. If the name is invalid or the option is missing, attributes will be stored in a thread local context. The syntax is a path in dotted notation, e.g. 'request.context.zipkin'.

This option enables support for an cooperative multithreading environment (e.g. asyncio).

```
settings['zipkin.request_context'] = 'request.context.zipkin'
```

zipkin.post_handler_hook

Callback function for processing actions after the tween functionality is executed and before the response is sent back.

The actions for example could be to modify the response headers.

```
settings['zipkin.post_handler_hook'] = post_handler_hook

def post_handler_hook(request, response):
    do_some_work(response)
```

zipkin.firehose_handler [EXPERIMENTAL]

Callback function for "firehose tracing" mode. This will log 100% of the spans to this handler, regardless of sampling decision.

This is experimental and may change or be removed at any time without warning.

2.1.3 Configuring your application

These settings can be added at Pyramid application setup like so:

```
def main(global_config, **settings):
    # ...
    settings['service_name'] = 'zipkin'
    settings['zipkin.transport_handler'] = scribe_handler
    settings['zipkin.stream_name'] = 'zipkin_log'
    settings['zipkin.blacklisted_paths'] = [r'^/foo/?']
    settings['zipkin.blacklisted_routes'] = ['bar']
    settings['zipkin.trace_id_generator'] = lambda req: '0x42'
    settings['zipkin.set_extra_binary_annotations'] = lambda req, resp: {'attr':␣
↪str(req.attr)}
    # ...and so on with the other settings...
    config = Configurator(settings=settings)
    config.include('pyramid_zipkin')
```

2.2 pyramid_zipkin Package

2.2.1 `pyramid_zipkin` Package

2.2.2 `tween` Module

2.2.3 `request_helper` Module

2.3 0.23.0 (2018-12-11)

- Add post_handler_hook to the tween.

2.4 0.22.0 (2018-10-02)

- Set *zipkin.use_pattern_as_span_name* to use the pyramid route pattern as span_name rather than the raw url.
- Requires py_zipkin >= 0.13.0.

2.5 0.21.1 (2018-06-03)

- Use renamed py_zipkin.storage interface
- Remove deprecated logger.debug() usage in tests
- Require py_zipkin >= 0.13.0

2.6 0.21.0 (2018-03-09)

- Added support for http.route annotation

2.7 0.20.3 (2018-03-08)

- Add max_span_batch_size to the zipkin tween settings.

2.8 0.20.2 (2018-02-20)

- Require py_zipkin >= 0.11.0

2.9 0.20.1 (2018-02-13)

- Added support for experimental firehose mode

2.10 0.20.0 (2018-02-09)

- Added support for using a request-specific stack for storing zipkin attributes.

2.11 0.19.2 (2017-08-17)

- Trace context is again propagated for non-sampled requests.

2.12 0.19.1 (2017-06-02)

- Modified tween.py to include host and port when creating a zipkin span.

2.13 0.19.0 (2017-06-01)

- Added zipkin.always_emit_zipkin_headers config flag.
- Skip zipkin_span context manager if the request is not being sampled to improve performance and avoid unnecessary work.

2.14 0.18.2 (2017-03-06)

- Using new update_binary_annotations functions from py_zipkin.
- Requires py_zipkin >= 0.7.0

2.15 0.18.1 (2017-02-06)

- No changes

2.16 0.18.0 (2017-02-06)

- Add automatic timestamp/duration reporting for root server span. Also added functionality for individual services to override this setting in configuration.

2.17 0.17.0 (2016-12-16)

- Add registry setting to force py-zipkin to add a logging annotation to server traces. Requires py-zipkin >= 0.4.4.

2.18 0.16.1 (2016-10-14)

- support for configuring custom versions of create_zipkin_attr and is_tracing through the pyramid registry.

2.19 0.16.0 (2016-10-06)

- Fix sample rate bug and make sampling be random and not depend on request id.

2.20 0.15.0 (2016-09-29)

- Make *get_trace_id* function more defensive about what types of trace ids it accepts. Converts "0x…" and "-0x…" IDs to remove the leading "Ox"s

2.21 0.14.0 (2016-09-29)

- Make *zipkin.transport_handler* a function that takes two arguments, a stream_name and a message.

2.22 0.13.1 (2016-09-21)

- Alias *create_headers_for_new_span* to *create_http_headers_for_new_span* for backwards compatibility.

2.23 0.13.0 (2016-09-12)

- Moved non-pyramid and zipkin-only code to py_zipkin package
- 'zipkin.transport_handler' now only takes a single message parameter
- *create_headers_for_new_span* is moved to py_zipkin and renamed to *create_http_headers_for_new_span*

2.24 0.12.3 (2016-07-27)

- Fix coverage command invocation to be compatible with coverage v4.2

2.25 0.12.2 (2016-07-15)

- make "service_name" default to "unknown" when not found in registry

2.26 0.12.1 (2016-07-08)

- Add @zipkin_span decorator for logging functions as spans

2.27 0.11.1 (2016-04-28)

- Binary annotation values are converted to str
- Removed restriction where only successful status codes are logged
- Added status code as a default binary annotation
- Prevent errors when ZipkinAttrs doesn't exist (usually in multithreaded environments)
- pyramid-zipkin is a pure python package

2.28 0.11.0 (2016-04-19)

- Renames ClientSpanContext to SpanContext, adds 'ss' and 'sr' annotations.

2.29 0.10.0 (2016-04-12)

- Always generate ZipkinAttrs, even when a request isn't sampled.

2.30 0.9.2 (2016-04-07)

- Don't set parent_span_id on root span

2.31 0.9.1 (2016-03-29)

- Made generate_random_64bit_string always return str, not unicode

2.32 0.9.0 (2016-03-27)

- Fixed bug where headers were not 64-bit unsigned hex strings.
- Added ClientSpanContext, that lets users log arbitrary trees of client spans.
- Deprecates "is_client=True" debug logging key in favor of a non-None "service_name" key for indicating that a span logged is a new client span.
- Batches up additional annotations in client before sending to the collector.

2.33 0.8.1 (2016-03-02)

- Spans without a span ID will generate a new span ID by default.

2.34 0.8.0 (2016-03-01)

- Add ability to override "service_name" attribute when logging client spans.

2.35 0.7.1 (2016-02-26)

- Don't re-compile path regexes

2.36 0.7.0 (2016-02-24)

- Don't enter ZipkinLoggingContext if request is not sampled.

2.37 0.6.0 (2016-02-06)

- Fix bug which was squashing identical span names.
- over=EXCVIEW ordering instead of over=MAIN

2.38 0.5.0 (2016-01-14)

- Add support for *set_extra_binary_annotations* callback.

2.39 0.4.0 (2016-01-07)

- Add *http.uri.qs* annotation which includes query string, *http.uri* doesn't.

2.40 0.3.0 (2015-12-29)

- Change config parameters to be generic for scribe/kafka transport.

2.41 0.2.2 (2015-12-09)

- Compatible with py33, py34. Replaced Thrift with thriftpy.

2.42 0.1.2 (2015-12-03)

- Re-assign empty list to threading_local.requests if attr not present instead of globally assigning empty list.

2.43 0.1.0 (2015-11-08)

- pyramid-zipkin setup.

CHAPTER 3

Indices and tables

- genindex
- modindex
- search