

---

# **aiozipkin Documentation**

***Release 1.1.1-***

**aiozipkin contributors**

**Mar 15, 2022**



# CONTENTS

<b>1</b>	<b>Features</b>	<b>3</b>
<b>2</b>	<b>Contents</b>	<b>5</b>
2.1	Tutorial . . . . .	5
2.2	Examples of aiozipkin usage . . . . .	5
2.3	Support of other collectors . . . . .	8
2.4	API . . . . .	9
2.5	Contributing . . . . .	12
<b>3</b>	<b>Indices and tables</b>	<b>13</b>
	<b>Python Module Index</b>	<b>15</b>
	<b>Index</b>	<b>17</b>



**aiozipkin** is Python 3.6+ module that adds distributed tracing capabilities for `asyncio` applications with zipkin (<http://zipkin.io>) server instrumentation.

`zipkin` is a distributed tracing system. It helps gather timing data needed to troubleshoot latency problems in microservice architectures. It manages both the collection and lookup of this data. Zipkin's design is based on the Google Dapper paper.

Applications instrumented with **aiozipkin** report timing data to `zipkin`. The Zipkin UI also presents a Dependency diagram showing how many traced requests went through each application. If you are troubleshooting latency problems or errors, you can filter or sort all traces based on the application, length of trace, annotation, or timestamp.



## FEATURES

- Distributed tracing capabilities to **asyncio** applications.
- Supports [zipkin](#) v2 protocol.
- Easy to use API.
- Explicit context handling, no thread local variables.
- Can work with [jaeger](#) and [stackdriver](#) ([google](#)) through zipkin compatible API.
- Can be integrated with AWS X-Ray by [aws](#) proxy.





## CONTENTS

### 2.1 Tutorial

### 2.2 Examples of aiozipkin usage

Below is a list of examples from [aiozipkin/examples](#)

Every example is a correct tiny python program.

#### 2.2.1 Basic Usage

```
import asyncio

import aiozipkin as az

async def run() -> None:
    # setup zipkin client
    zipkin_address = "http://127.0.0.1:9411/api/v2/spans"
    endpoint = az.create_endpoint("simple_service", ipv4="127.0.0.1", port=8080)

    # creates tracer object that traces all calls, if you want sample
    # only 50% just set sample_rate=0.5
    tracer = await az.create(zipkin_address, endpoint, sample_rate=1.0)

    # create and setup new trace
    with tracer.new_trace(sampled=True) as span:
        span.name("root_span")
        span.tag("span_type", "root")
        span.kind(az.CLIENT)
        span.annotate("SELECT * FROM")
        # imitate long SQL query
        await asyncio.sleep(0.1)
        span.annotate("start end sql")

    # create child span
    with tracer.new_child(span.context) as nested_span:
        nested_span.name("nested_span_1")
        nested_span.kind(az.CLIENT)
```

(continues on next page)

(continued from previous page)

```

        nested_span.tag("span_type", "inner1")
        nested_span.remote_endpoint("remote_service_1")
        await asyncio.sleep(0.01)

    # create other child span
    with tracer.new_child(span.context) as nested_span:
        nested_span.name("nested_span_2")
        nested_span.kind(az.CLIENT)
        nested_span.remote_endpoint("remote_service_2")
        nested_span.tag("span_type", "inner2")
        await asyncio.sleep(0.01)

    await tracer.close()
    print("-" * 100)
    print("Check zipkin UI for produced traces: http://localhost:9411/zipkin")

if __name__ == "__main__":
    loop = asyncio.get_event_loop()
    loop.run_until_complete(run())

```

## 2.2.2 aiohttp Example

Full featured example with aiohttp application:

```

import asyncio

from aiohttp import web

import aiozipkin as az

async def handle(request: web.Request) -> web.StreamResponse:
    tracer = az.get_tracer(request.app)
    span = az.request_span(request)

    with tracer.new_child(span.context) as child_span:
        child_span.name("mysql:select")
        # call to external service like https://python.org
        # or database query
        await asyncio.sleep(0.01)

    text = """
    <html lang="en">
    <head>
        <title>aiohttp simple example</title>
    </head>
    <body>
        <h3>This page was traced by aiozipkin</h3>
        <p><a href="http://127.0.0.1:9001/status">Go to not traced page</a></p>
    </body>

```

(continues on next page)

(continued from previous page)

```

</html>
"""
return web.Response(text=text, content_type="text/html")

async def not_traced_handle(request: web.Request) -> web.StreamResponse:
    text = """
    <html lang="en">
    <head>
        <title>aiohttp simple example</title>
    </head>
    <body>
        <h3>This page was NOT traced by aiozipkin</h3>
        <p><a href="http://127.0.0.1:9001">Go to traced page</a></p>
    </body>
    </html>
    """
    return web.Response(text=text, content_type="text/html")

async def make_app(host: str, port: int) -> web.Application:
    app = web.Application()
    app.router.add_get("/", handle)
    # here we aquire reference to route, so later we can command
    # aiozipkin not to trace it
    skip_route = app.router.add_get("/status", not_traced_handle)

    endpoint = az.create_endpoint("aiohttp_server", ipv4=host, port=port)

    zipkin_address = "http://127.0.0.1:9411/api/v2/spans"
    tracer = await az.create(zipkin_address, endpoint, sample_rate=1.0)
    az.setup(app, tracer, skip_routes=[skip_route])
    return app

def run() -> None:
    host = "127.0.0.1"
    port = 9001
    loop = asyncio.get_event_loop()
    app = loop.run_until_complete(make_app(host, port))
    web.run_app(app, host=host, port=port)

if __name__ == "__main__":
    run()

```

### 2.2.3 Microservices Demo

There is a larger micro services example, using aiohttp. This demo consists of five simple services that call each other, as result you can study client server communication and zipkin integration for large projects. For more information see:

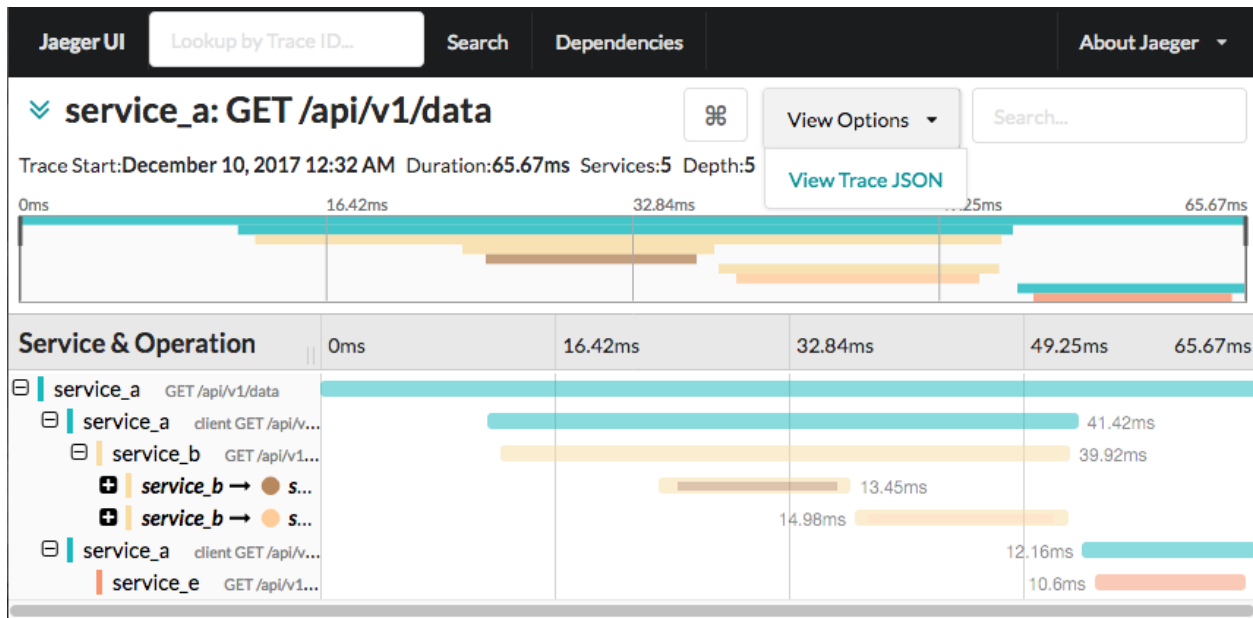
<https://github.com/aio-lib/aiozipkin/tree/master/examples>

## 2.3 Support of other collectors

**aiozipkin** can work with any other **zipkin** compatible service, currently we tested it with **jaeger** and **stackdriver**.

### 2.3.1 Jaeger support

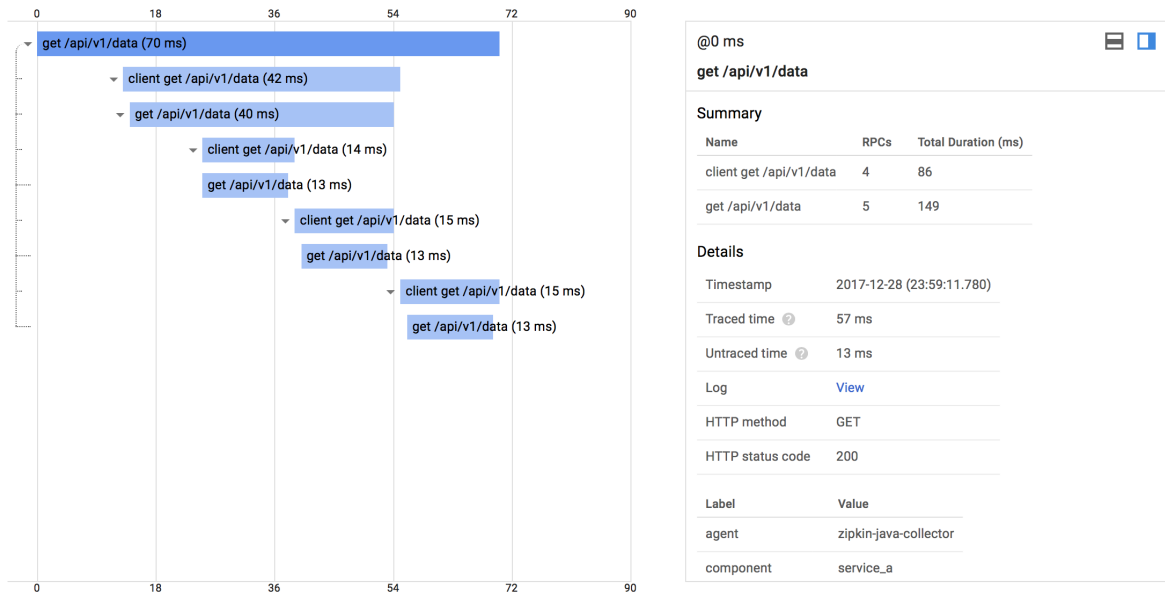
**jaeger** supports **zipkin** span format and as a result it is possible to use *aiozipkin* with **jaeger** server. You just need to specify *jaeger* server address and it should work out of the box. No need to run a local zipkin server. For more information see tests and **jaeger** documentation.



### 2.3.2 StackDriver support

Google **stackdriver** supports **zipkin** span format as a result it is possible to use *aiozipkin* with this **google** service. In order to make this work you need to setup zipkin service locally, that will send traces to the cloud. See **google** cloud documentation how to setup make zipkin collector:

Timeline



## 2.4 API

### 2.4.1 Core API Reference

#### aiozipkin.CLIENT

Constant indicates that span has been created on client side.

#### aiozipkin.SERVER

Constant indicates that span has been created on server side.

#### aiozipkin.PRODUCER

Constant indicates that span has been created by message producer.

#### aiozipkin.CONSUMER

Constant indicates that span has been created by message consumer.

#### aiozipkin.make\_context(headers: Dict[str, str])

Creates tracing context object from headers mapping if possible, otherwise returns *None*.

**Parameters** **headers** (*dict*) – hostname to serve monitor telnet server

**Returns** TraceContext object or None

#### coroutine aiozipkin.create(zipkin\_address, local\_endpoint, sample\_rate, send\_interval, loop, ignored\_exceptions)

Creates Tracer object

##### Parameters

- **zipkin\_address** (*Endpoint*) – information related to service address and name, where current zipkin tracer is installed
- **local\_endpoint** (*Endpoint*) – hostname to serve monitor telnet server
- **sample\_rate** (*float*) – hostname to serve monitor telnet server

- **send\_interval** (*float*) – hostname to serve monitor telnet server
- **loop** (*asyncio.EventLoop*) – hostname to serve monitor telnet server
- **Optional[List[Type[Exception]]]** – ignored\_exceptions list of exceptions which will not be labeled as error

**Returns** Tracer

**coroutine** aiozipkin.**create\_custom**(*transport, sampler, local\_endpoint, ignored\_exceptions*)

Creates Tracer object with a custom Transport and Sampler implementation.

**Parameters**

- **transport** (*TransportABC*) – custom transport implementation
- **sampler** (*SamplerABC*) – custom sampler implementation
- **local\_endpoint** (*Endpoint*) – hostname to serve monitor telnet server
- **Optional[List[Type[Exception]]]** – ignored\_exceptions list of exceptions which will not be labeled as error

**Returns** Tracer

**class** aiozipkin.**Endpoint**(*serviceName: str, ipv4=None, ipv6=None, port=None*)

This is a simple data only class, just holder for service related information:

**serviceName**

**ipv4**

**ipv6**

**port**

**class** aiozipkin.**TraceContext**(*trace\_id, parent\_id, span\_id, sampled, debug, shared*)

Immutable class with trace related data that travels across process boundaries.:

**Parameters**

- **trace\_id** (*str*) – hostname to serve monitor telnet server
- **parent\_id** (*Optional[str]*) – hostname to serve monitor telnet server
- **span\_id** (*str*) – hostname to serve monitor telnet server
- **sampled** (*str*) – hostname to serve monitor telnet server
- **debug** (*str*) – hostname to serve monitor telnet server
- **shared** (*float*) – hostname to serve monitor telnet server

**make\_headers()**

**Rtype dict** hostname to serve monitor telnet server

**class** aiozipkin.**Sampler**(*trace\_id, parent\_id, span\_id, sampled, debug, shared*)

TODO: add

**Parameters**

- **sample\_rate** (*float*) – XXX
- **seed** (*Optional[int]*) – seed value for random number generator

**is\_sampled**(*trace\_id*)

XXX

**Rtype bool** hostname to serve monitor telnet server

## 2.4.2 Aiohttp integration API

API for integration with `aiohttp.web`, just calling `setup` is enough for zipkin to start tracking requests. On high level attached plugin registers middleware that starts span on beginning of request and closes it on finish, saving important metadata, like route, status code etc.

`aiozipkin.APP_AIOZIPKIN_KEY`

Key, for aiohttp application, where aiozipkin related data is saved. In case for some reason you want to use 2 aiozipkin instances or change default name, this parameter should not be used.

`aiozipkin.REQUEST_AIOZIPKIN_KEY`

Key, for aiohttp request, where aiozipkin span related to current request is located.

`aiozipkin.setup(app, tracer, tracer_key=APP_AIOZIPKIN_KEY, request_key=APP_AIOZIPKIN_KEY)`

Sets required parameters in aiohttp applications for aiozipkin.

Tracer added into application context and cleaned after application shutdown. You can provide custom `tracer_key`, if default name is not suitable.

### Parameters

- **app** (`aiohttp.web.Application`) – application for tracer to attach
- **tracer** (`Tracer`) – aiozipkin tracer
- **skip\_routes** (`List`) – list of routes not to be traced
- **tracer\_key** (`str`) – key for aiozipkin state in aiohttp Application
- **request\_key** (`str`) – key for Span in request object

**Returns** `aiohttp.web.Application`

`aiozipkin.get_tracer(app, tracer_key=APP_AIOZIPKIN_KEY)`

Sets required parameters in aiohttp applications for aiozipkin.

By default tracer has `APP_AIOZIPKIN_KEY` in aiohttp application context, you can provide own key, if for some reason default one is not suitable.

### Parameters

- **app** (`aiohttp.web.Application`) – application for tracer to attach
- **tracer\_key** (`str`) – key where tracer stored in app

`aiozipkin.request_span(request, request_key=REQUEST_AIOZIPKIN_KEY)`

Return span created by middleware from request context, you can use it as parent on next child span.

### Parameters

- **app** (`aiohttp.web.Request`) – application for tracer to attach
- **request\_key** (`str`) – key where span stored in request

`aiozipkin.make_trace_config(tracer)`

Creates configuration compatible with aiohttp client. It attaches to relevant hooks and annotates timing.

**Parameters** `tracer` (*Tracer*) – to install in aiohttp tracer config

**Returns** `aiohttp.TraceConfig`

## 2.5 Contributing

### 2.5.1 Setting Development Environment

Thanks for your interest in contributing to aiozipkin, there are multiple ways and places you can contribute, help on documentation and tests is very appreciated.

To setup development environment, first of all just clone repository:

```
$ git clone git@github.com:aio-lib/aiozipkin.git
```

Create virtualenv with python3.6+. For example using *virtualenvwrapper* commands could look like:

```
$ cd aiozipkin
$ mkvirtualenv --python=`which python3.6` aiozipkin
```

After that please install libraries required for development:

```
$ pip install -r requirements-dev.txt
$ pip install -e .
```

### 2.5.2 Running Tests

Congratulations, you are ready to run the test suite:

```
$ make cov
```

To run individual test use following command:

```
$ py.test -sv tests/test_tracer.py -k test_name
```

Project uses [Docker](#) for integration tests, test infrastructure will automatically pull `zipkin:2` or `jaegertracing/all-in-one:1.0.0` image and start server, you don't to worry about this just make sure you have [Docker](#) installed.

### 2.5.3 Reporting an Issue

If you have found an issue with *aiozipkin* please do not hesitate to file an issue on the [GitHub](#) project. When filing your issue please make sure you can express the issue with a reproducible test case.

When reporting an issue we also need as much information about your environment that you can include. We never know what information will be pertinent when trying narrow down the issue. Please include at least the following information:

- Version of *aiozipkin* and *python*.
- Version *zipkin* server.
- Platform you're running on (OS X, Linux).



## INDICES AND TABLES

- `genindex`
- `modindex`
- `search`



## PYTHON MODULE INDEX

### a

`aiozipkin`, [9](#)



## INDEX

### A

`aiozipkin`  
    module, 9  
`APP_AIOZIPKIN_KEY` (in module *aiozipkin*), 11

### C

`CLIENT` (in module *aiozipkin*), 9  
`CONSUMER` (in module *aiozipkin*), 9  
`create()` (in module *aiozipkin*), 9  
`create_custom()` (in module *aiozipkin*), 10

### E

`Endpoint` (class in *aiozipkin*), 10

### G

`get_tracer()` (in module *aiozipkin*), 11

### I

`ipv4` (*aiozipkin.Endpoint* attribute), 10  
`ipv6` (*aiozipkin.Endpoint* attribute), 10  
`is_sampled()` (*aiozipkin.Sampler* method), 10

### M

`make_context()` (in module *aiozipkin*), 9  
`make_headers()` (*aiozipkin.TraceContext* method), 10  
`make_trace_config()` (in module *aiozipkin*), 11  
module  
    , 9

### P

`port` (*aiozipkin.Endpoint* attribute), 10  
`PRODUCER` (in module *aiozipkin*), 9

### R

`REQUEST_AIOZIPKIN_KEY` (in module *aiozipkin*), 11  
`request_span()` (in module *aiozipkin*), 11

### S

`Sampler` (class in *aiozipkin*), 10  
`SERVER` (in module *aiozipkin*), 9  
`serviceName` (*aiozipkin.Endpoint* attribute), 10

`setup()` (in module *aiozipkin*), 11

### T

`TraceContext` (class in *aiozipkin*), 10