pydantic

LOGFIRE

Why?

Building Libraries is easy

Building apps is much harder

- Networks
- Databases
- APIs
- Infra
- Users

I had a feeling the missing link was logging(ish)

What's a log?

A 1D list of tuple:

```
list[tuple[datetime, Level, str]]
```

```
[
    (datetime(...), 'info',    'GET /accounts/invoice/ -> 200'),
    (datetime(...), 'info',    'GET /add-on/complete/ -> 200'),
    (datetime(...), 'error',   'Stripe payment failed, insuff...'),
    (datetime(...), 'warning', 'POST /payments/42/complete/ -> 400'),
    (datetime(...), 'warning', 'POST /payments/37/complete/ -> 400'),
    (datetime(...), 'debug',   'Slow query detected SELECT...'),
]
```

This is what code looked like...

until about 1980

How it should be

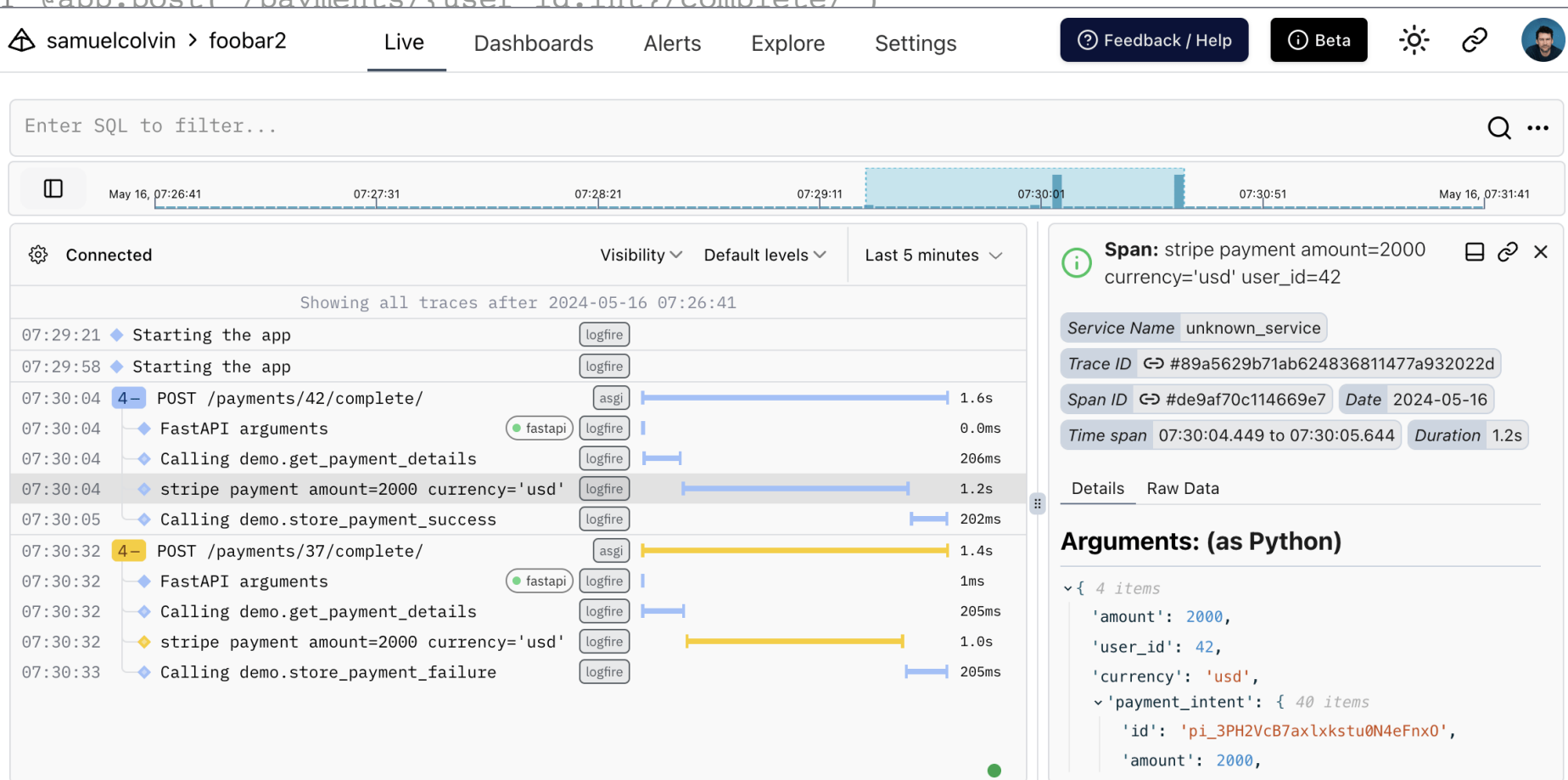
A nested structure:

```
[
    Log(
        start=datetime(...),
        end=datetime(...),
        message='POST /payments/42/complete/ -> 400',
        children=[
            Log(
                start=datetime(...),
                message='Stripe payment failed, insuff...',
                attributes={'reason': 'insufficent funds', ...}
            ),
        ]
    )
]
```

What we proposing here isn't new, but it's still not available to most developers

So we built it

```
1 @app.post('/payments/{user_id:int}/complete/')
```



C'mon, that's not new

No, but these are:

- Using OpenTelemetry ... Taming OpenTelemetry
- Python magic:
  - Autotracing - aka profiling
  - f-string magic
- GenAI/LLM + general purpose
- SQL

- But most of all: **it's easy to use**

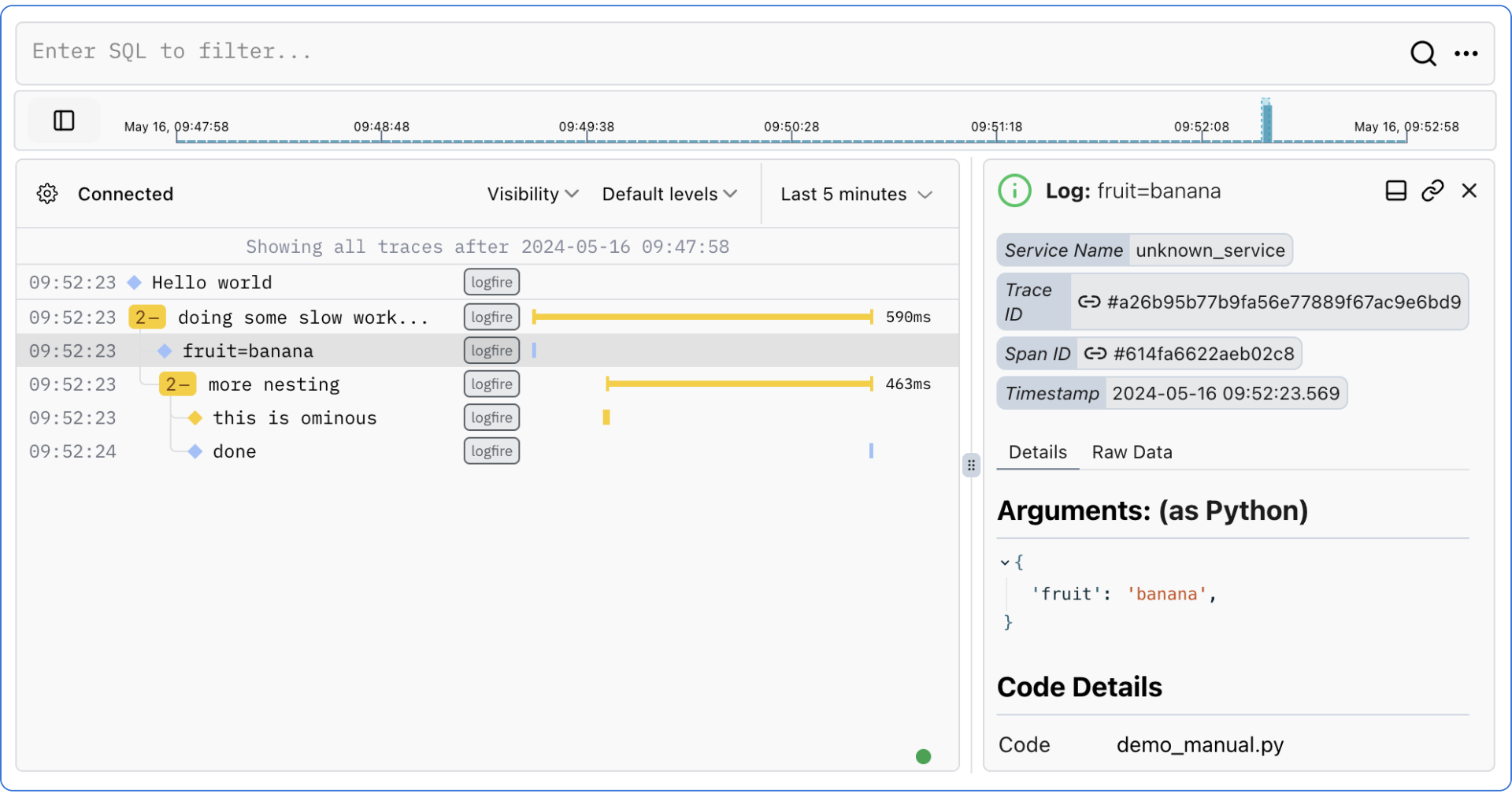
We're the right people to build this because we're not observability professionals

but, we need your help to make it great

Enough theory

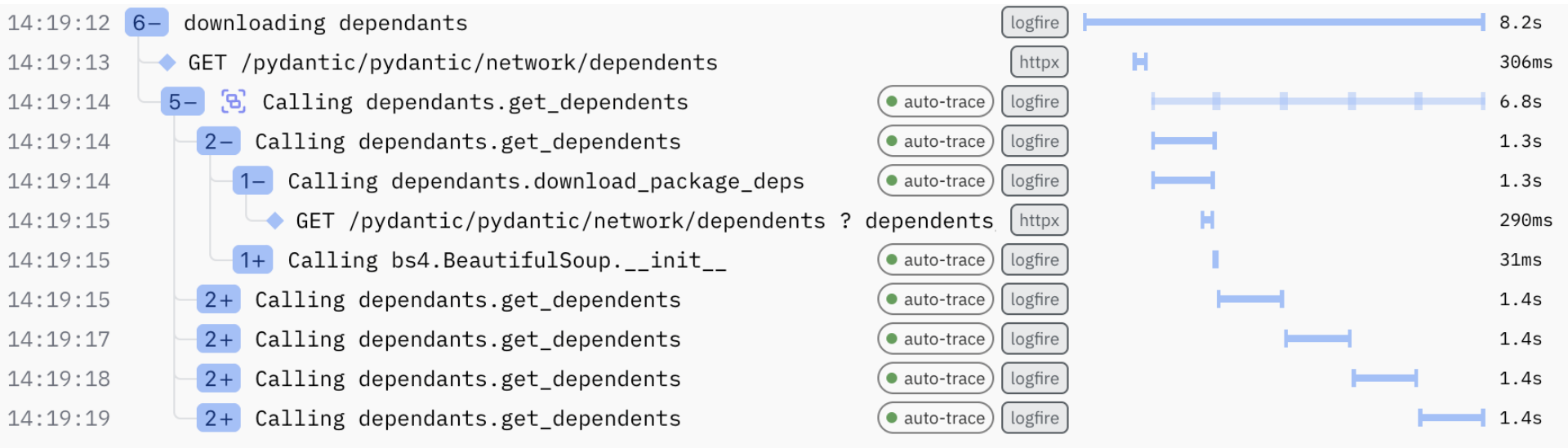
Show me the code!

Manual tracing



Auto tracing

```
logfire.install_auto_tracing(modules=['dependants', 'bs4.*'], min_duration=0.03)
```



Pretty Python objects

Pretty Pyth

```python
from datetime import datetime
from pydantic import BaseModel
import logfire

logfire.configure()

class Delivery(BaseModel):
    timestamp: datetime
    dims: tuple[int, int]

input_json = [
  '{"timestamp": "2020-01-02T03:04
  '{"timestamp": "2020-01-02T04:04
  '{"timestamp": "2020-01-02T05:04
]
deliveries = [
    Delivery.model_validate_json(j
]

logfire.info(f'{len(deliveries)} d
```

Arguments: (as Python)

```
{
  'deliveries': [
    Delivery(
      dims=(
        10,
        20,
      ),
      timestamp='2020-01-02T03:04:05+00:00',
    ),
    Delivery(
      dims=(
        15,
        25,
      ),
      timestamp='2020-01-02T04:04:05+00:00',
    ),
    Delivery(
      dims=(
        20,
        30,
      ),
      timestamp='2020-01-02T05:04:05+00:00',
    ),
  ],
  'len(deliveries)': 3,
}
```

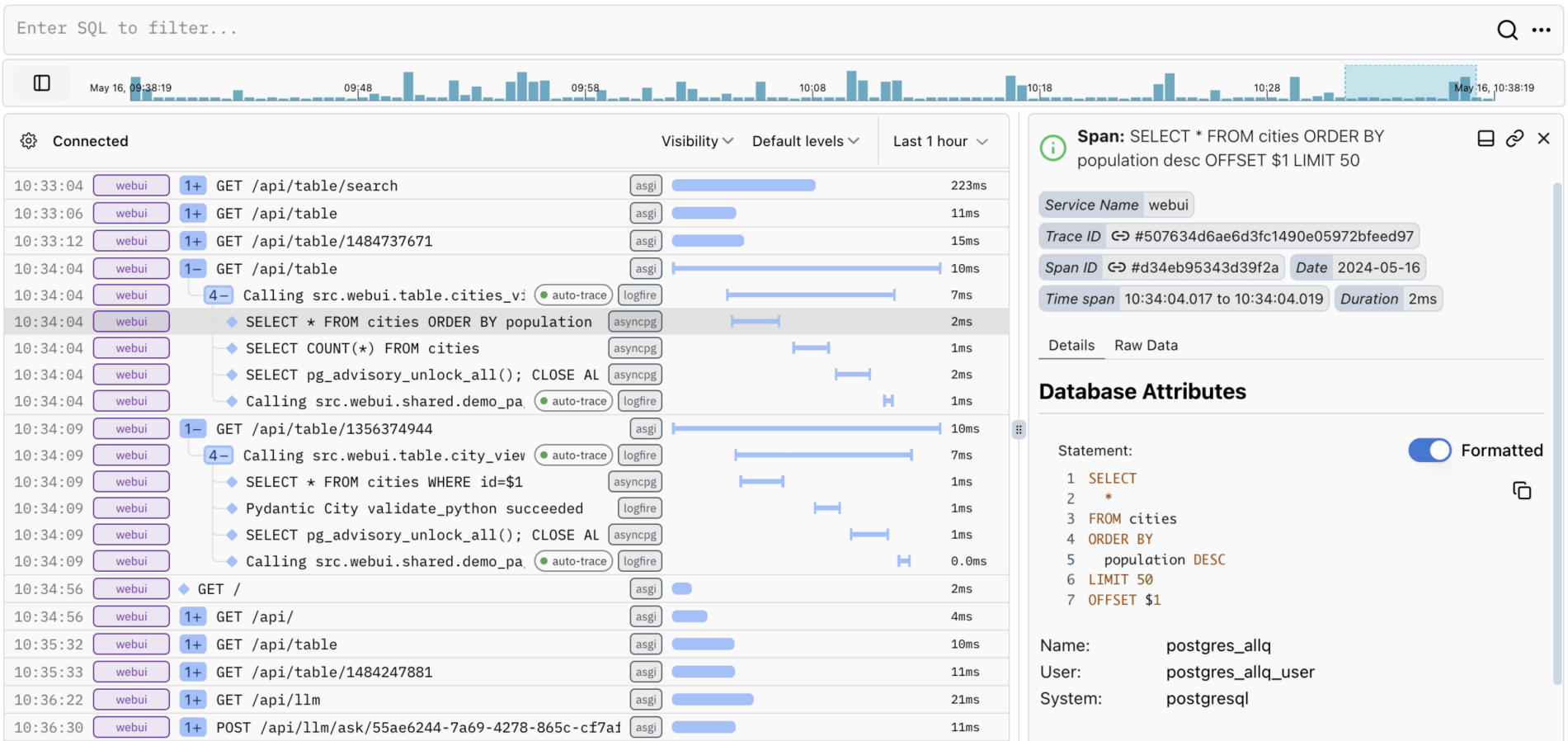
SQLAlchemy
Psycopg
Asyncpg
PyMongo
Sqlite3

Integration

Database

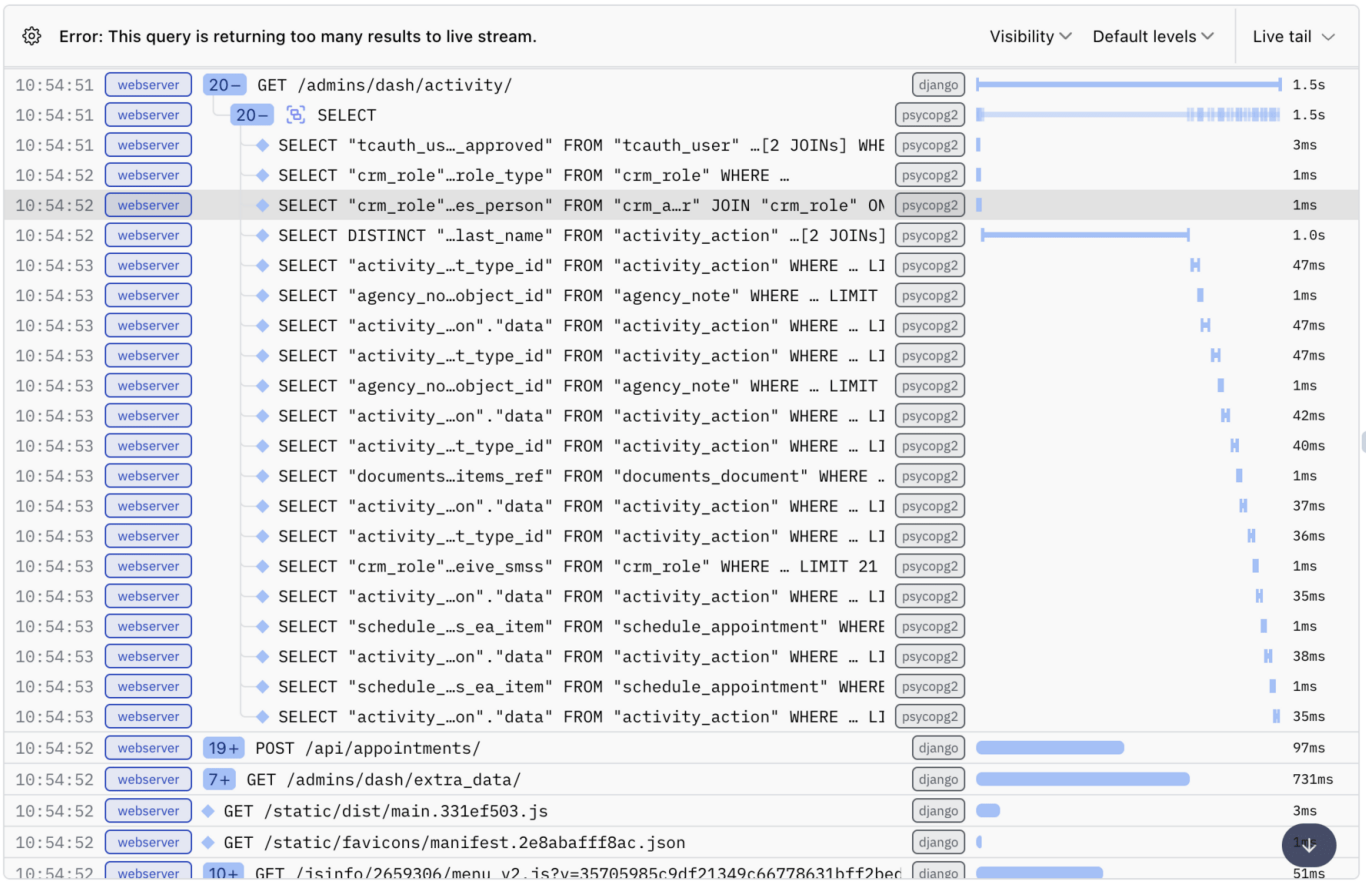
`logfire.instrument_asyncpg()`

Redis
ElasticSearch
MySQL
Kafka
and more...



Integration

Database - Realworld



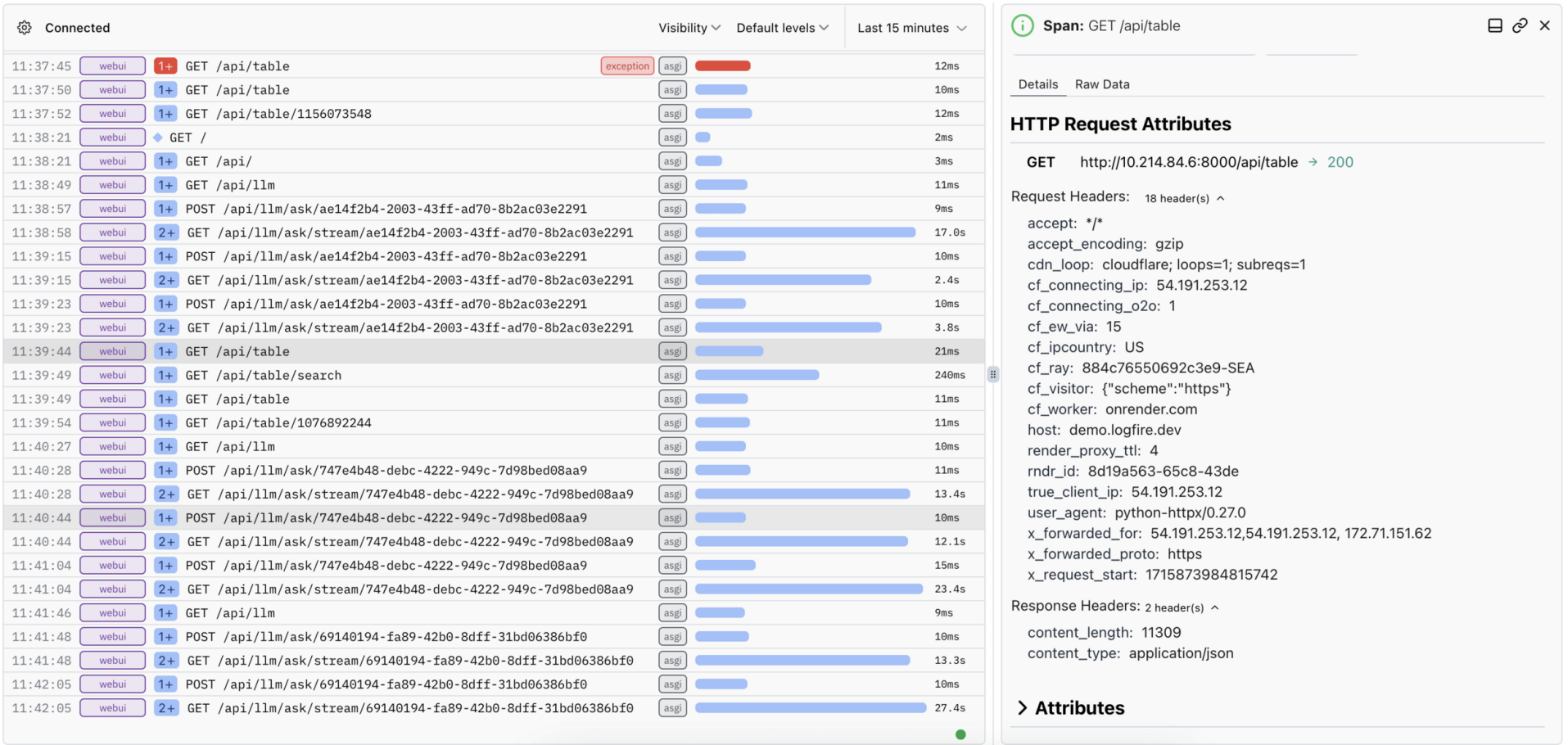
django
flask
starlette
ASGI

Integration

Webframework

AWS Lambda
falcon
tornado
and more...

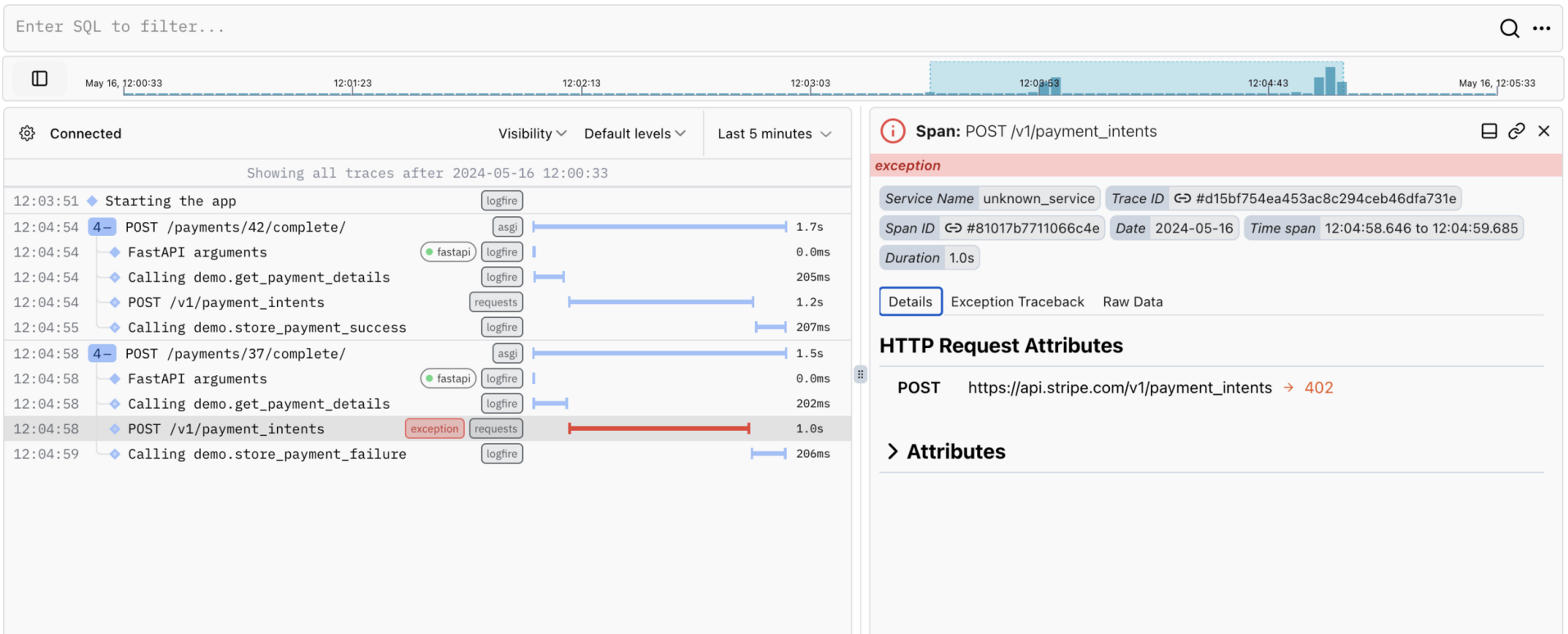
```
logfire.instrument_fastapi(app)
```



httpx
requests
aiohttp

Integration

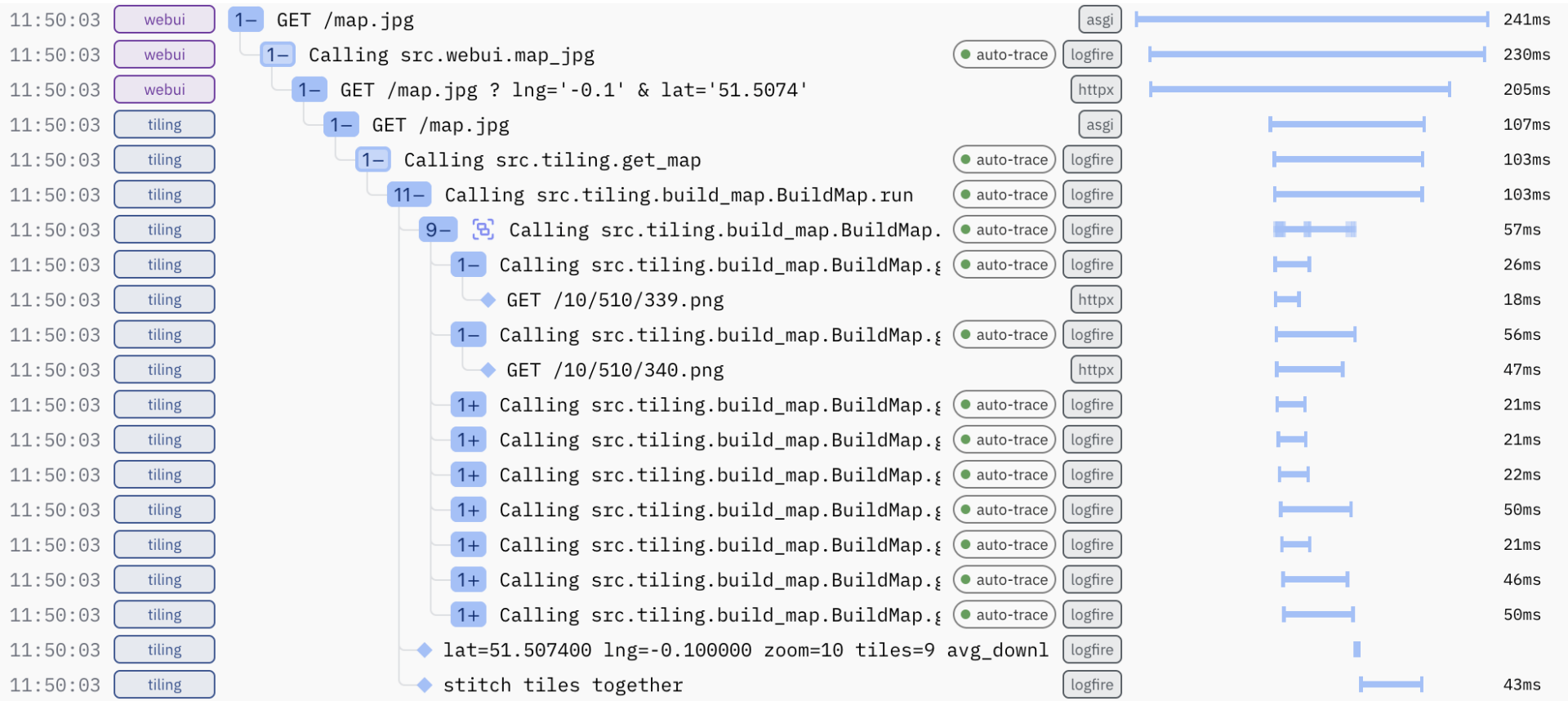
HTTP Requests




```
17          store_payment_failure(user_id)
18          return JSONResponse(content={'detail': 'Card error'},
19                              status_code=400)
20      else:
21          store_payment_success(user_id, intent)
```

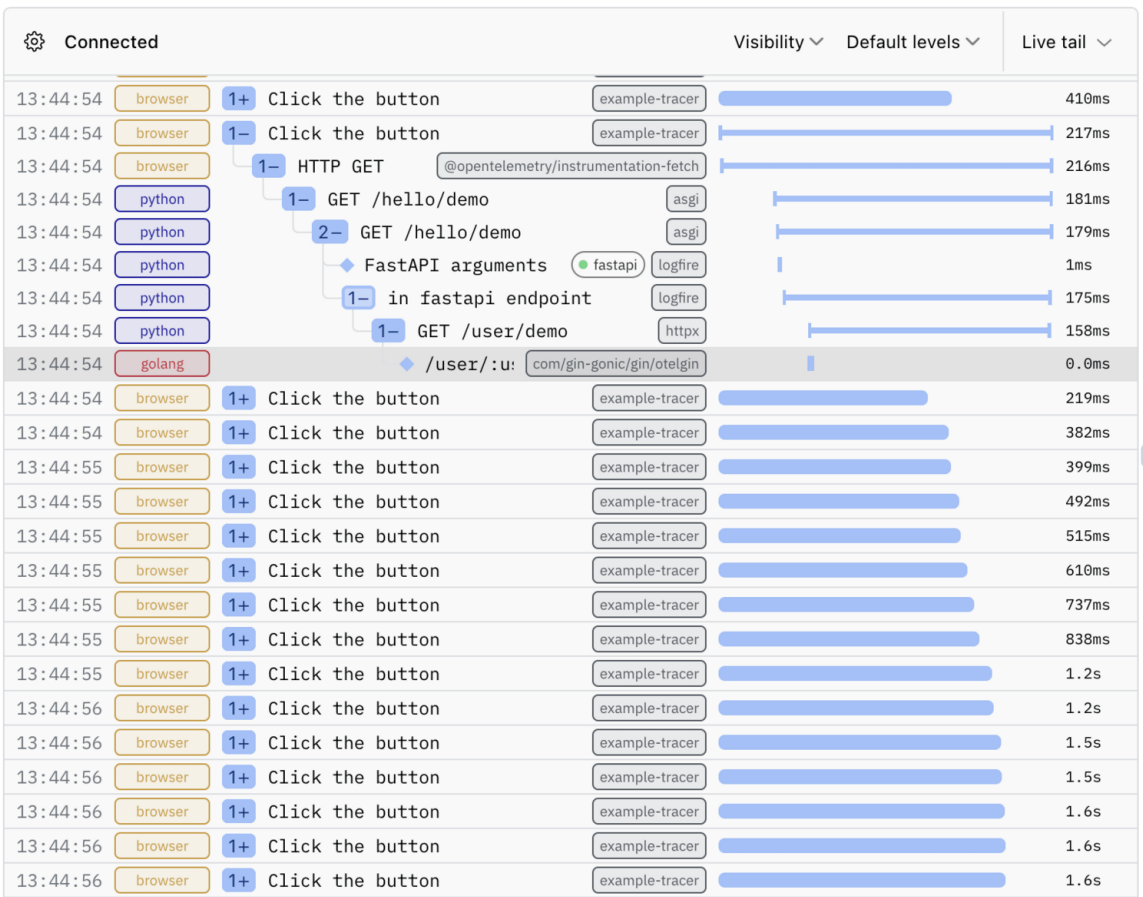
OpenTelemetry win

Distributed Tracing



OpenTelemetry win

Other Languages




```
21          });
22      });
23  };
```

Integration

Pydantic

```
from da
from py
import

logfire

class De
    time
    dims

input_js
    '{"ti
    '{"ti
    '{"ti
]
deliver
    Deli
]
Delivery
    '{"
)
```

12:27:09 Pydantic Delivery validate_json succeeded logfire 1ms
12:27:09 Pydantic Delivery validate_json succeeded logfire 1ms
12:27:09 Pydantic Delivery validate_json succeeded logfire 1ms
12:27:09 Pydantic Delivery validate_json failed logfire 0.0ms

Span: Pydantic Delivery validate_json failed

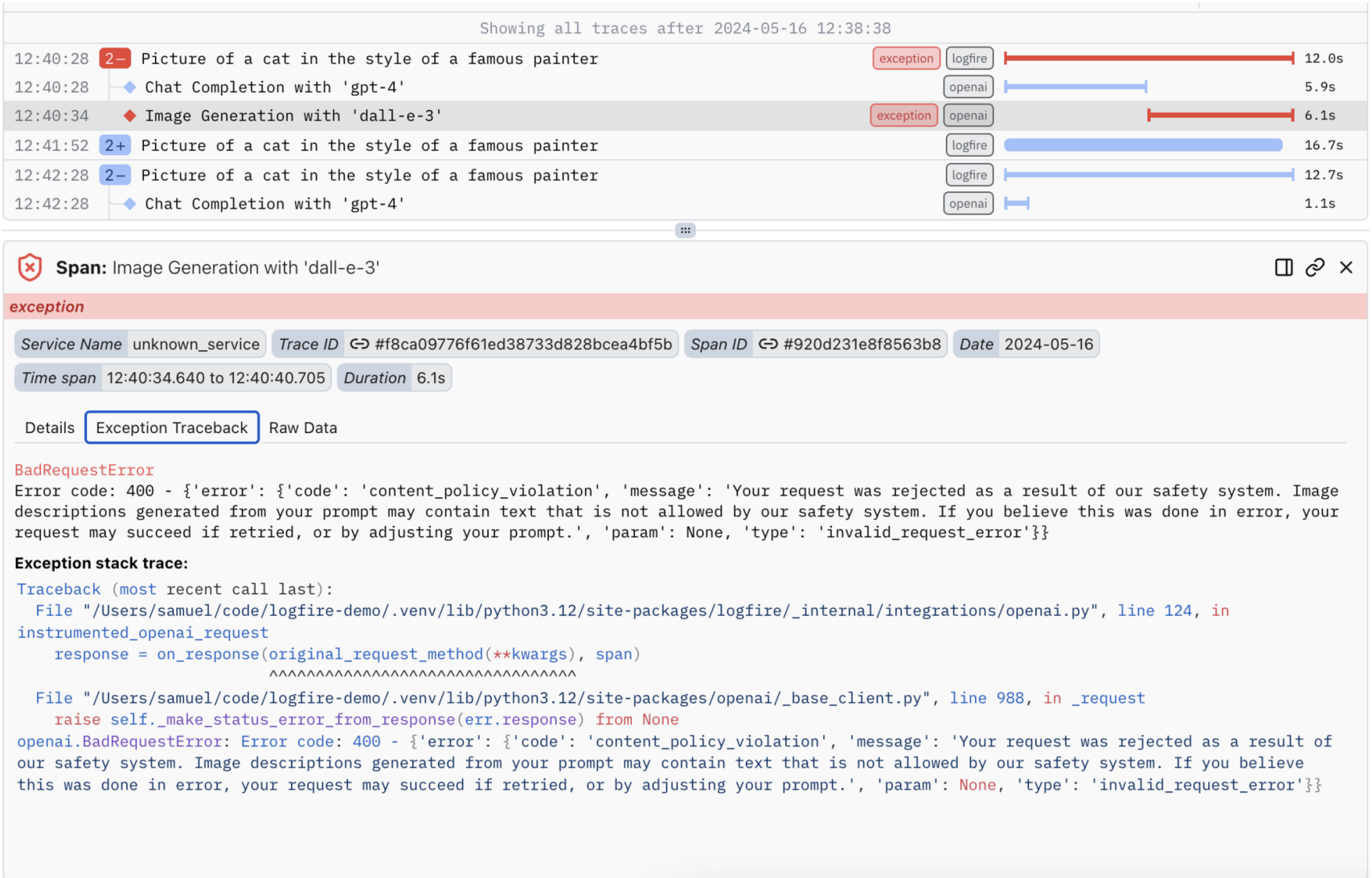
Details | Raw Data

Arguments: (as Python)


```
{ 6 items
  'errors': [
    { 4 items
      'loc': (
        'dims',
        1,
      ),
      'msg': 'Field required',
      'type': 'missing',
      'input': [
        '10',
      ],
    },
  ],
  'success': False,
  'input_data': '{"timestamp": "2020-01-02T03:04:05Z", "dims": ["10"]}',
  'error_count': 1,
  'schema_name': 'Delivery',
  'validation_method': 'validate_json',
```

Integration

OpenAI



Customization

Custom UI

Record data

Reset

```
{
  "attributes": {
    "async": false,
    "request_data": {
      "model": "gpt-4",
      "stream": true,
      "messages": [
        {
          "role": "system",
          "content": "Response entirely in plain text"
        },
        {
          "role": "user",
          "content": "please write a fun fantastical scenario in arou
```

Format

Custom panel function

Reset

```
import { z } from 'zod'

import { CustomPanel, MarkdownText, Section, Image, JsonObject } from './Custom
type Record = any

const toolCallSchema = z.object({
  function: z.object({
    arguments: z.string(),
    name: z.string(),
  }),
})

const messageSchema = z.object({
  role: z.string(),
```

Render

Rendered panels

Show type declarations

LLM Chat Completions

gpt-4

system

Response entirely in plain text

user

please write a fun fantastical scenario in around 100 words

assistant

Tool call: GetCurrentWeather

```
{ 1 items
  "location": "Tokyo"
}
```

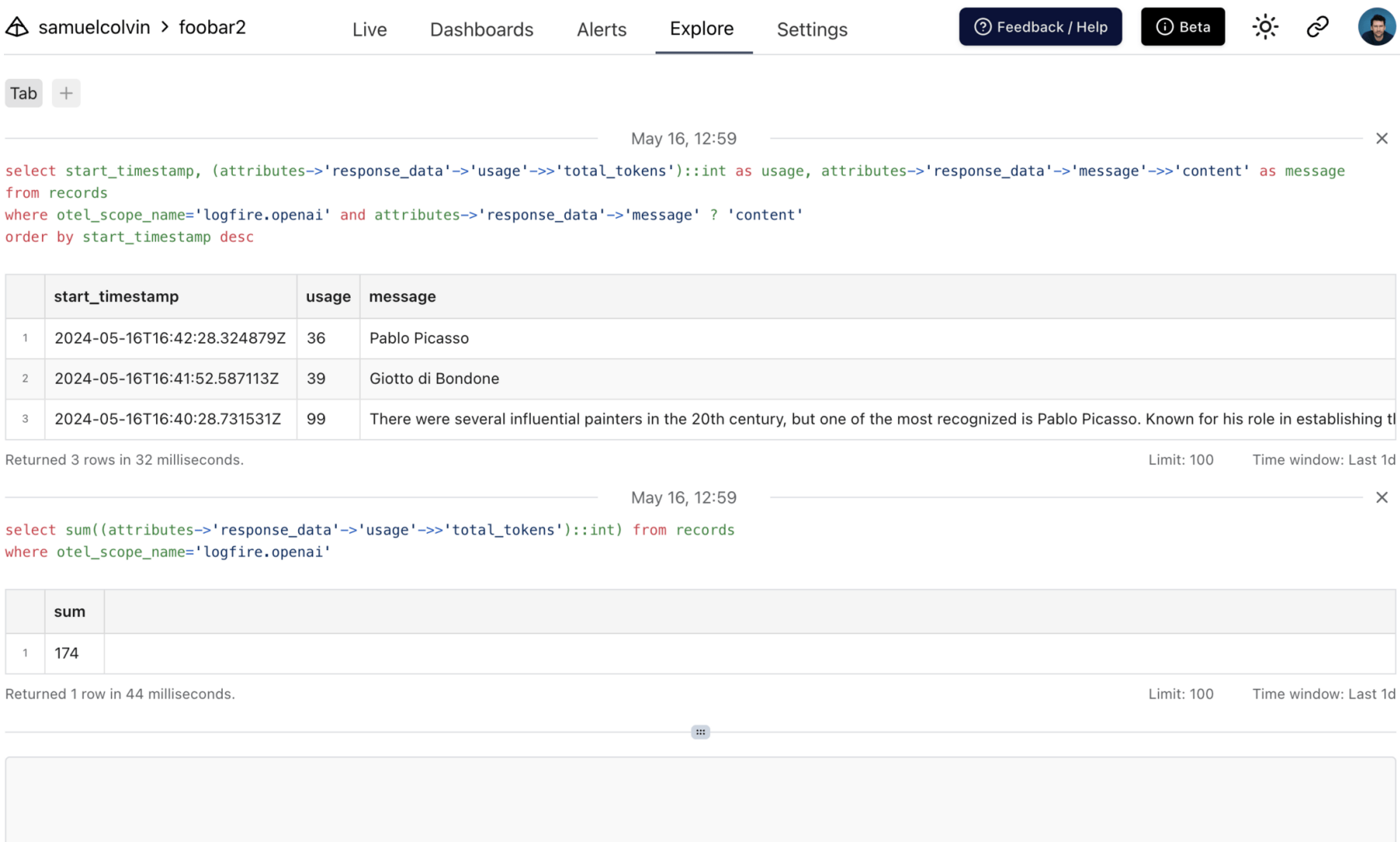
assistant

In the heart of an enchanted forest twinkling with pixie dust, a village of mushroom houses existed. The occupants were tiny fairy folk that rode on dragonflies, each with a thimbleful of magical abilities. One day, they hosted a Great Bake-Off, but the baker, Mr. Finch, couldn't find his magic spatula. Suddenly, a baby unicorn trotted into the village square carrying the lost spatula, a cheeky grin on her face. The fairies cheered and the biggest tart cherry pie was baked and shared, and the unicorn got the first slice for her cheeky service.

LlmImagesPanel would not be rendered for this record

Explore

SQL


samuelcolvin > foobar2
Live
Dashboards
Alerts
Explore
Settings
Feedback / Help
Beta
Tab
May 16, 12:59
select start_timestamp, (attributes->'response_data'->'usage'->>'total_tokens')::int as usage, attributes->'response_data'->'message'->>'content' as message
from records
where otel_scope_name='logfire.openai' and attributes->'response_data'->'message' ? 'content'
order by start_timestamp desc
start_timestamp
usage
message
1
2024-05-16T16:42:28.324879Z
36
Pablo Picasso
2
2024-05-16T16:41:52.587113Z
39
Giotto di Bondone
3
2024-05-16T16:40:28.731531Z
99
There were several influential painters in the 20th century, but one of the most recognized is Pablo Picasso. Known for his role in establishing tl
Returned 3 rows in 32 milliseconds.
Limit: 100
Time window: Last 1d
May 16, 12:59
select sum((attributes->'response_data'->'usage'->>'total_tokens')::int) from records
where otel_scope_name='logfire.openai'
sum
1
174
Returned 1 row in 44 milliseconds.
Limit: 100
Time window: Last 1d

We're here all week, to help you with Pydantic Logfire

And on slack after that

pydantic.dev/logfire

Thank you