tracetest

Observability-driven development with OpenTelemetry

by Adnan Rahić

Kubeshop™

Hi, I’m Adnan

- All things DevRel at Tracetest (Kubeshop)
- Failed startup founder,
  ex-freeCodeCamp leader
- Building open-source dev tools
  for 5+ years

Today I will talk about...

- The pain of testing microservices
- Integration testing and TDD is hard
- How observability-driven development can help
- Observability-driven development in practice

The pain of testing microservices

Here’s a problem you keep facing...

- Don’t know where an HTTP transaction fails
- Can’t track and test microservice-to-microservice communications
- Hard to mock

Here’s how you solve it...

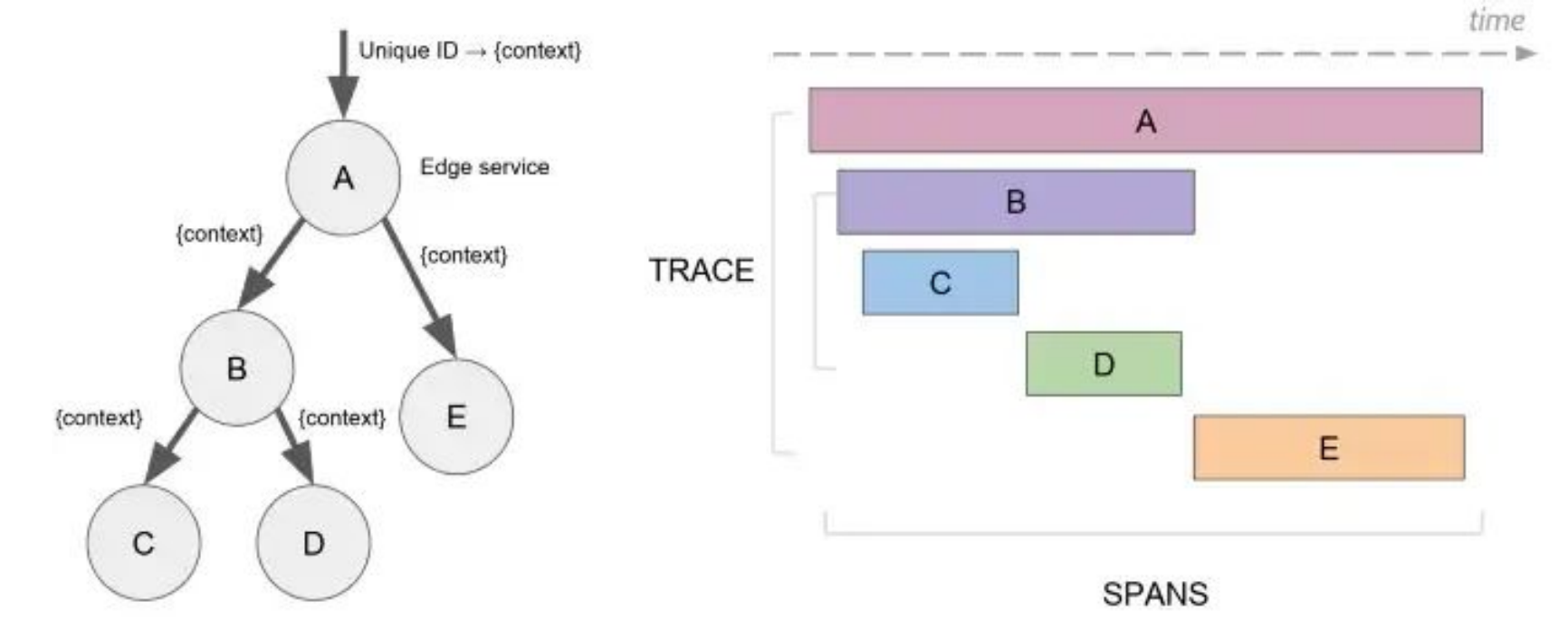
- Observability-driven development
- Distributed traces as test assertions

What is distributed tracing?

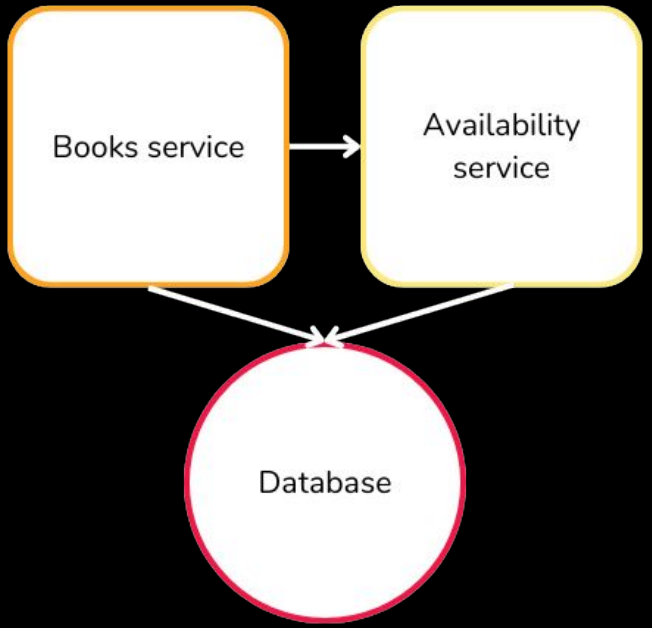
Distributed **tracing** refers to methods of observing **requests** as they propagate through **distributed systems**

– Lightstep

What is distributed tracing?

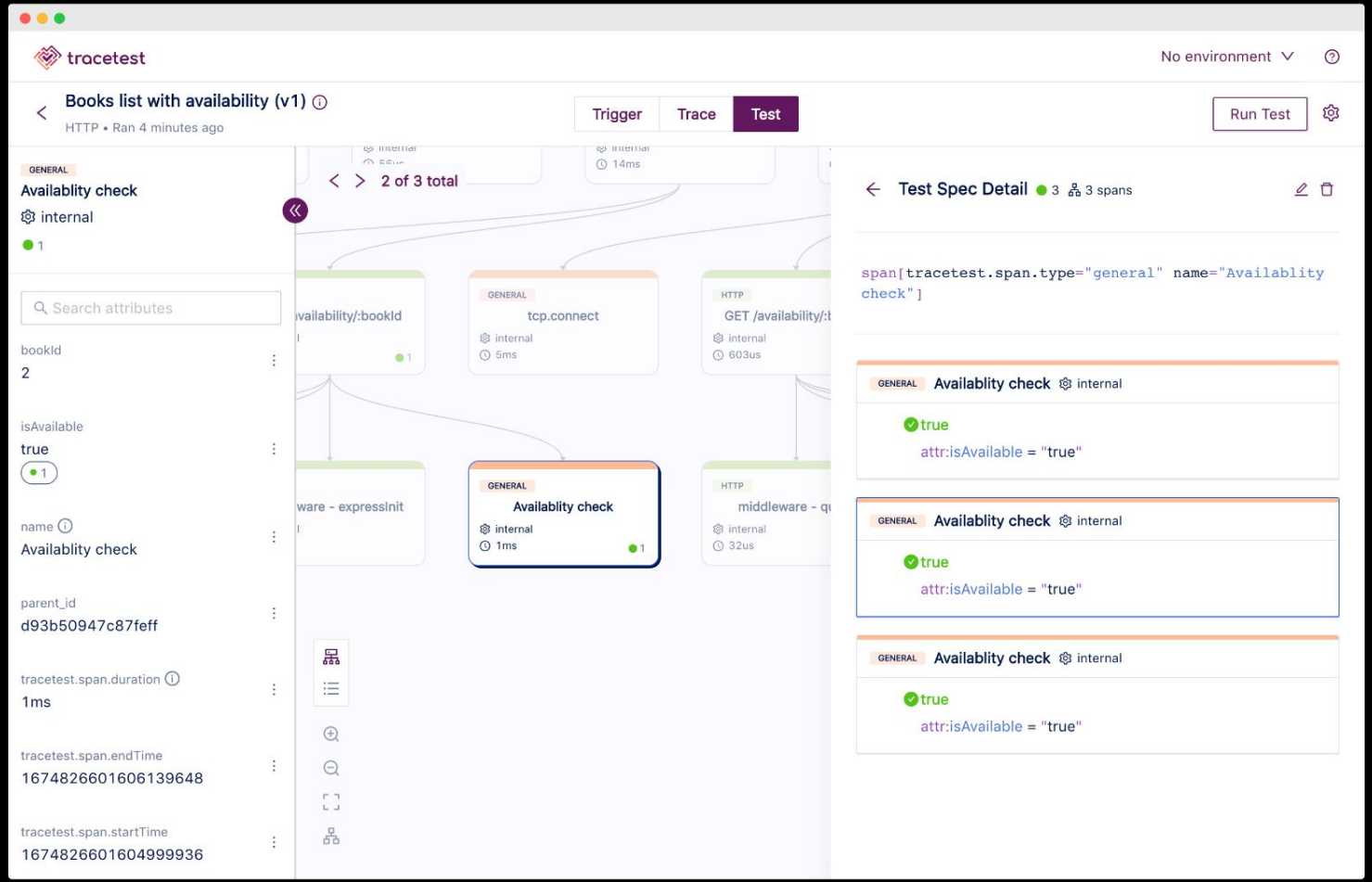


Our distributed system



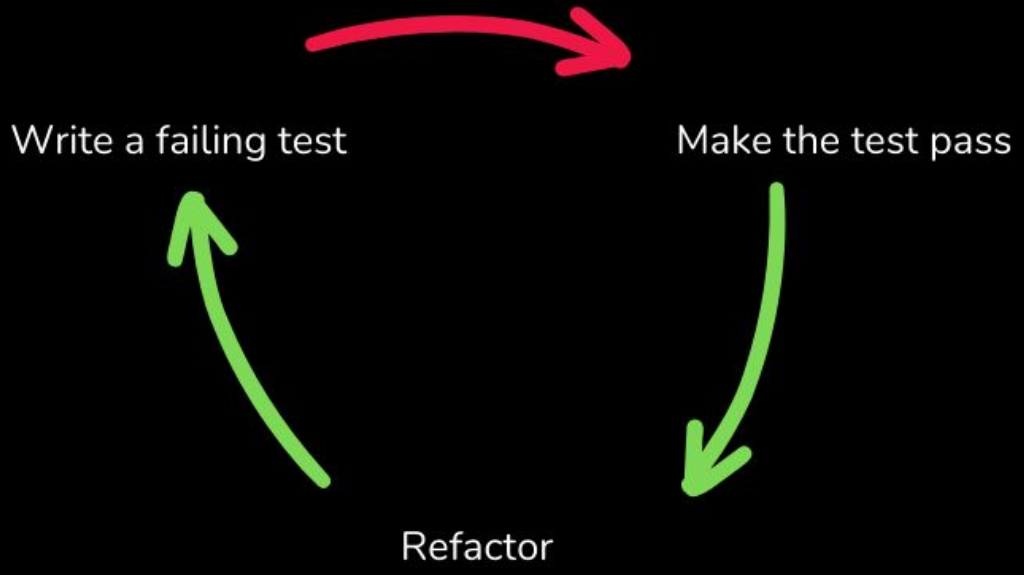
```
app.get('/availability/:bookId', availabilityHandler)

function availabilityHandler(req, res) {
  const span = tracer.startSpan('Availablity check')
  const bookId = req.params.bookId
  span.setAttribute('bookId', bookId)

  const isAvailable = isBookAvailable(bookId)
  span.setAttribute('isAvailable', isAvailable)

  res.json({ isAvailable })
  span.end()
}
```

```
app.get('/availability/:bookId', availabilityHandler)

function availabilityHandler(req, res) {
  const span = tracer.startSpan('Availablity check')
  const bookId = req.params.bookId
  span.setAttribute('bookId', bookId)

  const isAvailable = isBookAvailable(bookId)
  span.setAttribute('isAvailable', isAvailable)

  res.json({ isAvailable })
  span.end()
}
```


tracetest
No environment
Books list with availability (v1)
HTTP • Ran 4 minutes ago
Trigger
Trace
Test
Run Test
GENERAL
Availablity check
internal
1
2 of 3 total
Search attributes
bookId
2
isAvailable
true
1
name
Availablity check
parent_id
d93b50947c87feff
tracetest.span.duration
1ms
tracetest.span.endTime
1674826601606139648
tracetest.span.startTime
1674826601604999936
14ms
GENERAL
tcp.connect
internal
5ms
HTTP
GET /availability/:
internal
603us
GENERAL
Availablity check
internal
1ms
1
HTTP
internal
32us
Test Spec Detail
3
3 spans
span[tracetest.span.type="general" name="Availablity check"]
GENERAL
Availablity check
internal
true
attr:isAvailable = "true"
GENERAL
Availablity check
internal
true
attr:isAvailable = "true"
GENERAL
Availablity check
internal
true
attr:isAvailable = "true"

Integration testing and TDD need help

TDD red-green feedback loop



The pain points of TDD

- Integration tests require access to services and infra 🤔
- Design trigger, access database, auth, propagate envs, monitor message queues, … 🥲
- Can’t track what part of a microservice-chain failed

Just to start TDD-based integration testing...

- Write 90% of code just to make the test work 😕
- 10% of the code is the test itself 🤦‍♂️

Traditional integration test

```
const chai = require('chai')
const chaiHttp = require('chai-http')
chai.use(chaiHttp)
const app = require('../app')
const should = chai.should()
const expect = chai.expect
// ...
```

```
// ...
const booksListMock = require('../mocks/books/books_list.json')

describe('GET /books', () => {
  it('should return a list of books when called', done => {
    chai.request(app).get('/books')
     .end((err, res) => {
       res.should.have.status(200)
       expect(res.body).to.deep.equal(booksListMock)
       done()
    })
  })
})
```

Trace-based test

```yaml
type: Test
spec:
 id: W656Q0c4g
 name: Books List
 description: Try books list
 trigger:
   type: http
   httpRequest:
     url: http://app:8080/books
     method: GET
     # ...
```

```
# ...
specs:
- selector: span[name="GET /books" http.target="/books" ...]
  assertions:
  - attr:http.status_code = 200
- selector: span[name="Books List" ...]
  assertions:
  - attr:books.list.count = 3
```

How observability-driven development can help

What is ODD?

- Write code and observability instrumentation in parallel
- Not testing mocks
- No artificial tests

What is ODD?

- Testing real data from traces in real environments
- Works with your existing OpenTelemetry-based distributed tracing!

What is trace-based testing?

- Add assertions against span values
- Determine if test passed or failed
- Assert against:
  - API response
  - Trace data from spans in the distributed trace

Observability-driven development
in practice

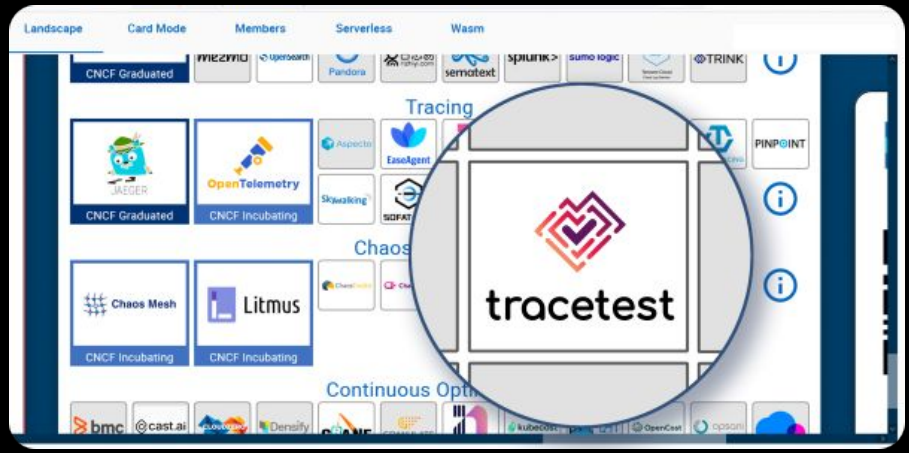
tracetest | 2023

How?

- Tracetest 🪄🎩

What is Tracetest?

- Open-source, CNCF landscape
- Uses OpenTelemetry trace spans as assertions



Why Tracetest?

- Works with existing tracing solutions
  - OpenTelelemetry Collector, New Relic, Lightstep, Jaeger, Tempo, OpenSearch, Elastic, & more...

- Run tests via Web UI and CLI

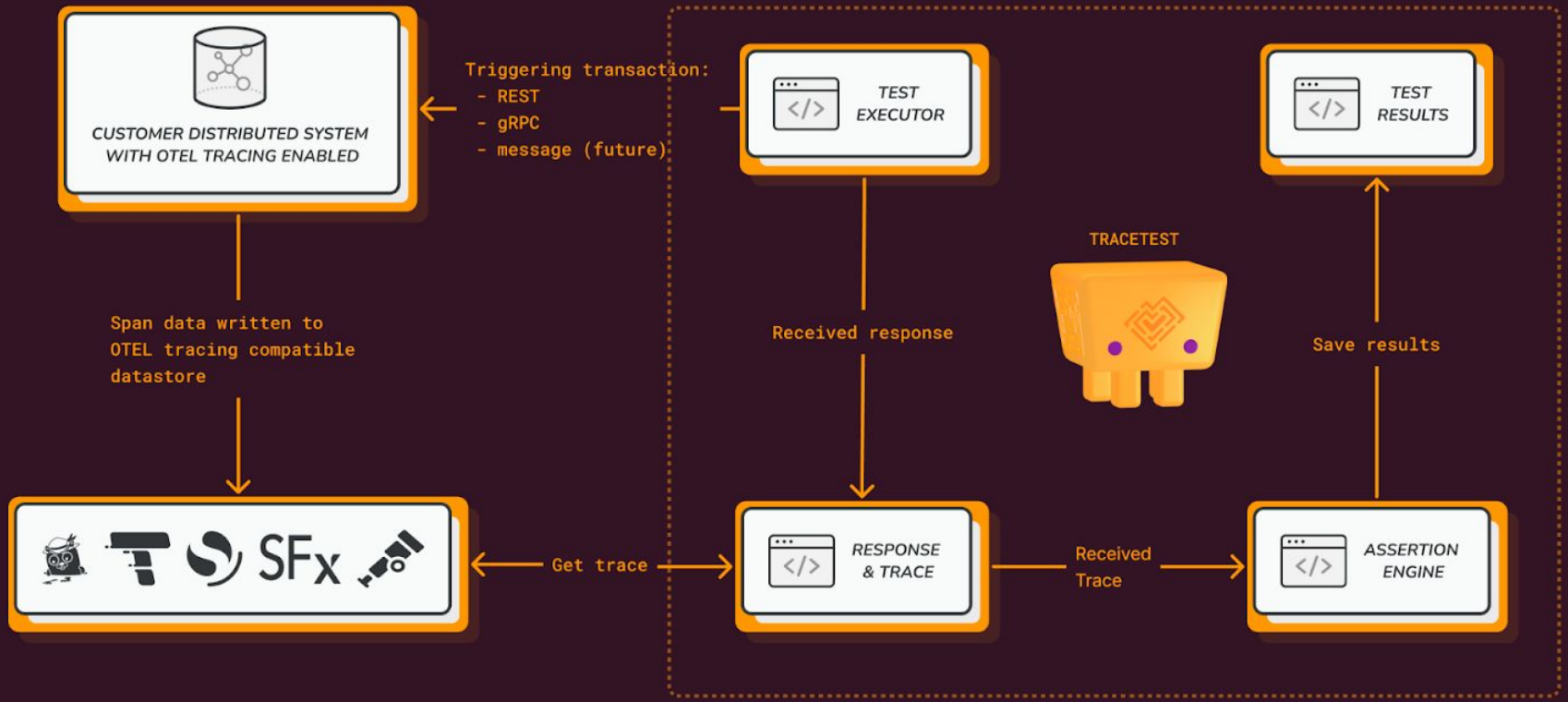
Why Tracetest?

- No artificial tests
- Test against real data
- Transactions for chaining tests into test suites
- Test environments - inputs/outputs between tests

Why Tracetest?

- No mocks
- Testing events of an async message queue
- Assertions based on timing
- Wildcard assertions for common activities

How Tracetest works



Hands-on observability-driven development

tracetest | 2023

```
type: Test
spec:
 id: W656Q0c4g
 name: Books List
 description: Try books list
 trigger:
   type: http
   httpRequest:
     url: http://app:8080/books
     method: GET
     ...
```

```
specs:
- selector: span[name="GET /books" http.target="/books" ...]
  assertions:
  - attr:http.status_code = 200
- selector: span[name="Books List" ...]
  assertions:
  - attr:books.list.count = 3
```

```
app.get('/books', booksListHandler)
function booksListHandler(req, res) {
  const books = getBooks()
  res.json(books)
}

function getBooks() {
  return [
    { id: 1, title: 'Harry Potter' },
    { id: 2, title: 'Foundation' },
    { id: 3, title: 'Moby Dick' },
  ]
}
```

```
$ tracetest test run -d ./http-test.yaml -w

[Output]
✘ Books List
 ✔ span[name="GET /books" http.target="/books" ...]
    ✔ attr:http.status_code = 200 (200)
 ✘ span[name="Books List" ...]
    ✘ attr:books.list.count = 3
```

```
function booksListHandler(req, res) {
  const span = tracer.startSpan('Books List')
  const books = getBooks()
  span.setAttribute('books.list.count', books.length)
  span.end()

  res.json(books)
}
```

```
function booksListHandler(req, res) {
  const span = tracer.startSpan('Books List')
  const books = getBooks()
  span.setAttribute('books.list.count', books.length)
  span.end()

  res.json(books)
}
```

```
$ tracetest test run -d ./test-api.yaml -w

[Output]
✔ Books List (http://localhost:11633/test/g_Qp-iT4g/run/4/test)
```

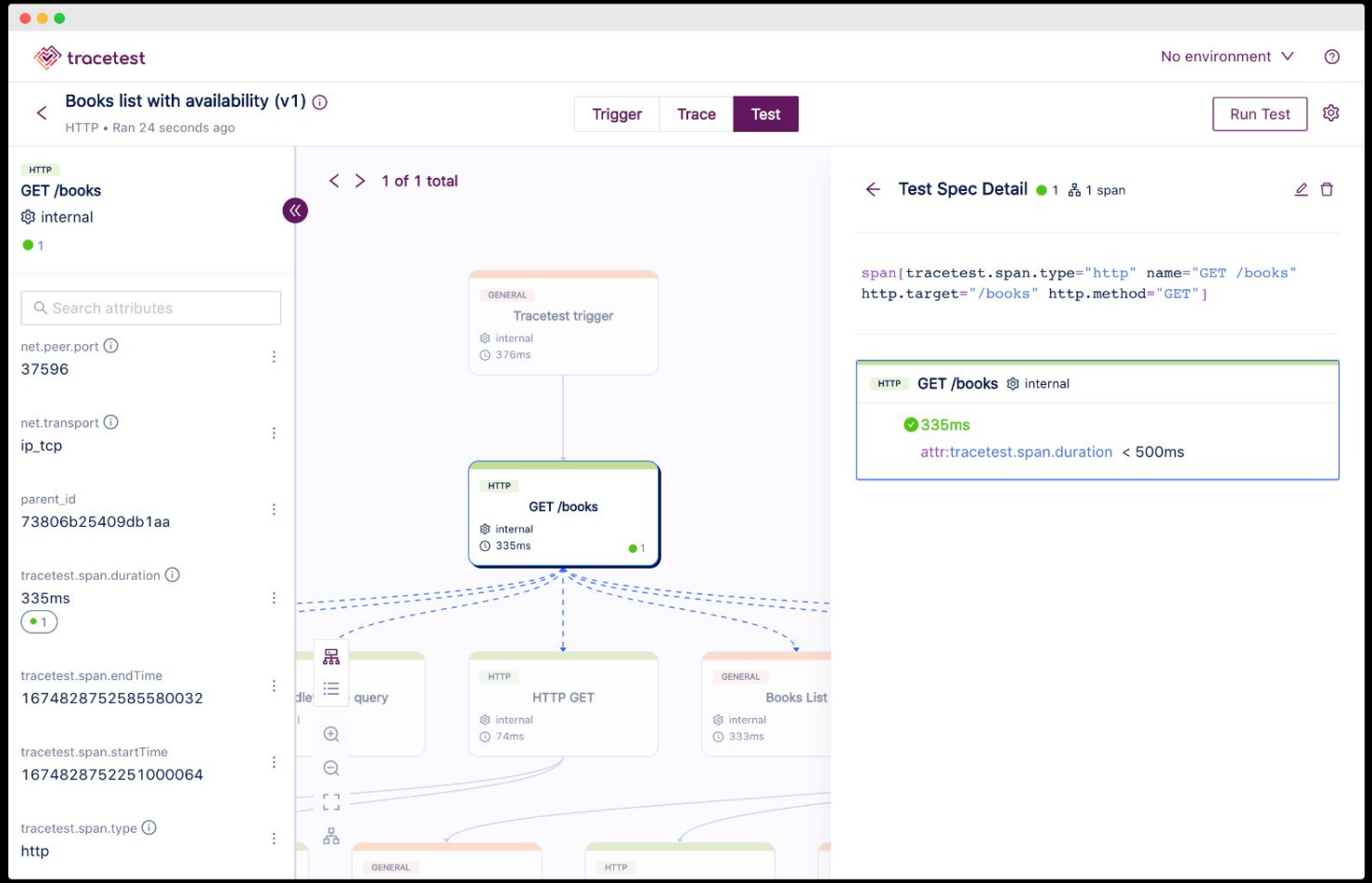
Assert on timing

```
specs:
- selector: span[name="GET /books" http.target="/books" ...]
  assertions:
  - attr:http.status_code = 200
  - attr:tracetest.span.duration < 500ms
- selector: span[name="Books List" ...]
  assertions:
  - attr:books.list.count = 3
```

```
specs:
- selector: span[name="GET /books" http.target="/books" ...]
  assertions:
  - attr:http.status_code = 200
  - attr:tracetest.span.duration < 500ms
- selector: span[name="Books List" ...]
  assertions:
  - attr:books.list.count = 3
```

```
$ tracetest test run -d ./http-test.yaml -w

[Output]
✘ Books List
 ✘ span[name="GET /books" http.target="/books" ...]
     ✔ attr:http.status_code = 200 (200)
     ✘ attr:tracetest.span.duration = 1890ms
 ✔ span[name="Books List" ...]
     ✔ attr:books.list.count = 3
```

```
$ tracetest test run -d ./http-test.yaml -w

[Output]
✘ Books List
 ✘ span[name="GET /books" http.target="/books" ...]
    ✔ attr:http.status_code = 200 (200)
    ✘ attr:tracetest.span.duration = 1890ms
 ✔ span[name="Books List" ...]
    ✔ attr:books.list.count = 3
```


tracetest
No environment
Books list with availability (v1)
HTTP • Ran 24 seconds ago
Trigger
Trace
Test
Run Test
HTTP
GET /books
internal
1
Search attributes
net.peer.port
37596
net.transport
ip_tcp
parent_id
73806b25409db1aa
tracetest.span.duration
335ms
1
tracetest.span.endTime
1674828752585580032
tracetest.span.startTime
1674828752251000064
tracetest.span.type
http
1 of 1 total
GENERAL
Tracetest trigger
internal
376ms
HTTP
GET /books
internal
335ms
1
HTTP
HTTP GET
internal
74ms
GENERAL
Books List
internal
333ms
query
GENERAL
HTTP
Test Spec Detail
1
1 span
span[tracetest.span.type="http" name="GET /books" http.target="/books" http.method="GET"]
HTTP
GET /books
internal
335ms
attr:tracetest.span.duration < 500ms

Assert on every part of an HTTP transaction

```
async function booksListHandler(req, res) {
  const span = tracer.startSpan('Books List')
  const books = await getAvailableBooks()
  span.setAttribute('books.list.count', books.length)
  span.end()

  res.json(books)
}
```

```
async function booksListHandler(req, res) {
  const span = tracer.startSpan('Books List')
  const books = await getAvailableBooks()
  span.setAttribute('books.list.count', books.length)
  span.end()

  res.json(books)
}
```

```
async function getAvailableBooks() {
 const books = getBooks()
 const availableBooks = await Promise.all(
   books.map(async book => {
     const endpoint = `http://availability:8080/${book.id}`
     const { data: { isAvailable } } = await axios.get(`${endpoint}`)
     return { ...book, isAvailable }
   })
 )
 return availableBooks
}
```

```
async function getAvailableBooks() {
 const books = getBooks()
 const availableBooks = await Promise.all(
   books.map(async book => {
     const endpoint = `http://availability:8080/${book.id}`
     const { data: { isAvailable } } = await axios.get(`${endpoint}`)
     return { ...book, isAvailable }
   })
 )
 return availableBooks
}
```

```
app.get('/availability/:bookId', availabilityHandler)

function availabilityHandler(req, res) {
  const span = tracer.startSpan('Availablity check')
  const bookId = req.params.bookId
  span.setAttribute('bookId', bookId)

  const isAvailable = isBookAvailable(bookId)
  span.setAttribute('isAvailable', isAvailable)

  res.json({ isAvailable })
  span.end()
}
```

```
app.get('/availability/:bookId', availabilityHandler)

function availabilityHandler(req, res) {
  const span = tracer.startSpan('Availablity check')
  const bookId = req.params.bookId
  span.setAttribute('bookId', bookId)

  const isAvailable = isBookAvailable(bookId)
  span.setAttribute('isAvailable', isAvailable)

  res.json({ isAvailable })
  span.end()
}
```

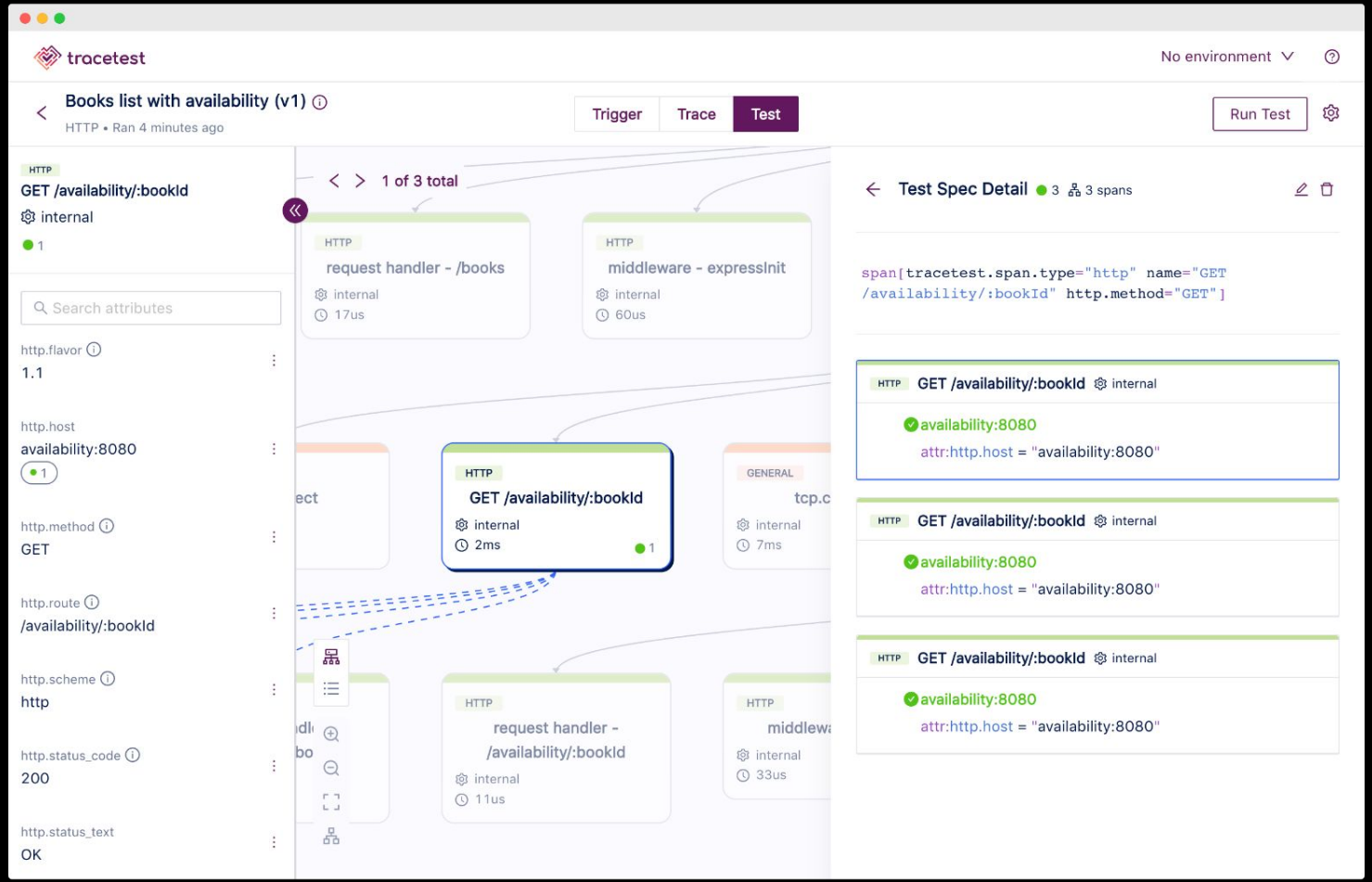
```
function isBookAvailable(bookId) {
 const { stock } = getStock().find(book => book.id == bookId)
 return stock > 0
}

function getStock() {
 return [
   { id: 1, stock: 6 },
   { id: 2, stock: 8 },
   { id: 3, stock: 0 }
 ]
}
```

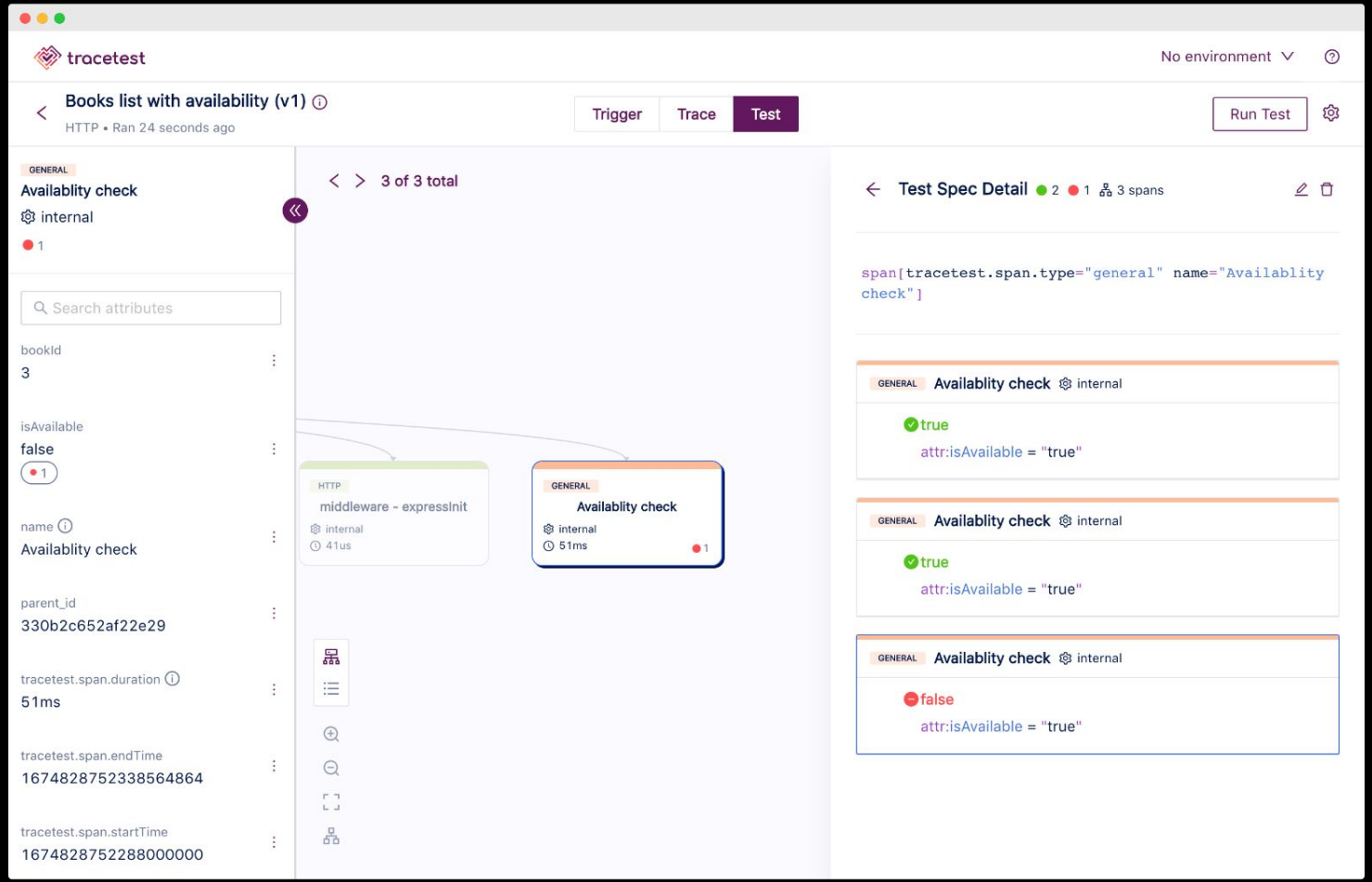
```
function isBookAvailable(bookId) {
 const { stock } = getStock().find(book => book.id == bookId)
 return stock > 0
}

function getStock() {
 return [
   { id: 1, stock: 6 },
   { id: 2, stock: 8 },
   { id: 3, stock: 0 }
 ]
}
```

```
specs:
- selector: span[name="GET /books" http.target="/books"]
  assertions:
  - attr:tracetest.span.duration  < 500ms
- selector: span[name="Books List"]
  assertions:
  - attr:books.list.count = 3
- selector: span[name="GET /availability/:bookId"]
  assertions:
  - attr:http.host = "availability:8080"
- selector: span[name="Availablity check"]
  assertions:
  - attr:isAvailable = "true"
```

```
specs:
- selector: span[name="GET /books" http.target="/books"]
  assertions:
  - attr:tracetest.span.duration  < 500ms
- selector: span[name="Books List"]
  assertions:
  - attr:books.list.count = 3
- selector: span[name="GET /availability/:bookId"]
  assertions:
  - attr:http.host = "availability:8080"
- selector: span[name="Availablity check"]
  assertions:
  - attr:isAvailable = "true"
```

```
$ tracetest test run -d ./http-test.yaml -w

[Output]
✘ Books List
    ✔ span[name="GET /books" http.target="/books"]
          ✔ attr:tracetest.span.duration  < 500ms (19ms)
    ✔ span[name="Books List"]
          ✔ attr:books.list.count = 3 (3)
    ✔ span[name="GET /availability/:bookId" http.method="GET"]
          ✔ attr:http.host = "availability:8080" x3
    ✘ span[name="Availability check"]
          ✔ attr:isAvailable = "true" (true) x2
          ✘ attr:isAvailable = "true" (false)
```

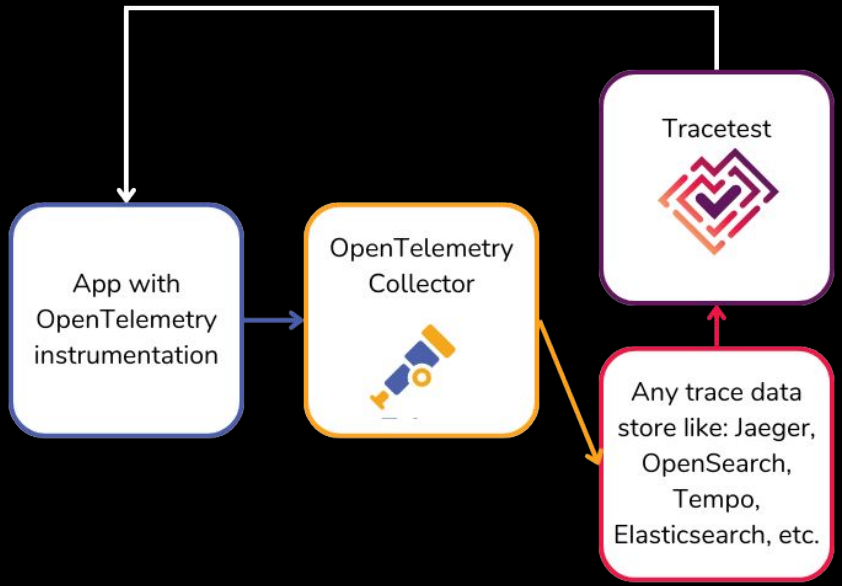

tracetest
No environment
Books list with availability (v1)
HTTP • Ran 4 minutes ago
Trigger
Trace
Test
Run Test
HTTP
GET /availability/:bookId
internal
1
Search attributes
http.flavor
1.1
http.host
availability:8080
1
http.method
GET
http.route
/availability/:bookId
http.scheme
http
http.status_code
200
http.status_text
OK
1 of 3 total
HTTP
request handler - /books
internal
17us
HTTP
middleware - expressInit
internal
60us
HTTP
GET /availability/:bookId
internal
2ms
1
GENERAL
internal
7ms
HTTP
request handler - /availability/:bookId
internal
11us
HTTP
internal
33us
Test Spec Detail
3
3 spans
span[tracetest.span.type="http" name="GET /availability/:bookId" http.method="GET"]
HTTP GET /availability/:bookId internal
availability:8080
attr:http.host = "availability:8080"
HTTP GET /availability/:bookId internal
availability:8080
attr:http.host = "availability:8080"
HTTP GET /availability/:bookId internal
availability:8080
attr:http.host = "availability:8080"


tracetest
No environment
Books list with availability (v1)
HTTP • Ran 24 seconds ago
Trigger
Trace
Test
Run Test
GENERAL
Availablity check
internal
1
3 of 3 total
Search attributes
bookId
3
isAvailable
false
1
name
Availablity check
parent_id
330b2c652af22e29
tracetest.span.duration
51ms
tracetest.span.endTime
1674828752338564864
tracetest.span.startTime
1674828752288000000
HTTP
middleware - expressInit
internal
41us
GENERAL
Availablity check
internal
51ms
1
Test Spec Detail
2
1
3 spans
span[tracetest.span.type="general" name="Availablity check"]
GENERAL
Availablity check
internal
true
attr:isAvailable = "true"
GENERAL
Availablity check
internal
true
attr:isAvailable = "true"
GENERAL
Availablity check
internal
false
attr:isAvailable = "true"

Works with any distributed system with OpenTelemetry instrumentation


App with OpenTelemetry instrumentation
OpenTelemetry Collector
Any trace data store like: Jaeger, OpenSearch, Tempo, Elasticsearch, etc.

Tracetest triggers API, fetches response and trace data, runs assertions



Install Tracetest

- CLI
- Server

Installing Tracetest CLI

```
$ brew install kubeshop/tracetest/tracetest
```

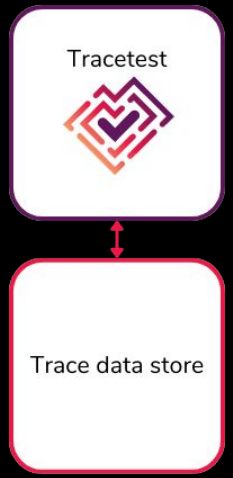
Installing Tracetest Server

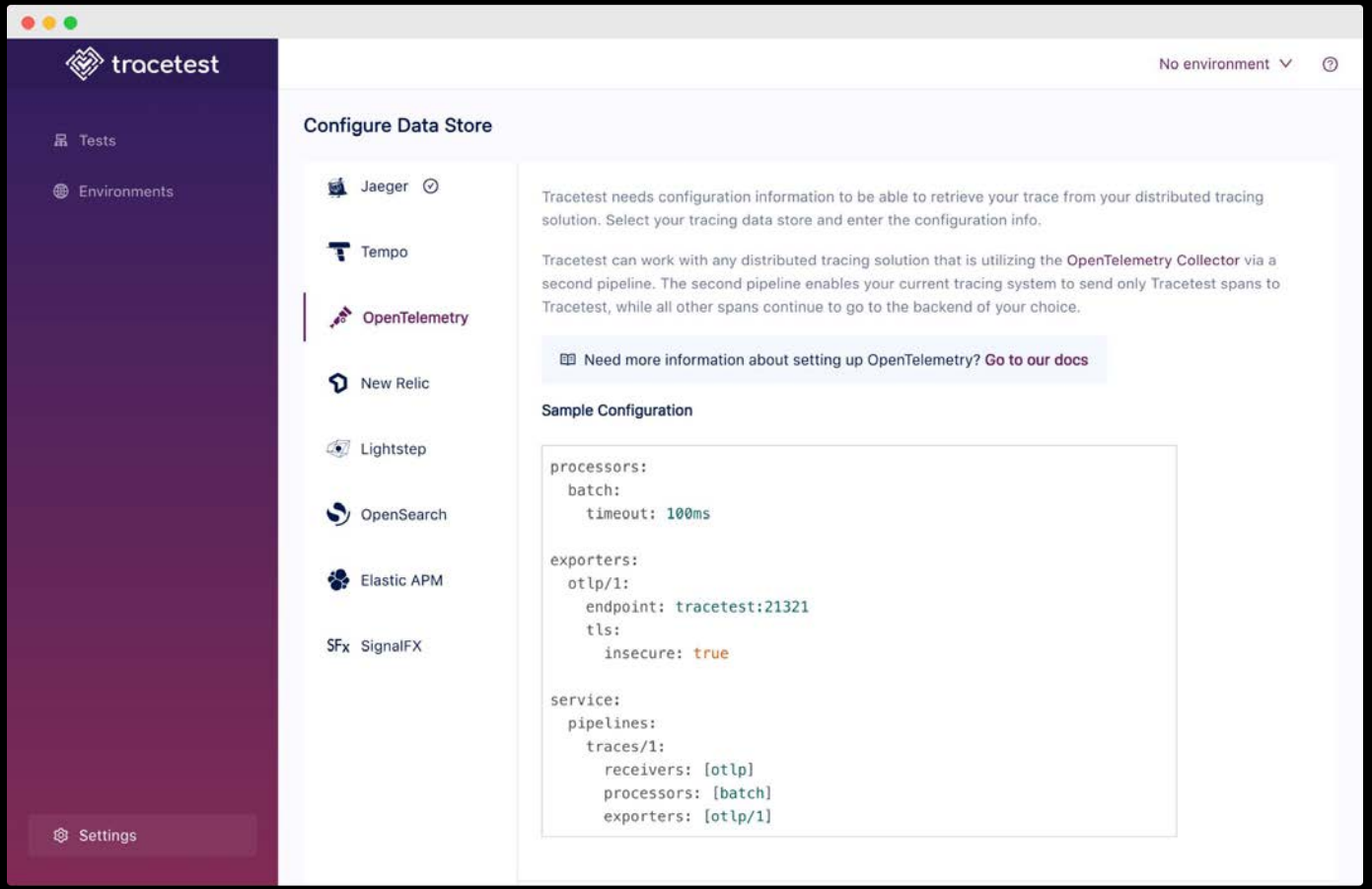
```
$ tracetest server install
```

Supported out-of-the-box

- Docker Compose
- Kubernetes

Connect a trace data store


Tracetest
Trace data store


tracetest
No environment
Tests
Environments
Settings
Configure Data Store
Jaeger
Tempo
OpenTelemetry
New Relic
Lightstep
OpenSearch
Elastic APM
SignalFX
Tracetest needs configuration information to be able to retrieve your trace from your distributed tracing solution. Select your tracing data store and enter the configuration info.
Tracetest can work with any distributed tracing solution that is utilizing the OpenTelemetry Collector via a second pipeline. The second pipeline enables your current tracing system to send only Tracetest spans to Tracetest, while all other spans continue to go to the backend of your choice.
Need more information about setting up OpenTelemetry? Go to our docs
Sample Configuration
processors:
batch:
timeout: 100ms
exporters:
otlp/1:
endpoint: tracetest:21321
tls:
insecure: true
service:
pipelines:
traces/1:
receivers: [otlp]
processors: [batch]
exporters: [otlp/1]

What did we learn?

ODD is awesome!

ODD is awesome!

- No mocks
- Test against real data
- No more black boxes

ODD is awesome!

- Know exactly what's happening in each microservice
- Assert on every step of a transaction

Let’s recap

- Testing on the back end is hard
- Testing distributed systems is hard
- Elevate your TDD with distributed tracing and ODD

Thank you! Any questions?

- Give Tracetest a star on GitHub
  - kubeshop/tracetest
- Give Tracetest a try
  - https://tracetest.io/download
- Read the full blog post
  - https://tracetest.io/blog/the-difference-between-tdd-and-odd

Tracetest community

Github Repository

Discord server

You can find me at...

- @adnanrahic on Twitter and GitHub
- adnan@kubeshop.io


tracetest