OpenTelemetry
an XKCD 927 Success Story

How standards proliferate:
(See: A/C chargers, character encodings, instant messaging, etc.)

Situation:
There are 14 competing standards.

14?! Ridiculous!
We need to develop
one universal standard
that covers everyone's
use cases. Yeah!

Soon:

Situation:
There are 15 competing standards.
Observability
Push Button  
Receive Bacon

https://knowyourmeme.com/memes/push-button-receive-bacon
Observables
Observables

• No bacon.
Observables

• No bacon.
• Warm air coming out.
Observables

• No bacon.
• Warm air coming out.
• Lots of noise.
Observables

• No bacon.
• Warm air coming out.
• Lots of noise.
Observables Responded To Inputs!
Observability

Logs

Metrics

Traces
Observability
Observability
Observability
Observability

- Logs
- Metrics
- Traces
Observability
Traces

• Tell a Story
• Particular Requests
• Tags & Data
• Distributed, Sampled
Metrics

• Tell a Story
• *Types* of Operations
• Time and Tags
Logs

- Tell a Story
- Fine Details
- Structured Data
- Collected/Indexed
(Distributed) Tracing Theory
Phoenix.Controller

Attributes

Key1: Val1
Key2: Val2
Phoenix.Endpoint
Phoenix.Router
Phoenix.Controller
Ecto.Query
Phoenix.View
Jason.encode
Trace

Phoenix.Controller

HTTP.Get  handle_responses  ...

Ecto.Query  ...

HTTP.Get
HTTP.Get

Phoenix.Endpoint

HTTP.Get

Rack.Request
Trace

HTTP.Get

Phoenix.Endpoint

... ... ...

HTTP.Get

Phoenix.Controller

handle_responses

Ecto.Query

Sampled: ✔
State: [...]
Context Propagation

https://www.w3.org/TR/trace-context/

§ 2.2.2 Field value

This section uses the Augmented Backus-Naur Form (ABNF) notation of RFC5234, including the HEXDIG rules from that document.

value = version "-" version-format
version = 2HEXDIG ; this document assumes version 00. Version 255 is forbidden

The value is US-ASCII encoded (which is UTF-8 compliant). Character − is used as a delimiter between fields.

Version (version) is a 1 byte representing an 8-bit unsigned integer. Version 255 is invalid. Current specification assumes the version is set to 00.

The following version-format definition is used for version 00.

version-format = trace-id "-" span-id "-" trace-flags

trace-id = 32HEXDIG ; 16 bytes array identifier. All zeroes forbidden
span-id = 16HEXDIG ; 8 bytes array identifier. All zeroes forbidden
trace-flags = 2HEXDIG ; 8 bit flags. Currently only one bit is used. See below for details
Collector 😐

[Diagram of two data structures, one marked with an 'X' and the other with a '✓']

Greg Mefford (@ferggo) #FOSDEM
Tracer: Head Sampling

Collector 🤔
BEAM Built-In Tracing

(not used for distributed tracing)
erlang:trace_pattern

http://erlang.org/doc/man/erlang.html#trace_pattern-2

erlang:trace_pattern(MFA, MatchSpec) -> integer() >= 0

Types

MFA = trace_pattern_mfa() | send | 'receive'
MatchSpec =
  (MatchSpecList :: trace_match_spec()) |
  boolean() |
  restart |
  pause
trace_pattern_mfa() = {atom(), atom(), arity() | '_' } |
  on_load
trace_match_spec() =
  {{{term()} | '_' | match_variable(), [term()],
  [term()]}}
match_variable() = atom()
  Approximation of '$1' | '$2' | '$3' | ...

The same as erlang:trace_pattern(Event, MatchSpec, []), retained for backward compatibility.

erlang:trace_pattern(MFA :: send, MatchSpec, FlagList :: []) -> OTP 19.0

Types

MatchSpec = (MatchSpecList :: trace_match_spec()) |
  boolean()
trace_match_spec() =
  {{{term()} | '_' | match_variable(), [term()],
  [term()]}}
match_variable() = atom()
  Approximation of '$1' | '$2' | '$3' | ...
Module recon_trace

- Description
- Data Types
- Function Index
- Function Details

recon_trace is a module that handles tracing in a safe manner for single Erlang nodes, currently for function calls only.

Authors: Fred Hebert (mononcqc@ferd.ca) [web site: http://ferd.ca/]

Description

recon_trace is a module that handles tracing in a safe manner for single Erlang nodes, currently for function calls only. Functionality includes:

- Nicer to use interface (arguably) than dbg or trace BIFs.
- Protection against dumb decisions (matching all calls on a node being traced, for example)
- Adding safe guards in terms of absolute trace count or rate-limiting
- Nicer formatting than default traces

Tracing Erlang Code

The Erlang Trace BIFs allow to trace any Erlang code at all. They work in two parts: pid specifications, and trace patterns.

Pid specifications let us decide which processes to trace. They can be specified with Erlang pid or regular expression. If we want to trace all processes on the same node, we can use the phrase all.
Interactive Process

Tracer Process

Trace Patterns

Application Process

Application Process

Application Process

Application Process

erlang:trace, recon_trace, etc.
BEAM

erlang:trace, recon_trace, etc.

Interactive Process

Tracer Process

Trace Patterns

Application Process
Application Process
Application Process
Application Process
**Prod-Safe**

**Interactive**

**Per BEAM**

**Local Only**

---

**Module recon_trace**

- Description
- Data Tracing
- Function Index
- Function Details

**Description**

`recon_trace` is a module that handles tracing in a safe manner for single Erlang nodes, currently for function calls only. Functionality includes:

- Nicer to use interface arguably than Dig or trace BIFs.
- Protection against dumb decisions (noting all calls on a node being traced, for example)
- Adding code snippets in terms of absolute trace count or rate-limiting
- Nicer formatting than default traces

**Tracing Erlang Code**

The Erlang Trace BIFs allow to trace any Erlang code at all. They work in two parts: pid specifications, and trace patterns.

PId specifications let you decide which processes to target. They can be specific pids, all pids, existing pids, or new pids (those not spawned at the time of the function call).

The trace patterns represent functions. Functions can be specified in two parts: specifying the modules, functions, and arguments, and then with Erlang match specifications to add constraints to arguments (see `match/2` for details).

**Erlang**


Version 10.3

User's Guide
Reference Manual
Release Notes
PDF
Top

Table of Contents

- erl_prim_loader
  - erl
  - Top of manual page
  - abs/0
  - abs/2/2
  - abs/3/2
  - abs/2/3
  - append_element/2
  - apply/2
  - apply/3
  - atom_to_binary/2
  - atom_to_list/1

---

```
erlang:trace_pattern(MFA, MatchSpec) -> integer() >= 0

Types

MFA = trace_pattern_mfa() | send | 'receive'
MatchSpec =
  (MatchSpecList :: trace_match_spec()) |
  boolean() |
  restart |
  pause
trace_pattern_mfa() = (atom(), atom(), arity() = 'a') |
on_load
trace_match_spec() =
  [[[term()] | '_-' | match_variable(), [term()]],
  match_variable() = atom()
  Approximation of $1$ 's $2$ 's $3$ ...

The same as `erlang:trace_pattern(Event, MatchSpec, [])`, retained for backward compatibility.

erlang:trace_pattern(MFA :: send, MatchSpec, FlagList :: []) -> integer() >= 0

Types

MatchSpec = (MatchSpecList :: trace_match_spec()) |
  boolean() |
trace_match_spec() =
  [[[term()] | '_-' | match_variable(), [term()]],
  match_variable() = atom()
  Approximation of $1$ 's $2$ 's $3$ ...

Sets trace pattern for message sending. Must be combined with `erlang:trace/3` to set the send trace flag for one or more processes. By default all messages sent from send traced processes are traced. To limit traced send events based on the message content, the sender and/or the receiver, use `erlang:trace_pattern/3`.
```
Observability
Super-Powers
Single-Request Flame Graphs
Stack-Traces

In Context
N+1 Queries
```
SELECT p0."title" FROM "posts" AS p0 WHERE (p0."id" = $1)
```
SELECT p0."id" FROM "posts" AS p0 WHERE (p0."user_id" = $1)

{  
  param_count 1
  sql  
  db phoenix_backend_repo  
  rows 1
}
Calls to the Same Service
Async
Opportunities
Network Latency
Culture Shift
Distributed Tracing Pitfalls
Sampling
Incomplete Traces
Clock Skew
OpenTelemetry (and Open\{Census, Tracing\})
# OpenTracing

**Cloud Native Computing Foundation (CNCF)**

Observability and Analysis - Tracing

**OpenTracing API for Go**

<table>
<thead>
<tr>
<th>Website</th>
<th><a href="https://opentracing.io/">https://opentracing.io/</a></th>
</tr>
</thead>
<tbody>
<tr>
<td>Repository</td>
<td><a href="https://github.com/opentracing/opentracing-go">https://github.com/opentracing/opentracing-go</a></td>
</tr>
<tr>
<td>Crunchbase</td>
<td><a href="https://www.crunchbase.com/organization/cloud-native-computing-foundation">https://www.crunchbase.com/organization/cloud-native-computing-foundation</a></td>
</tr>
<tr>
<td>LinkedIn</td>
<td><a href="https://www.linkedin.com/company/cloud-native-computing-foundation">https://www.linkedin.com/company/cloud-native-computing-foundation</a></td>
</tr>
<tr>
<td>Twitter</td>
<td>@opentracing</td>
</tr>
<tr>
<td>First Commit</td>
<td>3 years ago</td>
</tr>
<tr>
<td>Contributors</td>
<td>37</td>
</tr>
<tr>
<td>Headquarters</td>
<td>San Francisco, California</td>
</tr>
</tbody>
</table>

[Website](https://opentracing.io)
What is OpenTracing?

It is probably easier to start with what OpenTracing is NOT.

- OpenTracing is not a download or a program. Distributed tracing requires that software developers add instrumentation to the code of an application, or to the frameworks used in the application.

- OpenTracing is not a standard. The Cloud Native Computing Foundation (CNCF) is not an official standards body. The OpenTracing API project is working towards creating more standardized APIs and instrumentation for distributed tracing.

OpenTracing is comprised of an API specification, frameworks and libraries that have implemented the specification, and documentation for the project. OpenTracing allows developers to add instrumentation to their application code using APIs that do not lock them into any one particular product or vendor.

https://opentracing.io/docs/overview/what-is-tracing/
What is OpenTracing?

It is probably easier to start with what OpenTracing is NOT:

- OpenTracing is not a download or a program. Distributed tracing tools are typically designed for developers to add instrumentation to the code of an application, or to instrument an existing application.

- OpenTracing is not a standard. The Cloud Native Computing Foundation (CNCF) does not have an official standards body. The OpenTracing API project is working to create a specification for standardized APIs and instrumentation for distributed tracing.

OpenTracing is comprised of an API specification, frameworks and libraries that have implemented the specification, and documentation for the project. OpenTracing allows developers to add instrumentation to their application code using APIs that do not lock them into any one particular product or vendor.

https://opentracing.io/docs/overview/what-is-tracing/
What is OpenTracing?

It is probably NOT what you think tracing is NOT.

- OpenTracing is not a standard. The Cloud Native Computing Foundation is an official standards body. The OpenTracing API project is working towards creating more standardized APIs and instrumentation for distributed tracing.

OpenTracing is comprised of an API specification, frameworks and libraries that have implemented the specification, and documentation for the project. OpenTracing allows developers to add instrumentation to their application code using APIs that do not lock them into any one particular product or vendor.

https://opentracing.io/docs/overview/what-is-tracing/
What is OpenTracing?

OpenTracing is comprised of an API specification, framework, implementation, and documentation for the project. OpenTracing is NOT an official standard. The Cloud Native Computing Foundation is an official standards body. The OpenTracing API project is maintained by the Cloud Native Computing Foundation, with contributions from the community.

OpenTracing is comprised of an API specification, framework, implementation, and documentation for the project. OpenTracing is not a standard. The Cloud Native Computing Foundation is an official standards body. The OpenTracing API project is maintained by the Cloud Native Computing Foundation, with contributions from the community.

It's an "API Spec."

It's Various Implementations

Spandex, Otter, ExRay, (various others)
Enabling OpenTracing for NGINX

We can tell NGINX to trace every request by adding these two lines to `nginx.conf`:

```plaintext
http {
    opentracing_load_tracer /path/to/tracer/plugin /path/to/tracer/config;
    opentracing on;
    ...
}
```

Now, we'll see the following when admitting a new animal into the zoo:

![Diagram showing request times and traces]

OpenCensus

Easily collect telemetry like metrics and distributed traces from your services

What is OpenCensus?

💻 OpenCensus is a single distribution of libraries that collect metrics and distributed traces from your services

https://opencensus.io
What is OpenCensus?

OpenCensus is a single distribution of libraries that collect metrics and distributed traces from your services.

https://opencensus.io
OpenCensus is a single distribution of libraries that allows collecting distributed traces from your services. It provides a single set of libraries for both metrics and traces, and logs will be added in the future.

What is OpenCensus?

https://opencensus.io
https://opencensus.io/service/
Deployment Options

https://opencensus.io/service/
Tail-Based Sampling

https://opencensus.io/service/
Multi-Output, Chain-able

https://opencensus.io/service/
https://medium.com/opentracing/a-roadmap-to-convergence-b074e5815289
Effective observability requires high-quality telemetry

OpenTelemetry makes robust, portable telemetry a built-in feature of cloud-native software.

OpenTelemetry provides a single set of APIs, libraries, agents, and collector services to capture distributed traces and metrics from your application. You can analyze them using Prometheus, Jaeger, and other observability tools.

https://opentelemetry.io/
Effective observability requires high-quality telemetry. There are many different kinds of telemetry software.

**OpenTelemetry** provides a single set of APIs, libraries, agents, and collector services to capture distributed traces and metrics from your application. You can analyze them using Prometheus, Jaeger, and other observability tools.

https://opentelemetry.io/
https://opentelemetry.io/project-status/
Spandex
A modular/adapter based tracing ecosystem for elixir.

repositories
Repos
People
Teams
Projects

Find a repository...

spandex_phoenix
Phoenix Instrumentation tracer
Elixir 12 7 MIT Updated 10 days ago

spandex_datadog
A datadog adapter for the `spandex` library.
Elixir 5 7 MIT Updated 10 days ago

spandex_ecto
Tools for integrating Ecto with Spandex
Elixir 2 5 MIT Updated 24 days ago

Top languages
Elixir

People
GregMefford
Zach Daniel

https://github.com/spandex-project/
Easy Integration with Phoenix, Plug, and Ecto

https://github.com/spandex-project/
Spandex
A modular/adapter based tracing ecosystem for elixir.

- **spandex_phoenix**
  - Phoenix Instrumentation tracer
  - Elixir

- **spandex_datadog**
  - A datadog adapter for the `spandex` library
  - Elixir

- **spandex_ecto**
  - Tools for integrating Ecto with Spandex
  - Elixir

Only Supports Datadog APM

Implements OpenTracing

https://github.com/spandex-project/
Opencensus.io integrations for Erlang, Elixir, and other BEAM languages

<table>
<thead>
<tr>
<th>Repositories</th>
<th>People</th>
<th>Projects</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

18 results for repositories matching opencensus

Top languages
- Erlang
- Elixir

https://github.com/opencensus-beam/
Opencensus.io integrations for Erlang, Elixir, and other BEAM languages

- Implements OpenCensus
- Supports Various Trace Collectors (including Datadog APM)

https://github.com/opencensus-beam/
Telemetry
(Metrics and Events)
Telemetry

https://github.com/beam-telemetry/telemetry

Simple

Standard

Safe*
% Erlang

% In a library
telemetry:execute([web, request, done],
    #{latency => Latency}, #{status_code => Status})

% In a receiver
telemetry:attach("log-response-handler",
    [web, request, done],
    fun log_response_handler:handle_event/4, [])

# Elixir

# In a library
:telemetry.execute([:web, :request, :done],
    %{latency: latency}, %{status_code: status})

# In a receiver
:telemetry.attach("log-response-handler",
    [:web, :request, :done], &LogResponseHandler.handle_event/4, nil)
% Erlang
-module(log_response_handler).
#include_lib("kernel/include/logger.hrl")

handle_event([web, request, done],
              #{latency := Latency}, #{status_code := Status}, _Config) ->
  ?LOG_INFO("~p sent in ~p", [Status, Latency]).

# Elixir

defmodule LogResponseHandler do
  require Logger

  def handle_event([:web, :request, :done], measurements, metadata, _config) do
    Logger.info("#{metadata.status_code} sent in #{measurements.latency}")
  end
end
% Erlang
-module(log_response_handler).
-include_lib("kernel/include/logger.hrl")

handle_event([web, request, done],
              %{latency := Latency}, %{status_code := Status}, _Config) ->
  ?LOG_INFO("~p sent in ~p", [Status, Latency]).

# Elixir

defmodule LogResponseHandler do
  require Logger

  def handle_event([:web, :request, :done], measurements, metadata, _config) do
    Logger.info("#{metadata.status_code} sent in #{measurements.latency}" )
  end
end
% Erlang
-module(log_response_handler).
-include_lib("kernel/include/logger.hrl")

handle_event([web, request, done],
             #{latency := Latency}, #{status_code := Status}, _Config) ->
             ?LOG_INFO("\~p sent in \~p", [Status, Latency])

# Elixir

defmodule LogResponseHandler do
  require Logger

  def handle_event([:web, :request, :done], measurements, metadata, _config) do
    Logger.info("#{metadata.status_code} sent in #{measurements.latency}")
  end
end
% Erlang
-module(log_response_handler).
(include_lib("kernel/include/logger.hrl")

handle_event([web, request, done],
    #{latency := Latency}, #{status_code := Status}, _Config) ->
    ?LOG_INFO("~p sent in ~p", [Status, Latency]).

# Elixir

defmodule LogResponseHandler do
  require Logger

  def handle_event([:web, :request, :done], measurements, metadata, _config) do
    Logger.info("#{metadata.status_code} sent in #{measurements.latency}"))
  end
end
Simple!
Telemetry
execute

ETS

handler_1

handler_1

handler_2
One Strike; You’re Out!

handler_1

Telemetry

execute

ETS

handler_1
Safe!*
Plug
Phoenix
EctoSQL
Your Library
Standard!
Take-Aways

Instrument your App
Instrument your Libs
Use Telemetry
Learn about OpenTelemetry