Distributed Tracing
- From Theory to Practice -
A Small Favor
What is distributed tracing?
Tracing requests across distributed system boundaries
Wait, distributed systems??
“A distributed system is a collection of independent computers that appear to its users as a single coherent system.”

Andrew S. Tanenbaum and Maarten van Steen
Distributed Systems: Principles and Paradigms
A Simple Use Case
Web Request

Auth Process

Ecommerce Process

User

Orders

Items

Still Rails, but a new app

Original app
Web Request

Auth Process

Ecommerce Process

Orders

Recommendations

Billing

Original app

Python????
Microservices!

(j/k)
Services

(Micro or otherwise)
<insert container joke>
Why do we need distributed tracing?
Internal services look like external APIs
Why is this slow??

Blame data science?
“You can’t tell a coherent macro story about your application by monitoring individual processes”

Ben Seligman
People are bad guessers
How do you tell the story?
Distributed Tracing!
“Distributed tracing commoditizes knowledge”

- Adrian Cole
What’s Stopping You?
Outside the Ruby Wheelhouse
Domain Specific Vocabulary
Fractured Ecosystem
Theory -> Practice
The Basics
Black Box Tracing
Black Box Tracing
Why might this not work for you?
• Need lots of data
• Delayed results
• Can’t guarantee causality
def my_cool_system
  service_1
  service_2
end

def service_1
  Rails.logger "Service 1"
  execute_async_job
end

def execute_async_job
  Rails.logger "Async Job"
end

def service_2
  Rails.logger "Service 2"
end

Aggregated Log

01-01-2001 01:01:01 Service 1
01-01-2001 01:01:02 Async Job
01-01-2001 01:01:03 Service 2
```ruby
def my_cool_system
  service_1
  service_2
end

def service_1
  Rails.logger "Service 1"
  execute_async_job
end

def execute_async_job
  sleep 15
  Rails.logger "Async Job"
end

def service_2
  Rails.logger "Service 2"
end
```

---

**Aggregated Log**

```
01-01-2001 01:01:01 Service 1
01-01-2001 01:01:02 Service 2
01-01-2001 01:01:17 Async Job
```
White Box Tracing
Metadata Propagation
Realtime Analysis
History Lesson
Dapper
Zipkin
“Distributed Tracing”
“So, you want to trace your distributed system?
Key design insights from years of practical experience”

Raja R. Sambasivan, Rodrigo Fonseca, Ilari Shafer, Gregory R. Ganger

Tracing Your Applications
Main Components:

- Tracer
- Transport
- Collector
- Storage
- UI
Tracing Requests
Tracer:
Lives inside your apps, does the tracing
Trace:
The story of a request’s journey through your system
A trace tells this whole story
Span:
Each chapter in that story
A Trace is many
Parent - Child Relationships
Directed Acyclic Graph
Annotations:
Gives us richer insights into our spans
Client Start 01:01:01
Server Receive 01:01:02
Server Send 01:01:03
Client Receive 01:01:04
Trace id 123
Parent id 1
Span id 2

Auth

Client
Send

Server
Receive

Ecommerce
Transporting the Data
Web Request

Auth

Ecommerce

Orders

Recommendations

Billing

Transport

Trace id 123
Parent id nil
Span id 1

Collector

Storage
Propagates ids in band
Reports out of band
Viewing the Data
http://opentracing.io/documentation/
client transaction from start to end

load balancer transaction from start to end

authorization

billing

resource allocation and provisioning

container start-up

storage allocation

start-up scripts
All happening inside the “resource allocation & provisioning”
A widening gap here could indicate queueing
Tracing Incoming Requests
Rack!
```ruby
class RackApp
  def call(environment)
    ['200',
     {'Content-Type' => 'text/html'},
     ['Hello world']
    ]
  end
end
```

Takes an environment hash

Responds to .call()

Returns:
- status
- header
- body
class TracingRackMiddleware
  def initialize(app)
    @app = app
  end

  def call(env)
    @app.call(env)
  end
end
class TracingRackMiddleware
  def initialize(app)
    @app = app
  end

  def call(env)
    trace do  # Trace some stuff
      @app.call(env)
    end
  end
end
class TracingRackMiddleware
  def initialize(app)
    @app = app
  end

  def call(env)
    trace do
      @app.call(env)
    end
  end

  def trace(env, &block)
    span = Span.new("authentication", generate_span_id)
    span.record(SERVER_RECV) ← Received a request
    status, headers, body = yield ← Execute our rack app
    ensure
      span.record(SERVER_SEND) ← Sending back to the client
    end
  end
end

Non-pseudocode version:
# config/initializers/tracing.rb
Rails.application.config.middleware.use TracingRackMiddleware, {
  # some configuration
}

Use our middleware!
Rails.application.config.middleware.use TracingRackMiddleware, {
  service_name: "SERVICE_DOMAIN_NAME",
  service_port: 443,
  sample_rate: ENV.fetch("ZIPKIN_SAMPLE_RATE", 0.1).to_f,
  json_api_host: ENV["ZIPKIN_HOST"]
}

Sample a portion of requests
Tracing Outgoing Requests
More Middleware!
Faraday
class TracingFaradayMiddleware
  def initialize(app)
    @app = app
  end

  def call(env)
    trace!(env) do |env|
      @app.call(env)
    end
  end
end
class TracingFaradayMiddleware
  def initialize(app)
    @app = app
  end

  def call(env)
    trace!(env) do |env|
      @app.call(env)
    end
  end

  def trace!(env, &block)
    env = set_headers(env)  # Manipulate the headers
    span = Span.new("external_call", 1234)
    span.record(Trace::Annotation::CLIENT_SEND)
    status, headers, body = yield env
    ensure
      span.record(Trace::Annotation::CLIENT_RECV)
    end
  end
end
Each of these colors represents an instrumented application.
class TracingFaradayMiddleware
  def initialize(app)
    @app = app
  end

  def call(env)
    trace!(env) do |env|
      @app.call(env)
    end
  end

  def trace!(env, &block)
    env = set_headers(env)
    span = Span.new("external_call", 1234)
    span.record(Trace::Annotation::CLIENT_SEND) # Client Send
    status, headers, body = yield env
    ensure
      span.record(Trace::Annotation::CLIENT_RECV) # Client Receive
    end
  end
end
def self.client
  Faraday.new(url: base_url) do |connection|
    connection.use TracingFaradayMiddleware
    connection.adapter Faraday.default_adapter
  end
end
Checklist
Buy, Build, or Adopt
Buy?
Lightstep
TraceView...
and more?
Adopt an OSS Solution?
Zipkin
What about Open Tracing?
Standardizes Instrumentation
Where is OpenTracing at today?
Interoperability is Still Messy
“Ruby Support”
Rinse and Repeat
Build Your Own?
What are other folks doing?
End-to-End Tracing: Adoption and Use Cases

Jonathan Mace, Brown University

• 15 using Zipkin
• 9 using internal solutions
• 1 using other OSS solution
• 1 using paid solution

Jonathan Mace, Brown University

Infra Requirements and Limitations
Dependency matrix of:
- Tracer
- Transport Layer
- Collection Layer
- Storage Layer
Installing a Separate Agent
Authentication
Missing Authentication & Authorization
{ "buildpacks": [ 
{ 
  "url": "https://github.com/heroku/heroku-buildpack-apt"
}, 
{ 
  "url": "https://github.com/danp/heroku-buildpack-runit"
} 
] }
Advanced Package Tool (Apt)
Runit
Client Authorization
Basic auth via htpasswd

https://www.nginx.com/resources/admin-guide/restricting-access-auth-basic/
# config/initializers/zipkin.rb

```ruby
Rails.application.config.middleware.use ZipkinTracer::RackHandler, {
  service_name: "test.example.com",
  service_port: 443,
  json_api_host: ENV['ZIPKIN_HOST']
}
```

```
ENV['ZIPKIN_HOST'] = "https://username:password@my-zipkin.com"
```

Our app’s configuration file

Where we’re sending traces

Uses Basic Auth
Browser Authentication
bit.ly’s Oauth2 proxy

https://github.com/bitly/oauth2_proxy
Browser

Nginx (SSL Termination) $PORT

Authorized Herokai?

- yes: Zipkin:9411
- no: oauth2_proxy:1480

by Corey Donohoe
Giving people access
Sensitive Data
Custom Instrumentation
class ActiveRecord::ConnectionAdapters::AbstractAdapter
  prepend Tracing::SQL
end

module Tracing
  module SQL
    def log(sql, name = "SQL", binds = [], statement_name = nil)
      ZipkinTracer::TraceClient.local_component_span("sql query") do |span|
        span.record_tag("query", sql.to_s)
        super
      end
    end
  end
end

Monkey Patching with Prepend

Mimic log method
Wrap all sql calls and record the sql statement
What happens when data leaks?
Is Everyone On Board?
Get it on the Roadmap
Open PRs
Evaluating Distributed Tracing Solutions:

- Should you buy, build or adopt?
- What are your infrastructure requirements and limitations?
- How is it authenticated?
- Do you have sensitive data? What will you do if it leaks?
- Is everyone on board?
OMG, this is so much information
Try out Docker Zipkin
Come say hi at the

**Heroku Booth**

today, 3:30pm-4:30pm

[@practice_cactus](https://twitter.com/practice_cactus)