From zero to distributed traces
An OpenTracing Tutorial

Bryan Liles (Capital One), Yuri Shkuro (Uber), Won Jun Jang (Uber), Prithvi Raj (Uber)

Velocity NYC, Oct 2 2017
Agenda

- Why care about tracing
- Tracing demo
- Why care about OpenTracing
- CNCF Jaeger
- OpenTracing deep dive
- Showcase & open discussion
Getting the most of this workshop

- Learn the ropes. If you already know them, help teach the ropes :)
- Meet some people

Everyone can walk away with practical tracing experience and a better sense of the space.
Why care about Tracing

Tracing is fun
Today’s applications are complex
BILLIONS times a day!
How do we know what’s going on?
We use MONITORING tools

**Metrics / Stats**
- Counters, timers, gauges, histograms
- Four golden signals
  - utilization
  - saturation
  - throughput
  - errors
- Statsd, Prometheus, Grafana

**Logging**
- Application events
- Errors, stack traces
- ELK, Splunk, Fluentd

Monitoring tools must “tell stories” about your system
Metrics and logs don’t cut it anymore!

Metrics and logs are per-instance

We need to monitor distributed transactions
Systems and Distributed and Concurrent

“The Simple [Inefficient] Thing”

Basic Concurrency

Async Concurrency

Distributed Concurrency
How do we “tell stories” about distributed concurrency?
Enterprise

How cloud computing, big data and new devices are changing work

Google Cloud acquires Bitium
by Darrell Etherington

Microsoft Excel is about to get a lot smarter
by Frederic Lardinois

BlackBerry is making a comeback
by Ron Miller

Microsoft places bets on quantum computing
by Frederic Lardinois

Accela gets acquired by Berkshire Partners as it looks to move govtech services to the cloud
by John Mannes

Boston-based private equity shop Berkshire Partners announced this afternoon that it is acquiring Accela — a nearly 20 year old startup that sells regulatory management solutions to government clients. Accela has gone through a troika of CEOs in the last year. Previously acting CEO Mark Jung replaced Maury Blackman last October who had managed the company for about a decade. We... Read More
Distributed Tracing Systems

- Distributed transaction monitoring
- Root cause analysis
- Distributed context propagation
- Performance and latency optimization
- Service dependency analysis
Understanding Sampling

Tracing data can exceed business traffic.

Most tracing systems sample transactions:

- **Head-based sampling**: the sampling decision is made just before the trace is started, and it is respected by all nodes in the graph
- **Tail-based sampling**: the sampling decision is made after the trace is completed / collected
Let’s look at some traces
demo time: http://bit.do/jaeger-hotrod
Great... Why isn’t everyone tracing?

Tracing instrumentation has been too hard.

- **Lock-in is unacceptable**: instrumentation must be decoupled from vendors
- **Monkey patching doesn’t scale**: instrumentation must be explicit
- **Inconsistent APIs**: tracing semantics must not be language-dependent
- **Handoff woes**: tracing libs in Project X don’t hand-off to tracing libs in Project Y
Enter OpenTracing

http://opentracing.io
OpenTracing in a nutshell

OpenTracing addresses

the instrumentation problem.

Who cares? Developers building:

- Cloud-native / microservice applications
- OSS packages, especially near process edges (web frameworks, managed service clients, etc)
- Tracing and/or monitoring systems
Where does tracing code live?

OSS and commercial / in-house instrumentation
OpenTracing Architecture

- microservice process
- application logic
- µ-service frameworks
- Lambda functions
- RPC & control-flow frameworks
- existing instrumentation

OpenTracing API

CNCF Jaeger
- LIGHTSTEP
- Z I P K I N
- appdash
- INSTANA

tracing infrastructure
A young, growing project

~2 years old

**Tracer implementations**: Jaeger, Zipkin, LightStep, SkyWalking, others

All sorts of companies use OpenTracing:
Rapidly growing OSS and vendor support

- Golang
- Gopher
- Java
- Python
- Objective-C
- Dropwizard
- GRPC
- Spring
- Instana
- Flask
- Java Webservlet
- JDBI
- express
- Django
- Jaxr
- AWS X-Ray
Jaeger

A distributed tracing system
New CNCF Project: Jaeger

https://github.com/uber/jaeger
Jaeger - /ˈyāɡər/, noun: hunter

- Inspired by Google’s Dapper and OpenZipkin
- Started at Uber in August 2015
- Open sourced in April 2017
- Official CNCF project, Sep 2017
- Built-in OpenTracing support
- https://github.com/uber/jaeger
Jaeger: Technology Stack

- Go backend
- Pluggable storage
  - Cassandra, Elasticsearch, memory, ...
- React/Javascript frontend
- OpenTracing Instrumentation libraries
Jaeger: Community

- 10 full time engineers at Uber and Red Hat
- 30+ contributors on GitHub
- Already used by many organizations
  - including Symantec, Red Hat, Base CRM, Massachusetts Open Cloud, Nets, FarmersEdge, GrafanaLabs, Northwestern Mutual, Zenly
OpenTracing deep dive

Doc http://bit.do/velocity17
Materials

- Setup instructions: http://bit.do/velocity17
- Q&A: https://gitter.im/opentracing/workshop
Lesson 1
Hello, World
Lesson 1 Objectives

- Basic concepts
- Instantiate a Tracer
- Create a simple trace
- Annotate the trace
Basic concepts: SPAN

**Span**: a basic unit of work, timing, and causality.

A span contains:

- operation name
- start / finish timestamps
- tags and logs
- references to other spans
Basic concepts: TRACE

**Trace**: a directed acyclic graph (DAG) of spans

- Span A
  - Span B
    - Span D
    - Span E
  - Span C
    - Span F
    - Span G
    - Span H
Trace as a time sequence diagram

A

B

d

C

E

F  G  H

time
Basic concepts: OPERATION NAME

A human-readable string which concisely represents the work of the span.

- E.g. an RPC method name, a function name, or the name of a subtask or stage within a larger computation
- Can be set at span creation or later
- Should be low cardinality, aggregatable, identifying class of spans

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>get</td>
<td>too general</td>
</tr>
<tr>
<td>get_account/12345</td>
<td>too specific</td>
</tr>
<tr>
<td>get_account</td>
<td>good, “12345” could be a tag</td>
</tr>
</tbody>
</table>
Basic concepts: TAG

A key-value pair that describes the span overall.

Examples:

- `http.url` = "http://google.com"
- `http.status_code` = 200
- `peer.service` = "mysql"
- `db.statement` = "select * from users"

https://github.com/opentracing/specification/blob/master/semantic_conventions.md
Basic concepts: LOG

Describes an event at a point in time during the span lifetime.

- OpenTracing supports **structured logging**
- Contains a timestamp and a set of fields

```python
span.log_kv(
    {'event': 'open_conn', 'port': 433}
)
```
A tracer is a **concrete** implementation of the OpenTracing API.

```go
tracer := jaeger.New("hello-world")
span := tracer.StartSpan("say-hello")
// do the work
span.Finish()
```
Understanding Sampling

- Tracing data > than business traffic
- Most tracing systems sample transactions
- **Head-based sampling**: the sampling decision is made just before the trace is started, and it is respected by all nodes in the graph
- **Tail-based sampling**: the sampling decision is made after the trace is completed / collected
How to create Jaeger Tracer

cfg := &config.Configuration{
    Sampler: &config.SamplerConfig{
        Type: "const",
        Param: 1,
    },
    Reporter: &config.ReporterConfig{LogSpans: true},
}

tracer, closer, err := cfg.New(serviceName)
Lesson 2
Context and Tracing Functions
Lesson 2 Objectives

- Trace individual functions
- Combine multiple spans into a single trace
- Propagate the in-process context
How do we build a DAG?

```plaintext
span1 := tracer.StartSpan("say-hello")
// do the work
span1.Finish()

span2 := tracer.StartSpan("format-string")
// do the work
span2.Finish()
```

This just creates two independent traces!
Build a DAG with Span References

```plaintext
span1 := tracer.StartSpan("say-hello")
// do the work
span1.Finish()

span2 := tracer.StartSpan("format-string",
    opentracing.ChildOf(span1.Context()),
)
// do the work
span2.Finish()
```
Basic concepts: SPAN CONTEXT

Serializable format for linking spans across network boundaries.

Carries trace/span identity and baggage.

```go
type SpanContext struct {
    traceID   TraceID
    spanID    SpanID
    parentID  SpanID
    flags     byte
    baggage   map[string]string
}
```
Basic concepts: SPAN REFERENCE

Describes causal relationship to another span.

type Reference struct {
    Type    opentracing.SpanReferenceType
    Context SpanContext
}

47
Types of Span References

**ChildOf**: referenced span is an ancestor that depends on the results of the current span. E.g. RPC call, database call, local function

**FollowsFrom**: referenced span is an ancestor that does not depend on the results of the current span. E.g. async fire-n-forget cache write.
In-process Context Propagation

We don’t want to keep passing Spans around.

Need a more general request context.

- Go: context.Context (from std lib)
- Java, Python: thread-locals (WIP)
- Node.js: TBD (internally: @uber/node-context)
Lesson 3
Tracing RPC Requests
Lesson 3 Objectives

- Trace a transaction across more than one microservice
- Pass the context between processes using Inject and Extract
- Apply OpenTracing-recommended tags
Three Steps for Instrumentation

1. **My Service**
   - **Context Span**
   - **Handler**
     - ***instrumentation***
   - **Headers TraceID**
   - **inbound request**

2. **Client**
   - **Context Span**
   - **Client**
     - ***instrumentation***
   - **Headers TraceID**
   - **outbound request**

3. **Jaeger client library**
   - Send trace data to Jaeger (background thread)
Basic concepts: Inject and Extract

Tracer methods used to serialize Span Context to or from RPC requests (or other network comms)

void Inject(SpanContext, Format, Carrier)

SpanContext Extract(Format, Carrier)
OpenTracing does not define the wire format. It assumes that the frameworks for network comms allow passing the context (request metadata) as one of these (the `Format` enum):

1. **TextMap**: Arbitrary string key/value headers
2. **Binary**: A binary blob
3. **HTTPHeaders**: as a special case of #1
Basic concepts: Carrier

Each Format defines a corresponding Carrier interface that the Tracer uses to read/write the span context.

The instrumentation implements the Carrier interface as an adapter around their custom types.
Inject Example

Set(key, value)

TextMap Carrier → RPC Adapter → AddHeader(key, value) → RPC Request

Tracer

Write(byte[]) → Binary Carrier → Adapter → Write(byte[]) → RPC Request
Lesson 4
Baggage
Lesson 4 Objectives

- Understand distributed context propagation
- Use baggage to pass data through the call graph
Distributed Context Propagation

Problem: how to aggregate disk writes in Cassandra by “button” type (or experiment id, etc, etc)?

See the Pivot Tracing paper [http://pivottracing.io/](http://pivottracing.io/)
Basic concepts: Baggage

Baggage is a general purpose in-band key-value store.

```
span.SetBaggageItem("Bender", "Rodriguez")
```

Transparent to most services.

Powerful but dangerous

- Bloats the request size
Extra Credit
Logging v. Tracing
Monitoring == Observing Events

- **Low volume**
  - **Metrics** - Record events as *aggregates* (e.g. counters)
  - **Tracing** - Record *transaction-scoped* events
  - **Logging** - Record *unique* events

- **High volume**
## Logging v. Tracing

### Logging
- No context
- Low granularity (warn and ↑)
- Per-process sampling (at best)
- High volume, low fidelity

### Tracing
- Contextual
- High granularity (debug and ↓)
- Per-transaction sampling
- Lower volume, higher fidelity

Industry advice: don’t log on success ([https://vimeo.com/221066726](https://vimeo.com/221066726))
Q & A
Open Discussion
Thank You and See You in Austin!

• See you in Austin and Copenhagen!
• KubeCon + CloudNativeCon North America 2017
  - Austin, Texas (December 6 - 8, 2017)
  - Registration & Sponsorships now open: kubecon.io
• KubeCon + CloudNativeCon Europe 2018
  - Copenhagen, Denmark (May 2 - 4, 2018)
  - http://events.linuxfoundation.org/events/kubecon-and-cloudnativecon-europe
Jaeger at Uber

● Root cause and dependency analysis
● Distributed context propagation
  ○ Tenancy
  ○ Security
  ○ Chaos Engineering
● Data mining
  ○ Capacity Planning
  ○ Latency and SLA analysis
Jaeger: Roadmap

- Adaptive sampling
- Data mining pipeline
- Instrumentation in more languages
- Drop-in replacement for Zipkin
- Path-based dependency diagrams
- Latency histograms