Distributed Tracing
Understand how your components work together
About me

José Carlos Chávez

- Software Engineer at Typeform focused on the aggregate of responses services.
- Zipkin core team and open source contributor for Observability projects.

@jcchavezs / #BuildStuffLT
Distributed Systems

@jcchavezs / #BuildStuffLT
Distributed systems

A collection of independent components appears to its users as a single coherent system.

Characteristics:

- Concurrency
- No global clock
- Independent failures
Distributed systems

- Water heater
- Cold water storage tank
- Tank valve
- Shutoff valve
- First floor branch
- Gas supplier

爆$m#$
Distributed systems: Understanding failures

GET /media/e5k2

API Proxy

500 Internal Error

Media API

Images service → DB2

Videos service → DB3

Auth service → DB4

Error 1152
ER_ABORTING_CONNECTION

500 Internal Error
Distributed systems: Understanding failures

I AM HERE!
First floor distributor is clogged!
We do have that, it is called logs!
Logs & Concurrency

GET /media/e5k2

API Proxy

Media API

Images service

DB2

Videos service

DB3

Auth service

DB4

500 Internal Error

Error 1152

ER_ABORTING_CONNECTION
Logs & Concurrency

[24/Oct/2017 13:50:07 +0000] "GET /media HTTP/1.1" 200 ... **0/13548"
[24/Oct/2017 13:50:07 +0000] "GET /media HTTP/1.1" 200 ... **0/23948"
[24/Oct/2017 13:50:08 +0000] "GET /media HTTP/1.1" 200 ... **0/12396"
[24/Oct/2017 13:50:07 +0000] "GET /videos HTTP/1.1" 200 ... **0/23748"
[24/Oct/2017 13:50:07 +0000] "GET /images HTTP/1.1" 200 ... **0/23248"
[24/Oct/2017 13:50:07 +0000] "GET /auth HTTP/1.1" 200 ... **0/26548"
[24/Oct/2017 13:50:07 +0000] "POST /media HTTP/1.1" 200 ... **0/13148"
[24/Oct/2017 13:50:07 +0000] "GET /media HTTP/1.1" 200 ... **0/2588"
[24/Oct/2017 13:50:07 +0000] "GET /auth HTTP/1.1" 500 ... **0/3248"
[24/Oct/2017 13:50:07 +0000] "POST /media HTTP/1.1" 200 ... **0/23548"
[24/Oct/2017 13:50:07 +0000] "GET /images HTTP/1.1" 200 ... **0/22598"
[24/Oct/2017 13:50:07 +0000] "GET /videos HTTP/1.1" 200 ... **0/13948"
...

?
Distributed systems: Understanding failures

I AM HERE!
First floor distributor is clogged!
Distributed Tracing to unclog your pipes
Distributed tracing

API Proxy

Media API

Videos

Images

Auth

TraceID d52d38b69b0fb15efa

I AM HERE!

[1508410442] no cache for resource, retrieving from DBc

Aborted connection

error

Time
Distributed Tracing: What answers I get?

- What services did a request pass through?
- What occurred in each service for a given request?
- Where did the error happen?
- Where are the bottlenecks?
- What is the critical path for a request?
- Who should I page?

@jcchavezs / #BuildStuffLT
Benefits of Distributed Tracing

- (almost) Immediate feedback
- **System insight**, clarifies non trivial interactions
- **Visibility** to **critical paths** and dependencies
- Understand **latencies**
- **Request scoped**, not request’s **lifecycle** scoped.
Trace’s Anatomy

- A trace shows an execution path through a distributed system.
- A span in the trace represents a logical unit of work (with a start and end).
- A context includes information that should be propagated across services.
- Tags and logs (optional) add complementary information to spans.

@jcchavezs / #BuildStuffLT
Elements of distributed tracing

Leg 1: inbound propagation

Leg 2: outbound propagation

Leg 3: in-process propagation

Credits: Nic Munroe
Leg 1: Inbound propagation

When your service process a request or consume a message.

API Proxy → Media API

GET /media

TraceID: fAf3oXL6DS
SpanID: dZ0xHIBa1A

@jcchavezs / #BuildStuffLT
Leg 2: Outbound propagation

When your service makes an outbound call to another service.

```
GET /videos
```

TraceID: fAf3oXL6DS
ParentID: dZ0xHIBa1A
SpanID: y74fr5udj

@jcchavezs / #BuildStuffLT
Leg 3: In process propagation

When performing an operation inside the service

GET /images

Media API

Images service

redis.Get

mysql.Query

Cache service
Any overhead?

For users:
- Observability tools are meant to be **unintrusive**
- **Sampling** reduces overhead
- *(Don’t)* trace every single operation

For developers:
- Not all libraries are ready to plug instruments
- Instrumentation can be delegated to **common frameworks**
Introducing Apache Zipkin

@jcchavezs / #BuildStuffLT
Apache Zipkin

Based on B3 and inspired on Google Dapper (2010). It was open sourced by Twitter (2012) and joined Apache Incubator on September 2018.

- Mature tracing model emerged from users’ needs.
- Used by large companies like Netflix, SoundCloud and Yelp but also small ones.
- Strong community:
  - @zipkinproject
  - gitter.im/openzipkin
Zipkin: architecture

- **Service (instrumented)**
  - Collect spans
  - Transport
    - http/kafka/grpc

- **Collector**
  - Receive spans
  - Deserialize and schedule for storage
  - Store spans
    - DB
      - Cassandra/MySQL/ElasticSearch

- **API**
  - Visualize
  - Retrieve data

- **UI**
Zipkin: traces

Service Name: all
Span Name: all
Lookback: 1 hour
Annotations Query: e.g. "http.path=/foo/bar/ and cluster=foo and cache.miss"
Duration (μs) >=
Limit: 10
Sort: Newest First

Showing: 10 of 10 Services: all

331.522ms 14 spans
- all 0%
  - api-proxy x4 280ms
  - auth-service x2 64ms
  - images-service x2 93ms
  - media-api x3 188ms
  - tracking-service x1 44ms
  - videos-service x2 67ms

276.697ms 12 spans
- all 0%
  - api-proxy x3 278ms
  - auth-service x2 67ms
  - images-service x2 99ms
  - media-api x3 189ms
  - videos-service x2 70ms

2 minutes ago
## Zipkin: traces

<table>
<thead>
<tr>
<th>Service Name</th>
<th>Span Name</th>
<th>Lookback</th>
<th>Duration (μs)</th>
<th>Limit</th>
<th>Sort</th>
</tr>
</thead>
<tbody>
<tr>
<td>api-proxy</td>
<td>all</td>
<td>1 hour</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>auth-service</td>
<td>all</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>images-service</td>
<td>all</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>media-api</td>
<td>all</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tracking-service</td>
<td>all</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>videos-service</td>
<td>all</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 331.522ms 14 spans
- api-proxy x4 290ms
- auth-service x2 64ms
- images-service x2 93ms
- media-api x3 168ms
- tracking-service x1 44ms
- videos-service x2 87ms

(all 0%)

### 276.697ms 12 spans
- api-proxy x3 276ms
- auth-service x2 67ms
- images-service x2 99ms
- media-api x3 189ms
- videos-service x2 70ms

(all 0%)
Zipkin: traces

Service Name: api-proxy

Annotations Query:
- e.g. http.path=/foo/bar and cluster=foo

Span Name:
- all
- get
- get/media/{userid}
- get/tracking/{request_id}
- head
- post
- post/tracking/{request_id}

Lookback: 1 hour

Duration (μs) >=

Limit: 10

Sort: Newest First

331.522ms 14 spans
- all 0%
- api-proxy x4 280ms
- auth-service x2 64ms
- images-service x2 93ms
- media-api x3 168ms
- tracking-service x1 44ms
- videos-service x2 67ms

276.697ms 12 spans
- all 0%
- api-proxy x3 276ms
- auth-service x2 67ms
- images-service x2 99ms
- media-api x3 169ms
- videos-service x2 70ms
Zipkin: trace overview
Zipkin: tags and logs

api-proxy.get /media/{userid}: 262.361ms
AKA: api-proxy

<table>
<thead>
<tr>
<th>Date Time</th>
<th>Relative Time</th>
<th>Annotation</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>20.3.2018, 13:03:01</td>
<td></td>
<td>Server Receive</td>
<td>127.0.0.1:9423 (api-proxy)</td>
</tr>
<tr>
<td>20.3.2018, 13:03:02</td>
<td>262.361ms</td>
<td>Server Send</td>
<td>127.0.0.1:9423 (api-proxy)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>http.method</td>
<td>GET</td>
</tr>
<tr>
<td>http.status_code</td>
<td>200</td>
</tr>
<tr>
<td>http.url</td>
<td>/media/ZeOBC7</td>
</tr>
<tr>
<td>peer.hostname</td>
<td>ip-171-13-08-112.ec2.internal</td>
</tr>
<tr>
<td>request_id</td>
<td>78f13d22c2b5dc9cf46971237ecd23c18</td>
</tr>
</tbody>
</table>
Zipkin: traces with errors
Zipkin: traces for async operations
Zipkin: dependency graph

Start time: 2018-03-19 15:17
End time: 2018-03-20 15:17

Analyze Dependencies

Diagram:
- api-proxy
  - auth-service
  - media-api
    - images-service
    - videos-service
  - tracking-service
Zipkin: dependency graph
Q&As

@jcchavezs


@jcchavezs / #BuildStuffLT