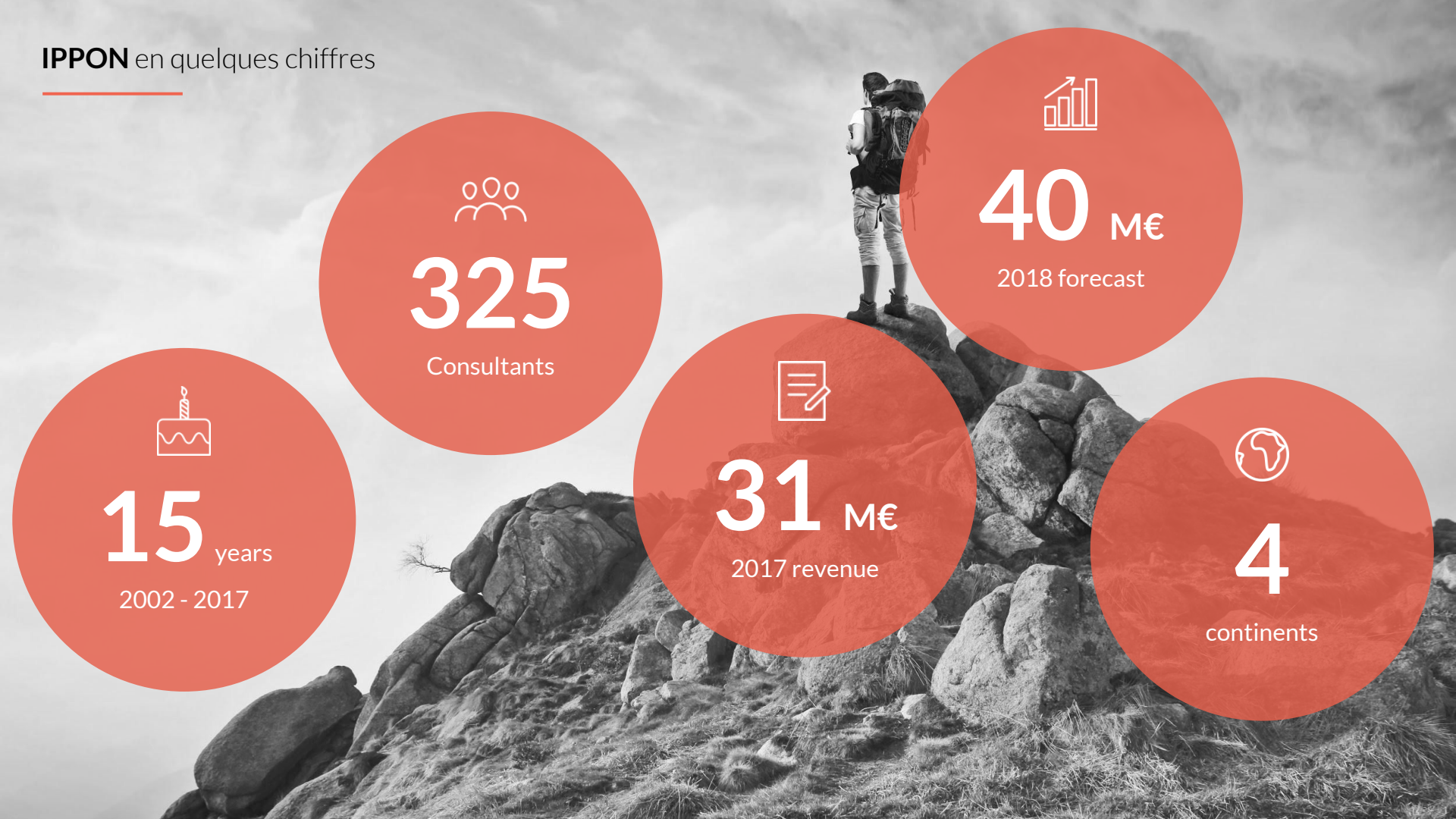
Microservices latency troubleshooting guide

IPPON en quelques chiffres



325
Consultants

40 M€
2018 forecast

15 years
2002 - 2017

31 M€
2017 revenue

4
continents

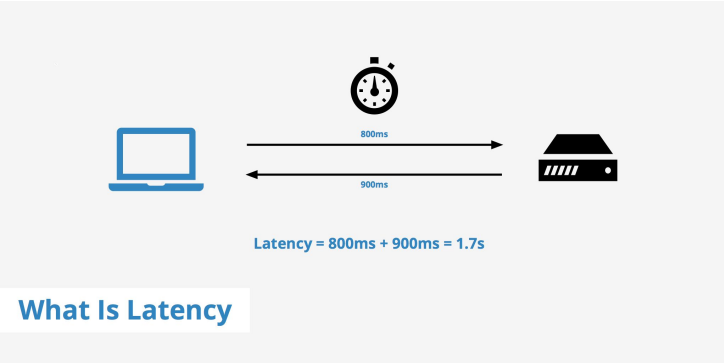
Plan

1. Why should I care ?
2. Do I really need tracing ?
3. Tracing basics
4. What should I use ?
5. Demo time

Why should I care

The latency

- *“Latency is a time interval between the stimulation and response, or, from a more general point of view, a time delay between the cause and the effect of some physical change in the system being observed.”*



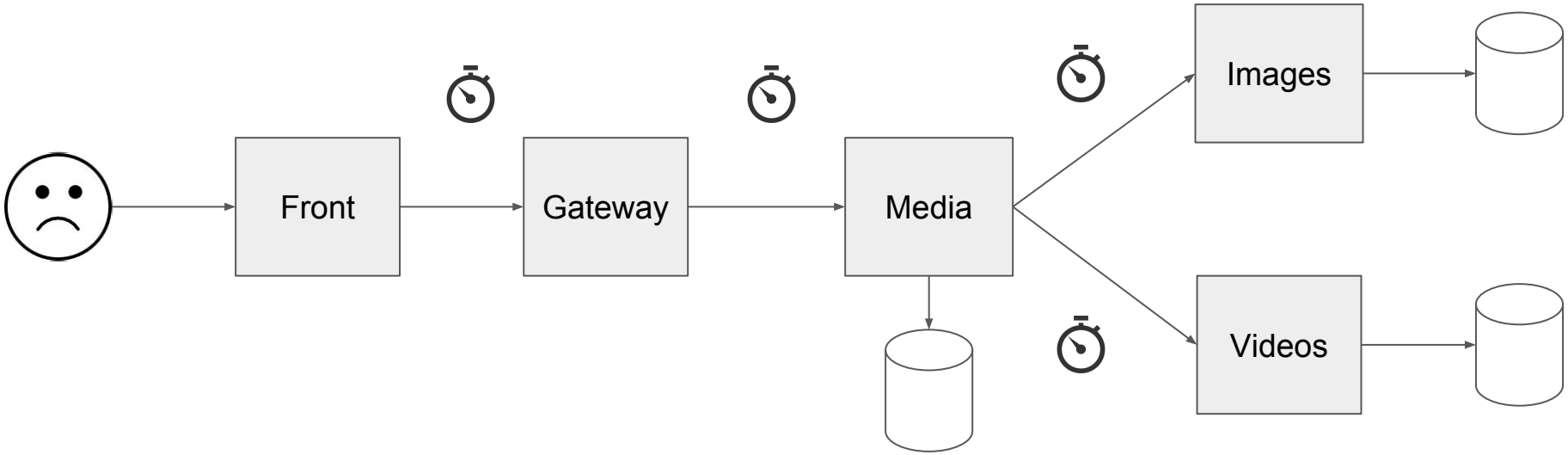
keycdn.com

- It has impact
  - Amazon found every 100ms of latency cost them 1% in sales.
  - Google found an extra 0.5 seconds in search page generation time dropped traffic by 20%.
  - A broker could lose $4 million in revenues per millisecond if their electronic trading platform is 5 milliseconds behind the competition

Enters the microservices

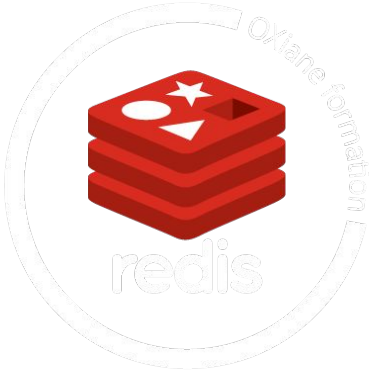
- Has some famous examples
  - Netflix
  - Google
  - AirBnB
- Fits well with the way we manage infrastructure
  - (multi) Cloud
  - Containers
  - FAAS
- Brings a lot of new challenges
  - Deployment
  - Interservice communication
  - **Monitoring**

Latency and network

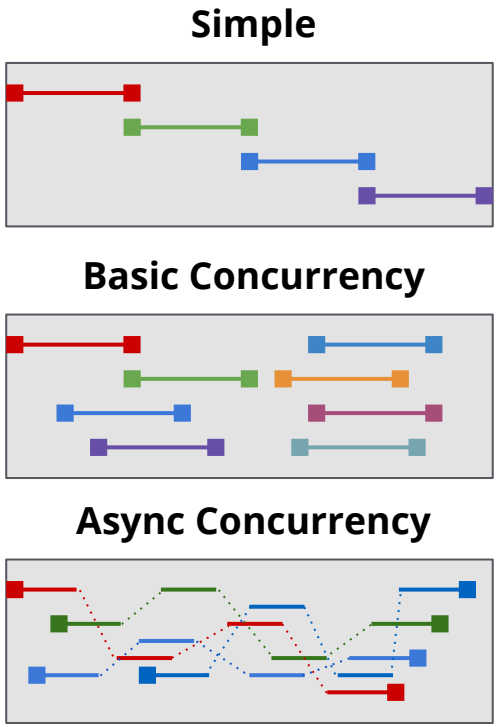


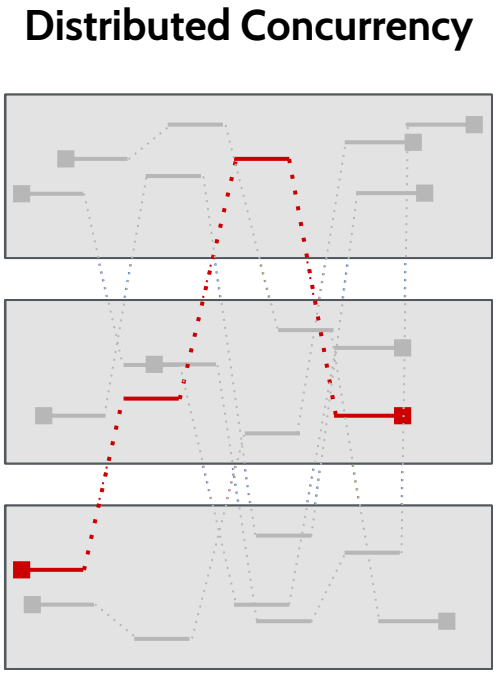
What if I use a queue ?

- Changes nothing in term of latency
- Latency is still a key indicator
- You still have to use network to send messages
- This is just another transport mechanism !



Latency and concurrency





Source: [B.Sigelman]

And there is more

- You need to align clock between differents instances
  - Use timestamps ?
  - Or to treat time at collect time ?
- How to gather data without impacting performances ?
- Does It work with thousands of services ?
- This is not trivial


Do I really need tracing ?

Can’t I just use logs ?

- *“In computing, a log file is a file that **records** [...] **events** that occur in an operating system or other software runs [...] Logging is the act of keeping a log.”*

- We are at the **event scale**

- The good part
  - Simple timestamped and structured messages
  - Tools are available to do it, easily
  - Easy to grep, and read manually
  - Easy to aggregate and parse

- Overhead ?
  - Grows with traffic and verbosity
  - Storage impact limited by pipeline filtering

- Perfect for monoliths, black boxes, exceptional cases

What about metrics ?

- *“A software metric is a standard of measure of a degree to which a software system or process possesses some property.”*
- ≃ Data combined from measuring events

- We are at the **software scale**

- The good part
  - Time and store values of anything
  - Tools are available to do it
  - Report duration buckets near-real time
  - Let you identify trends

- Overhead
  - Growing size is fixed
  - Storage impact limited by writing only what you need

- Perfect for identifying patterns and for alerting

What is tracing ?

- *“In software engineering, tracing involves a specialized use of logging to record information about a program's execution [...]. **Tracing is a cross-cutting concern**.”*
- ≃ Recording events with causal ordering

- We are at the **system scale**

- The good part
  - Common terminology
  - Tools are available to do it
  - Report data near-real time
  - Identify issues across services

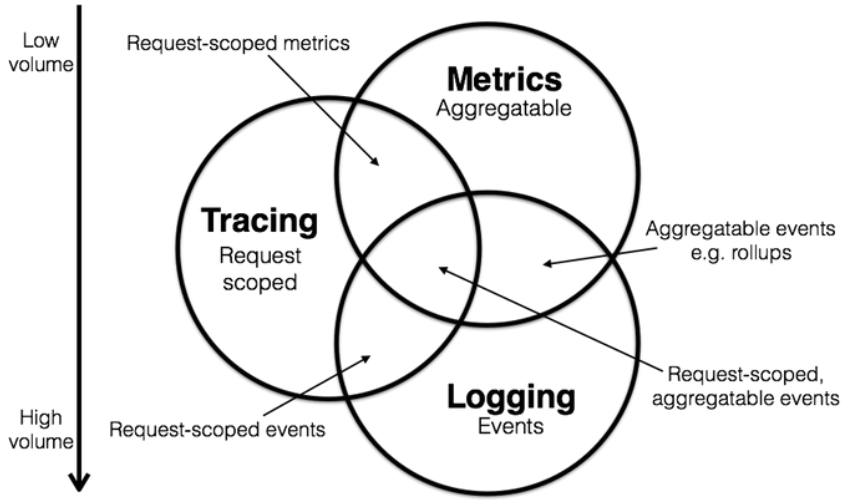
- Overhead
  - Grows with traffic
  - Storage impact limited by various techniques

- Perfect for distributed services

What you want ?

- Use everything

- They work together
  - Log correlation with traces
  - Correlate traces with metrics
  - Fetch metrics on collected traces



Source: Peter Bourgon

- Production issue debugging
  - Something is wrong
  - Lookup metrics for abnormal load, broken instances
  - Check for traces with abnormal latency, errors
  - Find the impacted service
  - Lookup for the logs in a particular service for a particular trace ID
  - Fix the issue
  - Enjoy your night

Tracing basics

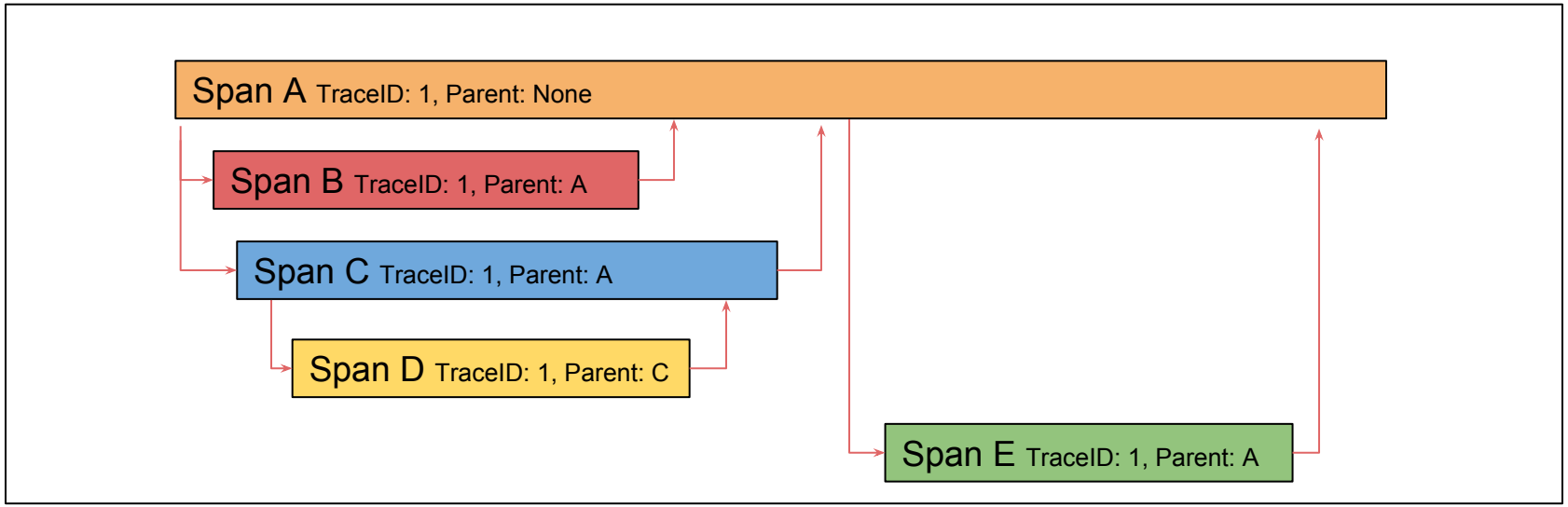
A bit of history

- Google used the dapper paper for more than ten years
- Been published online in 2010 by a Google team
- Twitter open sources Zipkin in 2012
- OpenTracing created in 2014
- OpenTracing joins the CNCF in 2016
- First Draft of trace context published by a W3C working group in 2018

Some vocabulary

A **Trace** represent your whole request across all services involved.

A **Span** is the basic unit of work in a given Trace. You can see it as the duration of an operation.



Spans in detail

service2.http:/readtimeout: 3.557s

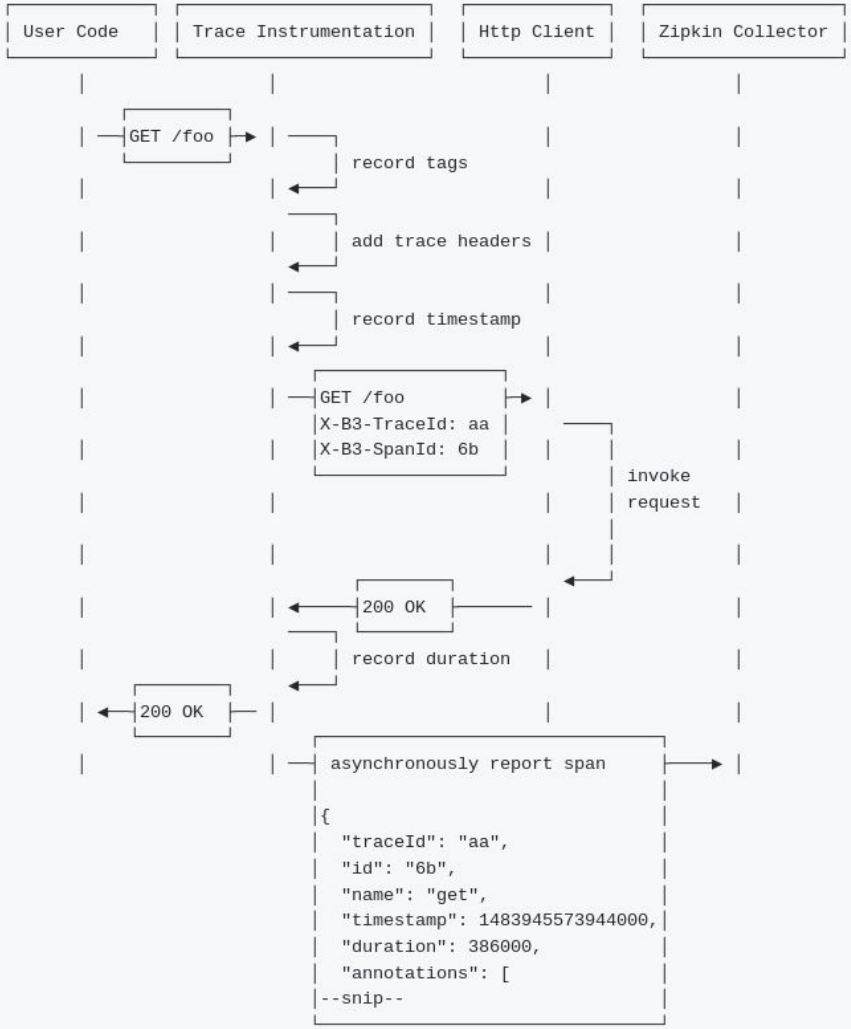
AKA: service1,service2

Date Time	Relative Time	Annotation	Address
19/12/2016, 14:19:23	307.000ms	Client Send	127.0.0.1:8081 (service1)
19/12/2016, 14:19:23	310.000ms	Server Receive	127.0.0.1:8082 (service2)
19/12/2016, 14:19:26	3.836s	Server Send	127.0.0.1:8082 (service2)
19/12/2016, 14:19:27	3.864s	Client Receive	127.0.0.1:8081 (service1)

Key	Value
error	Request processing failed; nested exception is org.springframework.web.client.ResourceAccessException: I/O error on GET request for "http://localhost:8082/blowup": Read timed out; nested exception is java.net.SocketTimeoutException: Read timed out
http.host	localhost
http.method	GET
http.path	/readtimeout
http.status_code	500
http.url	http://localhost:8082/readtimeout
mvc.controller.class	BasicErrorController
mvc.controller.method	error

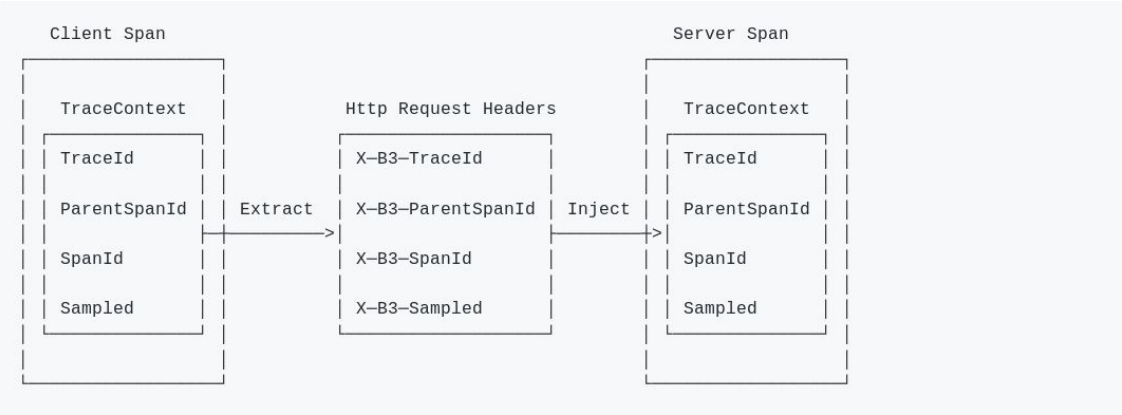
Tags and baggage

- Tags
  - Key value pairs
  - Additional data on spans
  - Not shared with other spans
  - Ex : http.host, http.method ...
- Baggages
  - Key value pairs
  - Part of the propagated context
  - Be careful about latency impact if you add too many


User Code
Trace Instrumentation
Http Client
Zipkin Collector
GET /foo
record tags
add trace headers
record timestamp
GET /foo
X-B3-TraceId: aa
X-B3-SpanId: 6b
invoke
request
200 OK
record duration
200 OK
asynchronously report span
{
"traceId": "aa",
"id": "6b",
"name": "get",
"timestamp": 1483945573944000,
"duration": 386000,
"annotations": [
--snip--

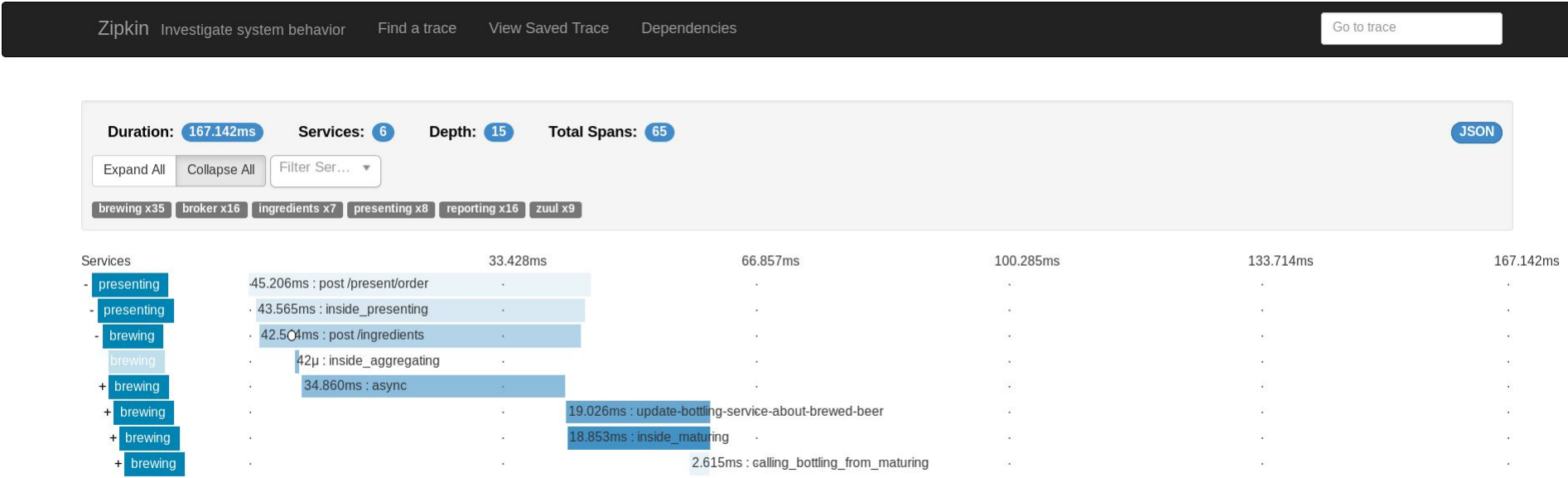
How to keep context

- Inbound / Outbound
  - Read/Write request headers
  - Read/Write queue headers
  - Read/Write request body



- We need to keep a context per request
  - ThreadLocal
  - Pass context between methods

Visualisation



Overhead

- On the wire the cost is low
  - Few headers
  - Baggage
- Span reporting
  - batch
  - asynchronous
- Sampling
  - Percentage
  - User defined

- From a developer point of view
  - Transparent most of usual cases
  - Need awareness for advanced cases

What should I use ?

Standardization ?

- CNCF and Opentracing
  - More or less an interface to implement to be compliant
  - Is a good idea
  - Lot of implementations
  - But
    - wire format not fixed
    - Will create a lot of tracing library, maybe incompatible
    - Maybe too much responsibility

- W3C trace context headers working group

Tracing libraries

- Zipkin
  - One of the historic tracing system
  - Nearly all languages supported
  - Initiated by Twitter, now community driven
- Jaeger
  - One of Opentracing first implementation
  - Has a lot of traction
  - Backed by Uber, now in the CNCF
- Opencensus
  - The newcomer
  - Alpha stage
  - Backed by Google & Microsoft

In the cloud

TRACE REQUESTS

AWS X-Ray traces requests made to your application.

X-Ray collects data about the request from each of the underlying application services it passes through.

RECORD TRACES

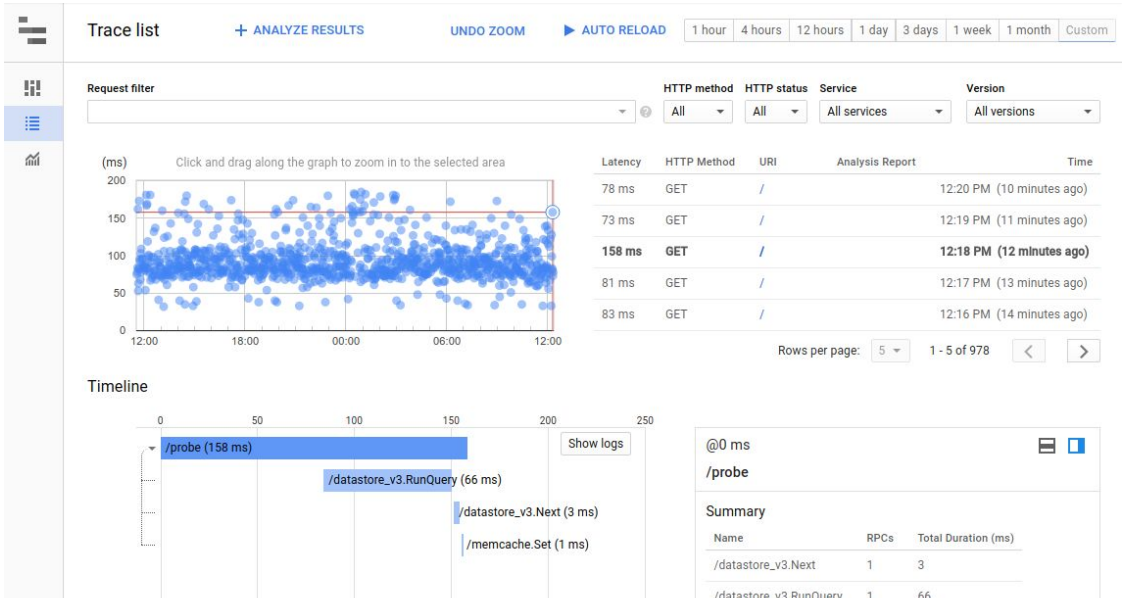
X-Ray combines the data gathered from each service into singular units called traces.

VIEW SERVICE MAP

View the service map to see trace data such as latencies, HTTP statuses, and metadata for each service.

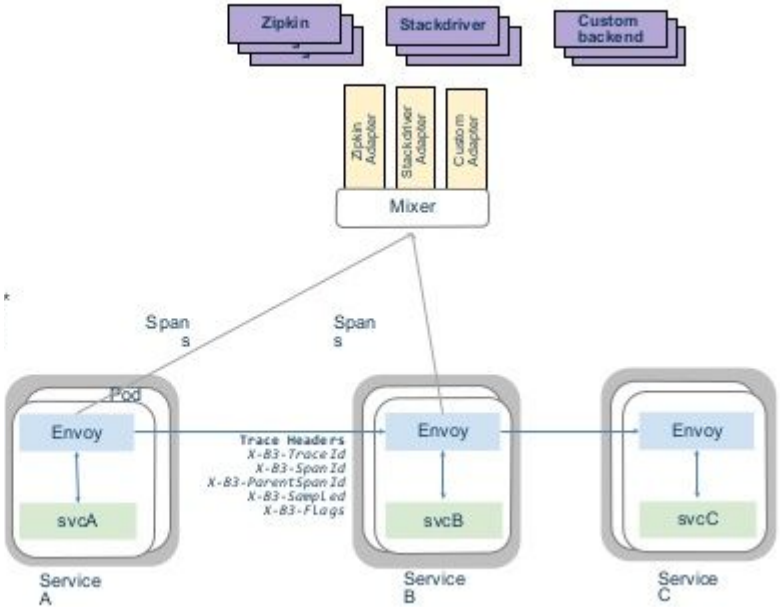
ANALYZE ISSUES

Drill into the service showing unusual behavior to identify the root issue.

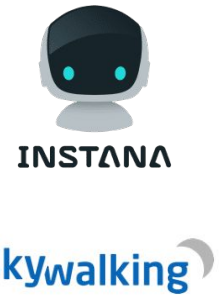


Without code

- Services mesh add new monitoring capabilities
  - Automatically collect requests
  - The apps needs to propagate headers



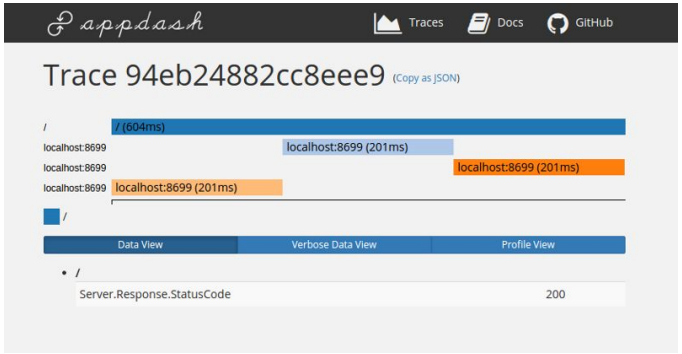
- APM
  - Instana
  - Skywalking
  - …

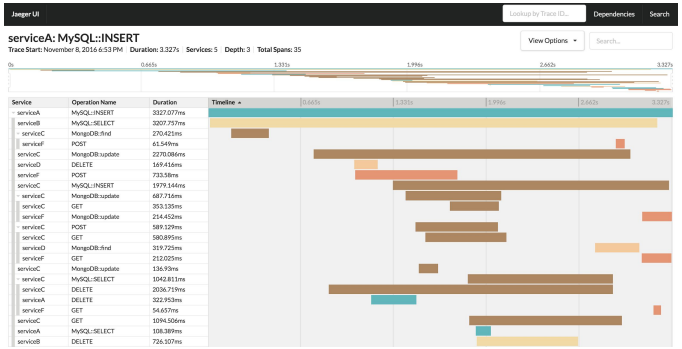


Visualisation tools

- Zipkin
- Lightstep
- Jaeger
- appdash
- Tracer
- Stackdriver
- X-ray
- …

- Backed by a storage infrastructure
  - Elasticsearch
  - Cassandra
  - MySQL
  - Cloud magic





Demo time

The example

2

1

Java integration

```
private Tracer tracer;

private void getGlass() throws InterruptedException {
    Span s = tracer.nextSpan().start();
    s.name("glass");
    logger.info("Fetching a glass");
    s.tag("type", "old fashioned");
    Thread.sleep(50);
    s.annotate("fetched");
    Thread.sleep(100);
    s.annotate("cleaned");
    s.finish();
}
```

NodeJS integration

```
// Continuation local storage
var ctxImpl = new CLSContext('zipkin');

// Add a http transport to local zipkin instance
var recorder = new BatchRecorder({
    logger: new HttpLogger({
    endpoint: 'http://localhost:9411/api/v1/spans'
  })
});

// Create a tracer
var tracer = new Tracer({ctxImpl, recorder});

// Add the Zipkin middleware to express js
app.use(zipkinMiddleware({
  tracer,
  serviceName: 'dude', // name of this application
  sampler: new zipkin.sampler.CountingSampler(1)
}));

// instrument the client
var zipkinRest = rest.wrap(restInterceptor, {tracer, serviceName: 'dude'});

// Configure REST endpoints
app.get('/cocktail', (req,res) => {
    zipkinRest('http://localhost:8081/make')
        .then(
            (response) => res.send(response.entity),
            (response) => console.error("Error", response.status)
        )
});
```

Go integration

```
// Initialize zipkin http collector
collector, err := zipkintracer.NewHTTPCollector("http://localhost:9411/api/v1/spans")
// Create the tracer
tracer, err := zipkintracer.NewTracer(
    zipkintracer.NewRecorder(collector, false, ":8082", "clerk"),
)
opentracing.InitGlobalTracer(tracer)
// Define handlers
http.HandleFunc("/fetchIngredients", func(w http.ResponseWriter, r *http.Request) {
    // Fetch the request context
    wireContext, err := opentracing.GlobalTracer().Extract(
        opentracing.HTTPHeaders,
        opentracing.HTTPHeadersCarrier(r.Header))
    if err != nil {
        log.Print(err)
    }
    // Create a new span from the context
    serverSpan := opentracing.StartSpan("clerk", ext.RPCServerOption(wireContext))
    defer serverSpan.Finish()

    // We need to explicitly pass context to other methods
    getMilk(serverSpan.Context())
    fmt.Fprint(w, "Good milk and ice for you !")
})
```

Wrapping up

What we saw

- You need distributed tracing to
  - Do root cause analysis
  - Understand your system
  - Optimize performances
  - Increase efficiency
  - Improve reliability

- Various tools allow you to do this easily
  - With tracers
  - With framework instrumentation
  - With cloud first tools
  - With your favourite APM

- Collaborate wells with other monitoring tools
  - Log
  - Metrics
  - Traces

- This is a moving field
  - Standards are emerging
  - New tracers & instrumentations
  - A must have for distributed systems !

Useful resources

- Code
  - My demo
  - Brewing demo
  - OpenZipkin project
  - Jaeger tracing project

- Slides & conferences
  - FullstackFest conference by @adrianfcole
  - MicroXchg conference by @jcchavezs
  - Kubecon conference by @YuriShkuro
  - Slides from @el_bhs

- Blogs
  - A blog on observability
  - Metrics Tracing & Logging by @peterbourgon
  - Distributed Tracing core concepts by @MunroeNic

PARIS
BORDEAUX
NANTES
LYON
MARRAKECH
WASHINGTON DC
NEW-YORK
RICHMOND
MELBOURNE

contact@ippon.fr
www.ippon.fr - www.ippon-hosting.com - www.ippon-digital.fr
@ippontech
-
01 46 12 48 48