An Introduction to Distributed Tracing and Zipkin

Bohdan Levchenko

23 March, 2017

Agenda

- What is distributing tracing?

Agenda

- What is distributing tracing?
- Homemade solutions

Agenda

- What is distributing tracing?
- Homemade solutions
- Zipkin terminology & architecture

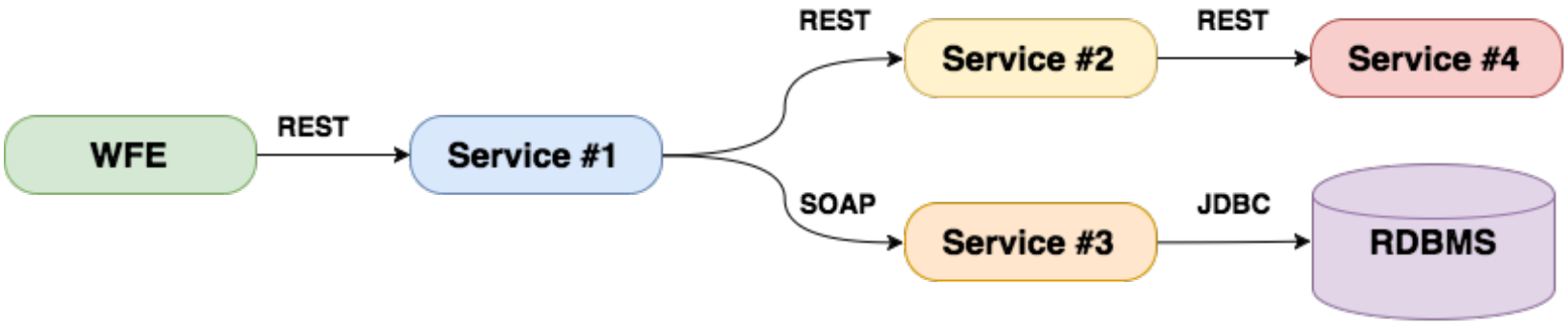
Agenda

- What is distributing tracing?
- Homemade solutions
- Zipkin terminology & architecture
- Performace

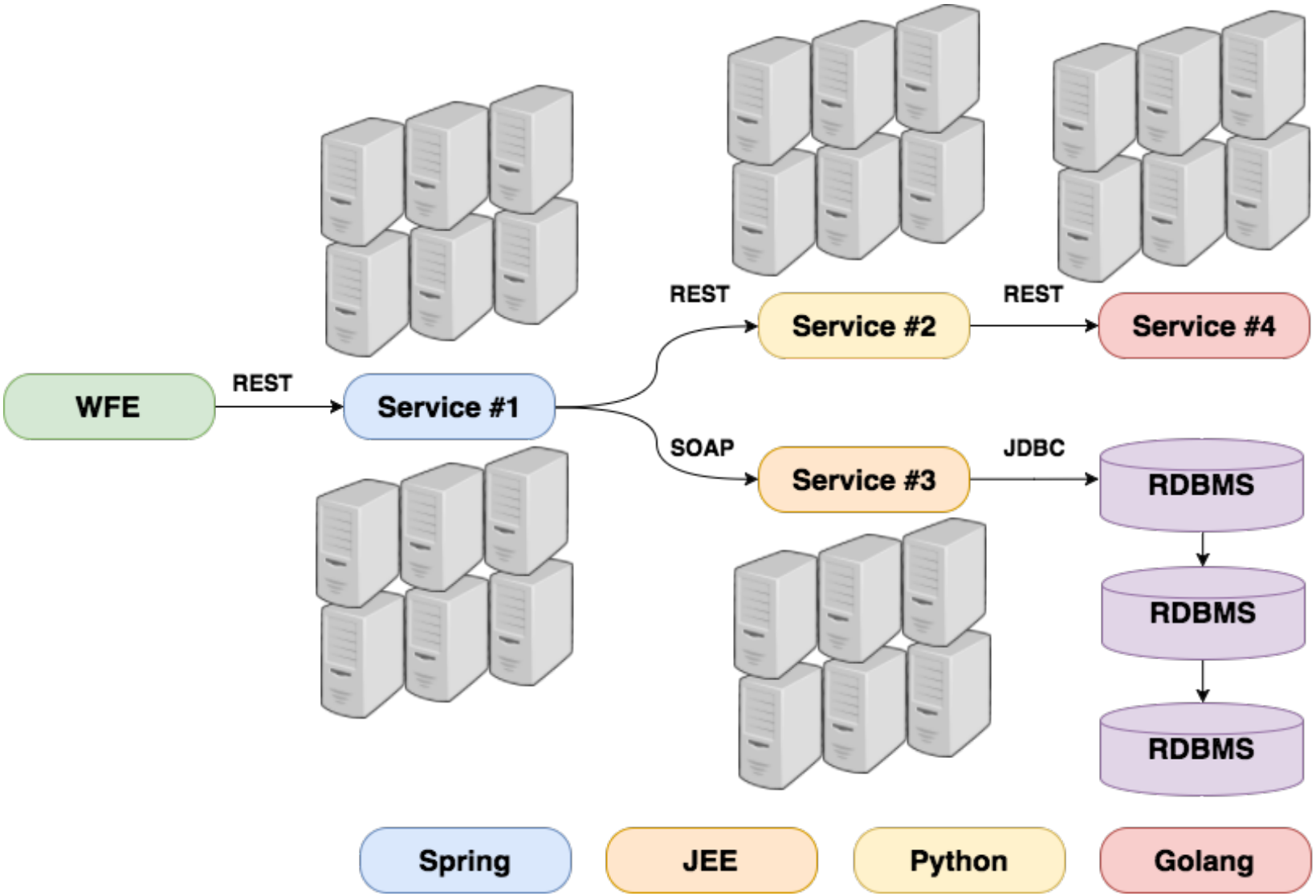
The beginning of everything

> In short, the microservice architectural style is an approach to developing a single application as a suite of small services, each running in its own **process** and communicating with lightweight mechanisms, often an **HTTP** resource API. These services are built around business capabilities and independently deployable by fully automated deployment machinery.
>
> — Martin Fowler

Microservices (expectations)



Microservices (reality)



500: Internal server error

On which server/instance/container was that NPE thrown?

500: Internal server error

On which server/instance/container was that NPE thrown?

> Mmm... We need a tracer or something?
>
> — Anonymous developer

Demo time...

Challenge accepted! (Homemade tracer)

```
$ curl -XPOST 'http://skynet.com:9001/annihilate?id=4321'
```

```
@Bean
Filter tracingFilter(final TraceIdHolder traceIdHolder) {
    return new Filter() {
        @Override public void init(FilterConfig filterConfig) throws ServletException {}

        @Override
        public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
                throws IOException, ServletException {
            final HttpServletRequest httpRequest = ((HttpServletRequest) request);
            traceIdHolder.setTraceId(httpRequest.getHeader(TraceIdHolder.HEADER_NAME));
            chain.doFilter(request, response);
        }

        @Override public void destroy() {}
    };
}
```

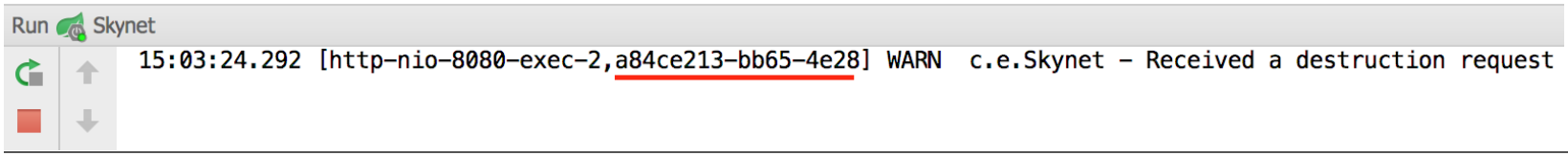
```
@Bean
ClientHttpRequestInterceptor traceInterceptor() {
    return (request, body, execution) -> {
        request.getHeaders().add(TraceIdHolder.HEADER_NAME, TraceIdHolder.getTraceId());
        return execution.execute(request, body);
    };
}
```

Challenge accepted! (Homemade tracer v2 with logging support)

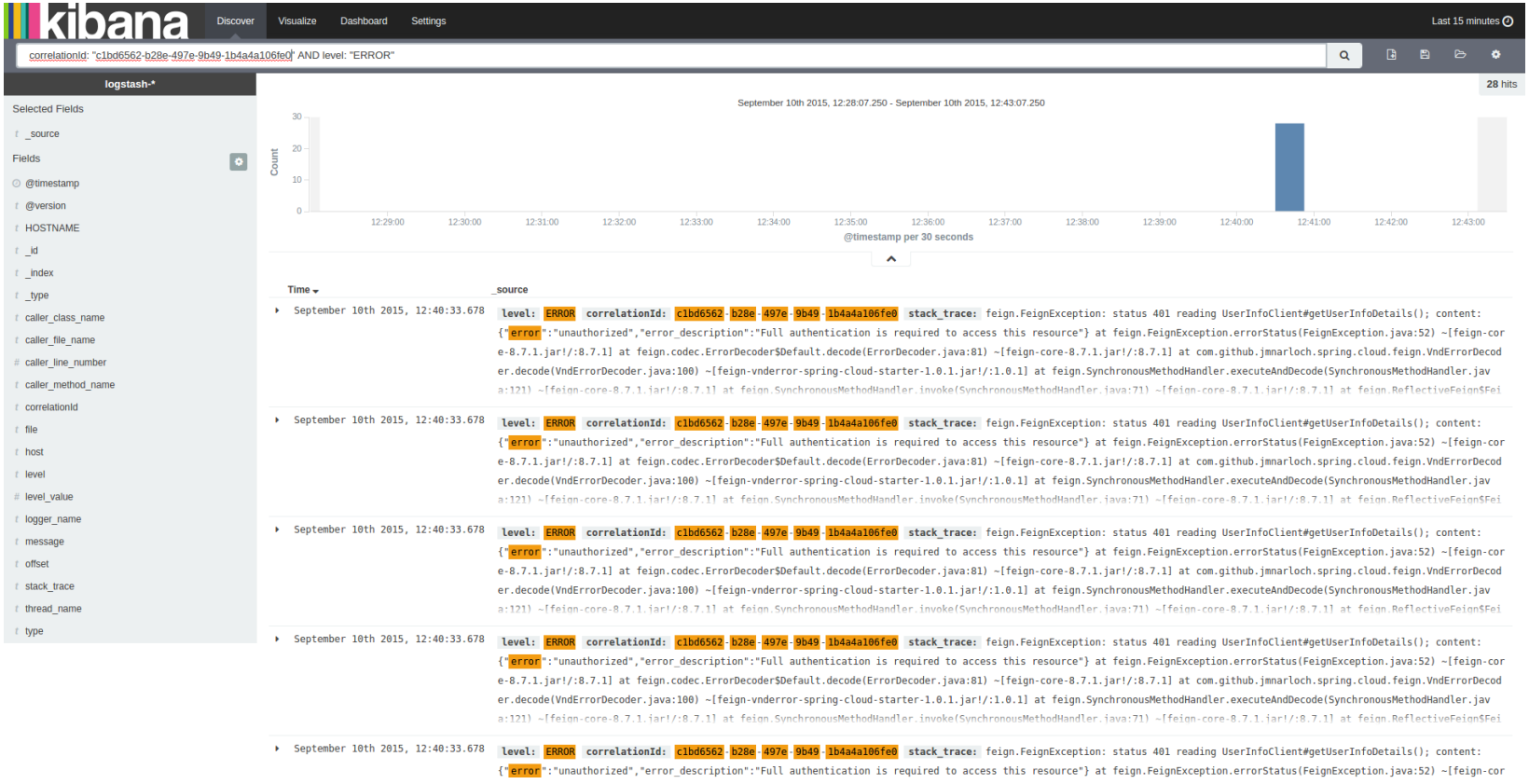
```
log.warn("Received a destruction request.");
```

```
public class TraceIdPatternLayoutEncoder extends PatternLayoutEncoder {

    public static class TraceIdConverter extends ClassicConverter {
        @Override
        public String convert(ILoggingEvent event) {
            return TraceIdHolder.getTraceId();
        }
    }

    @Override
    public void start() {
        final PatternLayout patternLayout = new PatternLayout();
        patternLayout.getDefaultConverterMap().put("traceId", TraceIdConverter.class.getName());
        patternLayout.setContext(context);
        patternLayout.setPattern(getPattern());
        patternLayout.setOutputPatternAsHeader(outputPatternAsHeader);
        patternLayout.start();
        this.layout = patternLayout;
        this.started = true;
    }
}
```

```
<appender name="CONSOLE" class="ch.qos.logback.core.ConsoleAppender">
    <encoder class="com.example.CorrelationLayoutEncoder">
        <pattern>%d{HH:mm:ss.SSS} [%thread,%id] %-5level %logger{5} - %msg%n</pattern>
    </encoder>
</appender>
```



It works!


kibana
Discover
Visualize
Dashboard
Settings
Last 15 minutes
correlationId: "c1bd6562-b28e-497e-9b49-1b4a4a106fe0" AND level: "ERROR"
logstash-*
Selected Fields
_source
Fields
@timestamp
@version
HOSTNAME
_id
_index
_type
caller_class_name
caller_file_name
caller_line_number
caller_method_name
correlationId
file
host
level
level_value
logger_name
message
offset
stack_trace
thread_name
type
28 hits
September 10th 2015, 12:28:07.250 - September 10th 2015, 12:43:07.250
Count
@timestamp per 30 seconds
Time
_source
September 10th 2015, 12:40:33.678
level: ERROR correlationId: c1bd6562-b28e-497e-9b49-1b4a4a106fe0 stack_trace: feign.FeignException: status 401 reading UserInfoClient#getUserInfoDetails(); content:
{"error":"unauthorized","error_description":"Full authentication is required to access this resource"} at feign.FeignException.errorStatus(FeignException.java:52) ~[feign-core-8.7.1.jar!/:8.7.1] at feign.codec.ErrorDecoder$Default.decode(ErrorDecoder.java:81) ~[feign-core-8.7.1.jar!/:8.7.1] at com.github.jmnarloch.spring.cloud.feign.VndErrorDecoder.decode(VndErrorDecoder.java:100) ~[feign-vnderror-spring-cloud-starter-1.0.1.jar!/:1.0.1] at feign.SynchronousMethodHandler.executeAndDecode(SynchronousMethodHandler.jav

It works, but...!

But:

- You *need* to aggregate logs to one place

It works, but...!

But:

- You *need* to aggregate logs to one place
- Tons of boilerplate code for different libraries

It works, but...!

But:

- You *need* to aggregate logs to one place
- Tons of boilerplate code for different libraries
- Async calls?

It works, but...!

But:

- You *need* to aggregate logs to one place
- Tons of boilerplate code for different libraries
- Async calls?
- Doesn’t help much with **"The system is slow!"** problem

It works, but...!

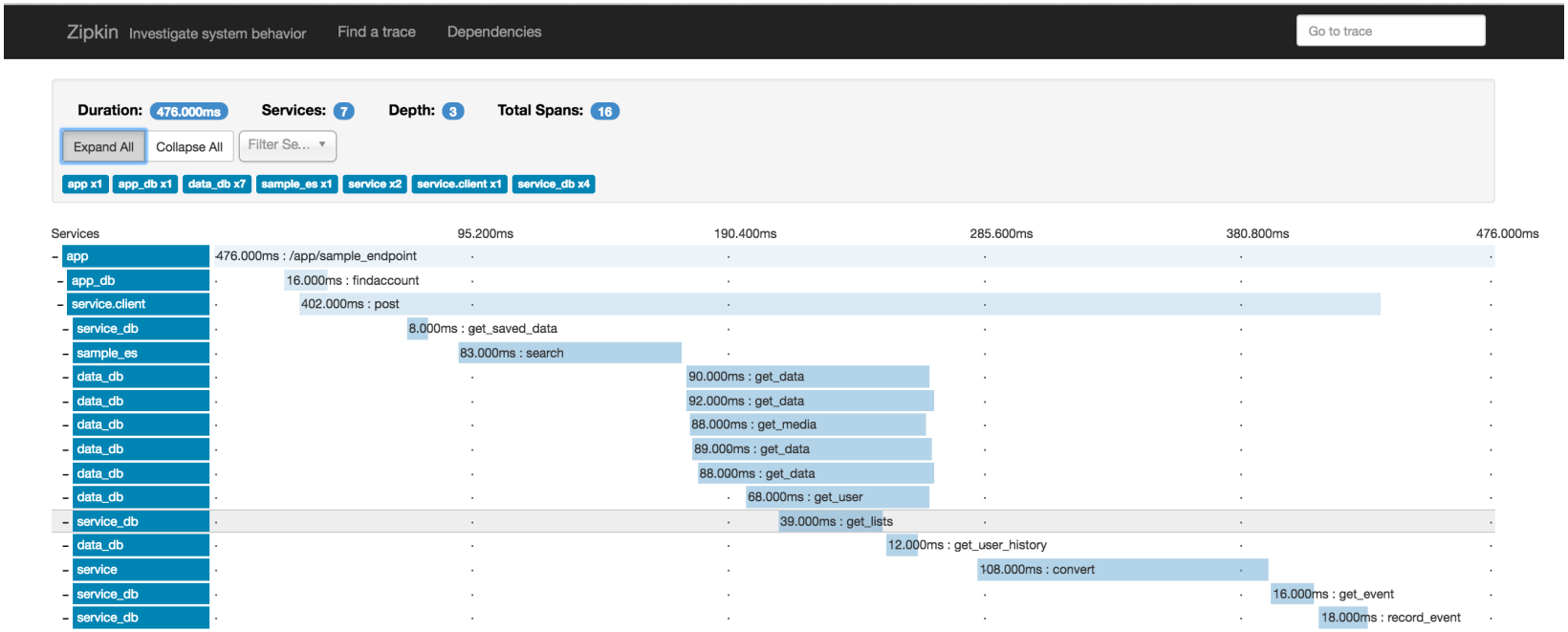
But:

- You *need* to aggregate logs to one place
- Tons of boilerplate code for different libraries
- Async calls?
- Doesn’t help much with **"The system is slow!"** problem

> Mmm... So how to tackle all of these?
>
> — Anonymous developer

The answer is Zipkin!

Zipkin!

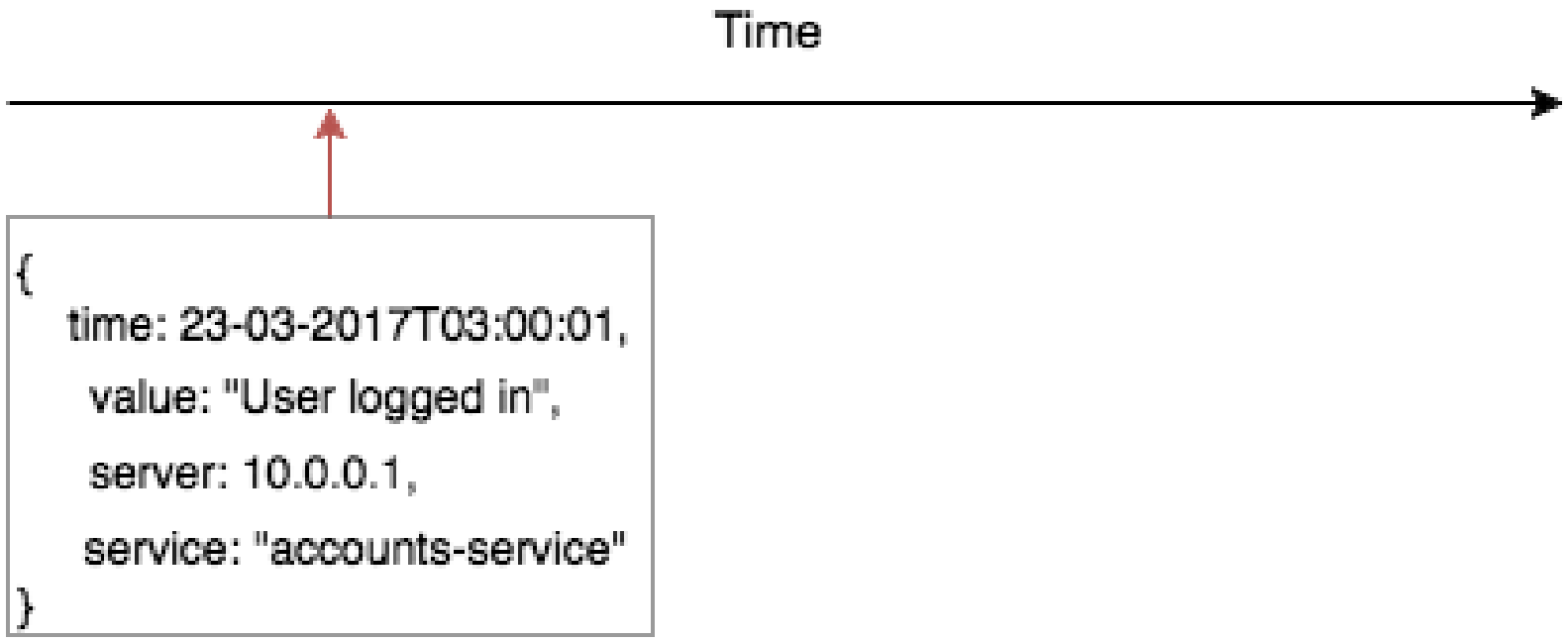


Zipkin is a distributed tracing system. It helps gather timing data needed to troubleshoot latency problems in microservice architectures. It manages both the collection and lookup of this data.

Based on the Google Dapper 2010 paper. Implemented and open sourced by Twitter.

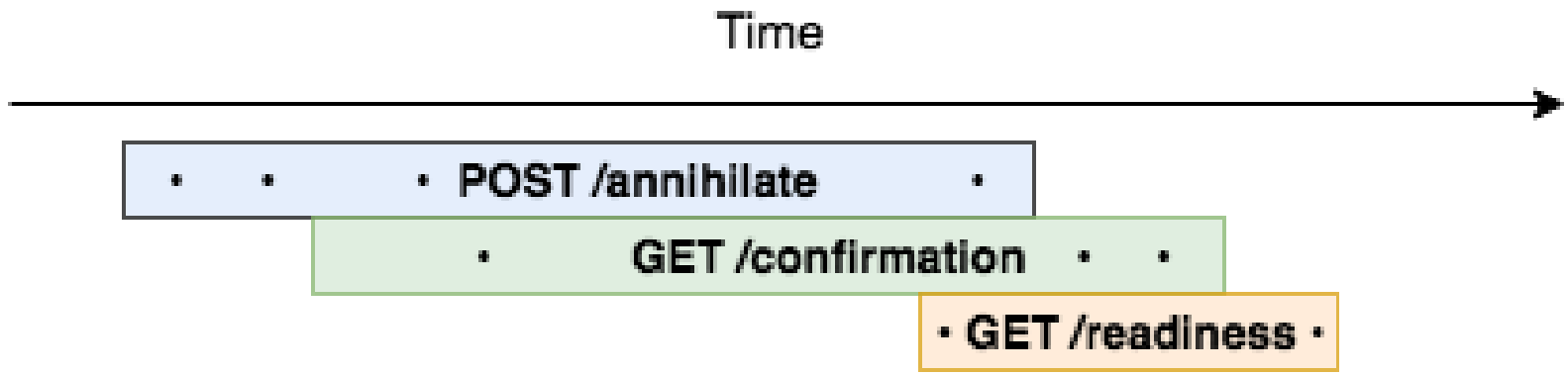
Zipkin terminology

Annotation - string data bounded to specific point in time.



Zipkin terminology

Span - represents a basic unit of work; wraps a set of annotations.
Trace - respresents a tree-like collection of spans.

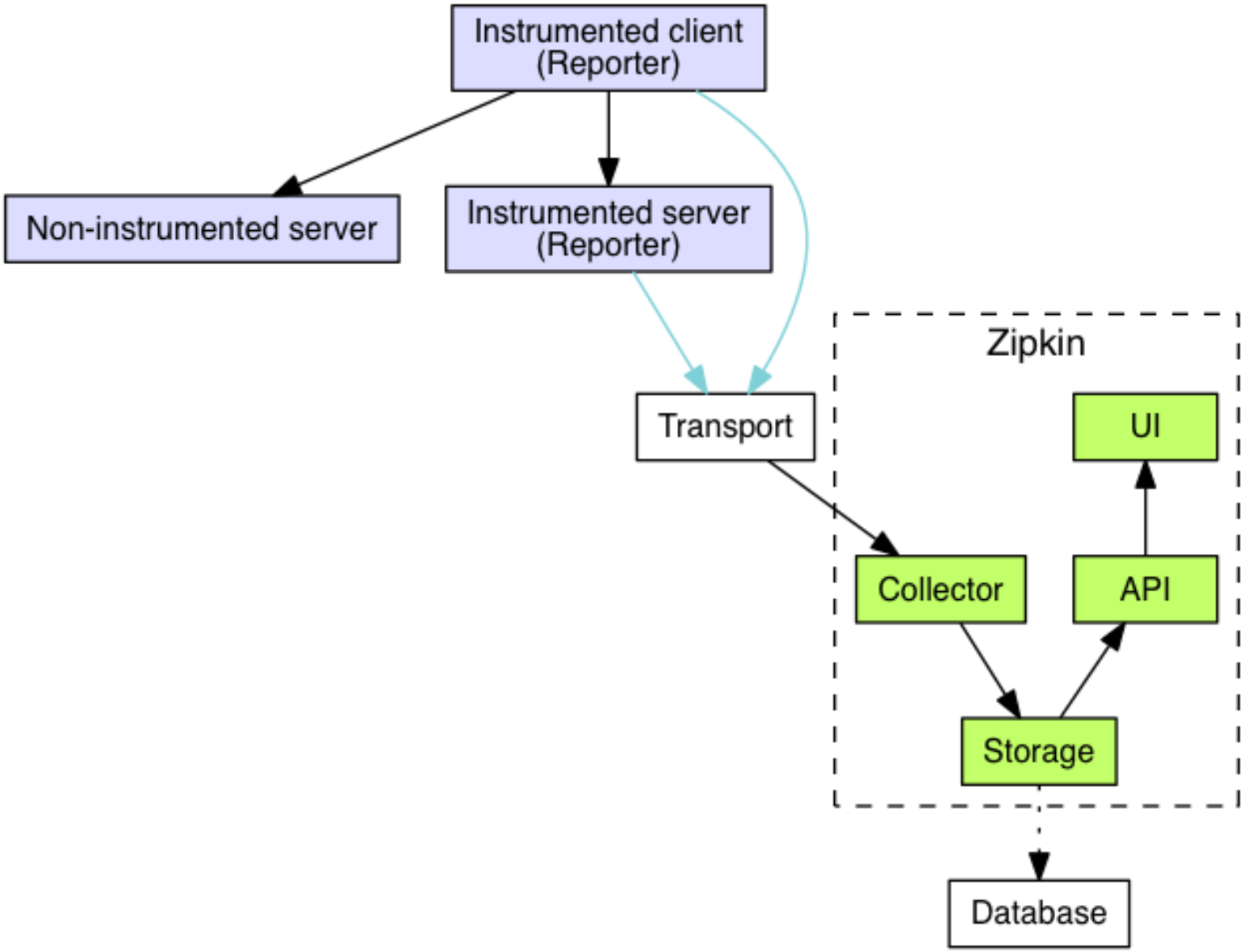


Zipkin components

There are 4 components that make up Zipkin:

- collector
- storage
- search
- web UI

Zipkin architecture



Zipkin architecture

OpenZipkin libraries

Language	Library	Framework	Propagation Supported	Transports Supported	Sampling Supported?
Go	zipkin-go-opentracing	Go kit, or roll your own with OpenTracing	Http (B3), gRPC (B3)	Http, Kafka, Scribe	Yes
Java	brave	Jersey, RestEASY, JAXRS2, Apache HttpClient, Mysql	Http (B3), gRPC (B3)	Http, Kafka, Scribe	Yes
JavaScript	zipkin-js	cujoJS, express, restify	Http (B3)	Http, Kafka, Scribe	Yes
Ruby	zipkin-ruby	Rack	Http (B3)	Http, Kafka, Scribe	Yes
Scala	zipkin-finagle	Finagle	Http (B3), Thrift	Http, Kafka, Scribe	Yes

Zipkin architecture

Community libraries

Language	Library	Framework	Propagation Supported	Transports Supported
C#	ZipkinTracerModule	OWIN, HttpHandler	Http (B3)	Http
C#	Zipkin4net	Any	Http (B3)	Any
Go	go-zipkin	x/net Context		Kafka
Go	monkit-zipkin	Monkit	Http (B3), easy to add others	Scribe, UDP, easy to add others
Java	cassandra-zipkin-tracing	Apache Cassandra	CQL (B3)	Http, Kafka, Scribe
Java	Dropwizard Zipkin	Dropwizard	Http (B3), Thrift	Http, Scribe
Java	htrace	HDFS, HBase		Http, Scribe
Java	Spring Cloud Sleuth	Spring, Spring Cloud (e.g. Stream, Netflix)	Http (B3), Messaging (B3)	Http, Spring Cloud Stream Compatible
Java	Wingtips	Any Servlet API framework, roll-your-own, async framework support	Http (B3)	Http
Python	py_zipkin	Any	Http (B3)	Pluggable
Python	pyramid_zipkin	Pyramid	Http (B3)	Kafka | Scribe
Python	swagger_zipkin	Swagger (Bravado), to be used with py_zipkin	Http (B3)	Kafka | Scribe
Python	flask_zipkin	Flask	Http (B3)	Pluggable
Scala	akka-tracing	Akka, Spray, Play	Http (B3), Thrift	Scribe
Scala	play-zipkin-tracing	Play	Http (B3)	Http

Demo time...

Performance?

> Mmm... That’s impressive, but wouldn’t it affect the preformace of my system?
>
> — Anonymous developer

Performance?

> Sampling!
>
> — Anonymous developer

What to see next?

- Dapper, a Large-Scale Distributed Systems Tracing Infrastructure
- Zipkin at Twitter
- Introducing CallTracing(tm), based on RabbitMQ, Spring and Zipkin
- Implementing Microservices tracing with Spring Cloud and Zipkin

Questions?