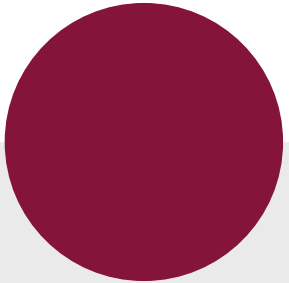
.msg .consulting .solutions .partnership



Logs mit Kontext – Log4j im Vergleich mit Zipkin

JavaLand 2017 – 28. März 2017

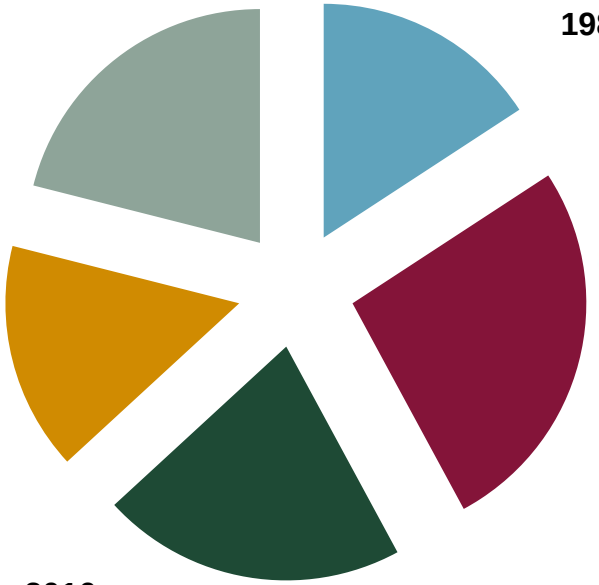
Logging und Tracing mit Kontextinformationen

1. Für die Fehlersuche
2. Kontextinformationen bei Log4j
3. Kontext für Webanwendungen
4. Tracing über Schnittstellengrenzen hinaus
5. Werkzeuge für Entwicklung und Produktion
6. Recap – Was bietet mir welches Werkzeug?

Mein Sponsor und Arbeitgeber – msg systems ag

1980 gegründet

in 18 Städten
in Deutschland präsent



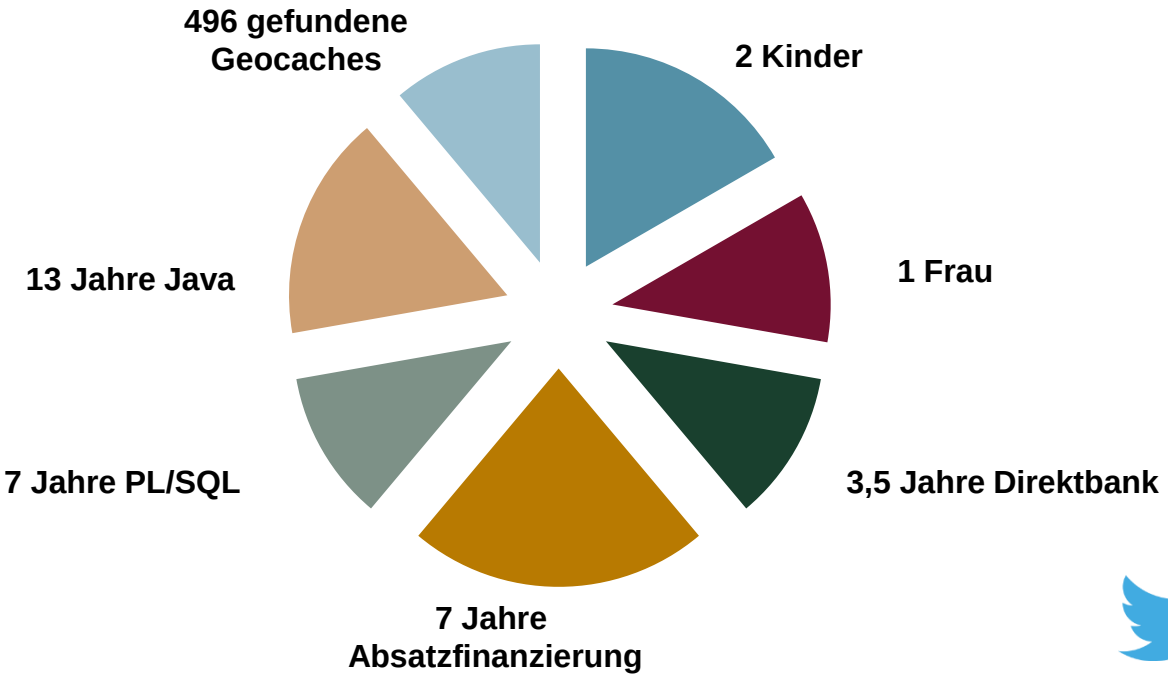
JUnit Lambda

25 Länder

mehr als 6.000 Kollegen

812 Millionen € Umsatz 2016

Wer ich bin – Principal IT Consultant im Geschäftsbereich Travel & Logistics

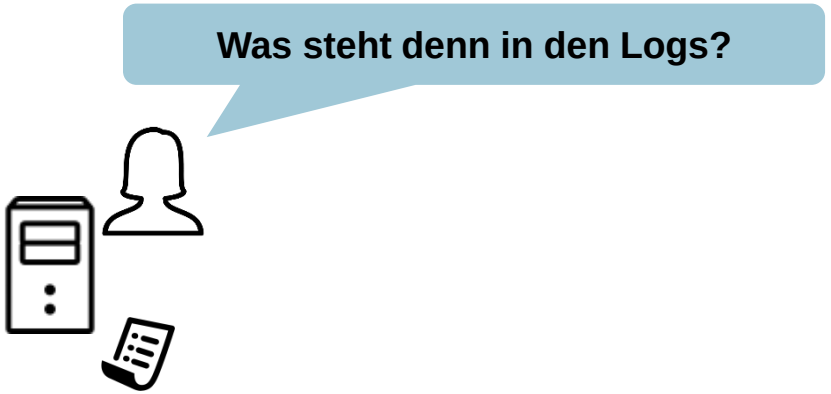


@ahus1de

Logging und Tracing mit Kontextinformationen

1. **Für die Fehlersuche**
2. Kontextinformationen bei Log4j
3. Kontext für Webanwendungen
4. Tracing über Schnittstellengrenzen hinaus
5. Werkzeuge für Entwicklung und Produktion
6. Recap – Was bietet mir welches Werkzeug?

Ein Fehler ist aufgetreten



Willkommen in der Logging-Hölle!

Log:

```
07:26:00.595 d.a.t.d.Invoice ERROR - can't load item ID 4711
```

Code:

```
for (Invoice i : invoiceRespository.findAll()) {
    i.calculateTotal();
}
```

Logging und Tracing mit Kontextinformationen

1. Für die Fehlersuche
2. **Kontextinformationen bei Log4j**
3. Kontext für Webanwendungen
4. Tracing über Schnittstellengrenzen hinaus
5. Werkzeuge für Entwicklung und Produktion
6. Recap – Was bietet mir welches Werkzeug?

Steckbrief log4j

Features:

API für Logging, Implementierung für verschiedene Formate

asynchrones Logging, Re-Konfiguration zur Laufzeit, Filter-Support

Verfügbarkeit und Lizenz:

- Apache Projekt unter Apache License 2.0

Historie:

- 1.2 verfügbar seit Mai 2002 (seit 2015 end of life)
- 2.0 verfügbar seit Juli 2014
- 2.7 verfügbar seit Oktober 2016

Logging mit Kontextinformationen – log4j 1.x und slf4j

```
for (Invoice i : respository.findAll()) {
    MDC.put(INVOICE_ID, Long.toString(i.getId()));
    try {
        i.calculateTotal();
    } finally {
        MDC.remove(INVOICE_ID);
    }
}
```

Logging mit Kontextinformationen – log4j 2.x

```
for (Invoice i : respository.findAll())
    try (final CloseableThreadContext.Instance c
            = CloseableThreadContext.put(
                    INVOICE_ID, String.valueOf(i.getId()))) {
        i.calculateTotal();
    }
}
```

Pattern in der Log-Ausgaben anpassen

Konfiguration:

```
<PatternLayout pattern="%d{HH:mm:ss.SSS} %X %-5level ..."/>
```

Logausgabe:

```
08:39:42.969 {invoiceId=1} ... - can't load item ID 4711
```

Logging und Tracing mit Kontextinformationen

1. Für die Fehlersuche
2. Kontextinformationen bei Log4j
3. **Kontext für Webanwendungen**
4. Tracing über Schnittstellengrenzen hinaus
5. Werkzeuge für Entwicklung und Produktion
6. Recap – Was bietet mir welches Werkzeug?

Filter für mehr Kontextinformationen im Log

```
@Provider
public class RequestFilter implements ContainerRequestFilter {

    @Override
    public void filter(ContainerRequestContext containerRequestContext) {
        MDC.put("http.url",
            containerRequestContext.getUriInfo().getRequestUri().toString());
        MDC.put("http.method", containerRequestContext.getMethod());
        Principal principal =
            containerRequestContext.getSecurityContext().getUserPrincipal();
        if(principal != null) {
            MDC.put("user", principal.getName());
        } else {
            MDC.put("user", "anonymous");
        }
    }
}
```

Filter für mehr Kontextinformationen im Log

```
@Provider
public class ResponseFilter implements ContainerResponseFilter {

    @Override
    public void filter(ContainerRequestContext containerRequestContext,
                       ContainerResponseContext containerResponseContext) {
        MDC.remove("http.url");
        MDC.remove("http.method");
        MDC.remove("user");
    }
}
```

Logausgabe für Webanwendungen

```
08:52:54.276 {http.method=GET,
    http.url=http://localhost:8080/api/startBillingRun, invoiceId=1,
    user=Theo Tester} ERROR d.a.t.d.Invoice - can't load item ID 4711
```

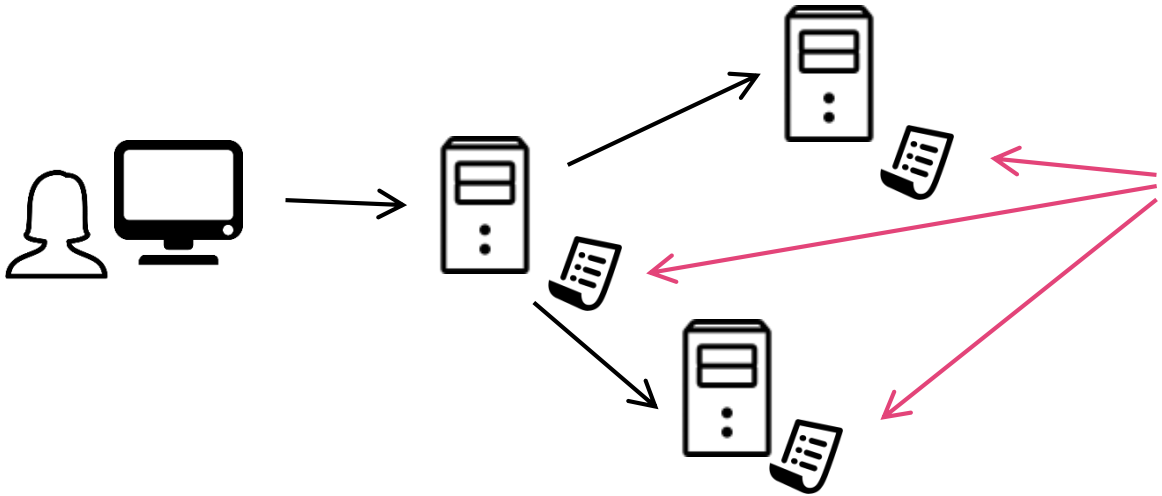
Zusätzlich möglich:

- Client-IP-Adresse
- Teil der Session-ID
- Browser-Kennung (User Agent)

Logging und Tracing mit Kontextinformationen

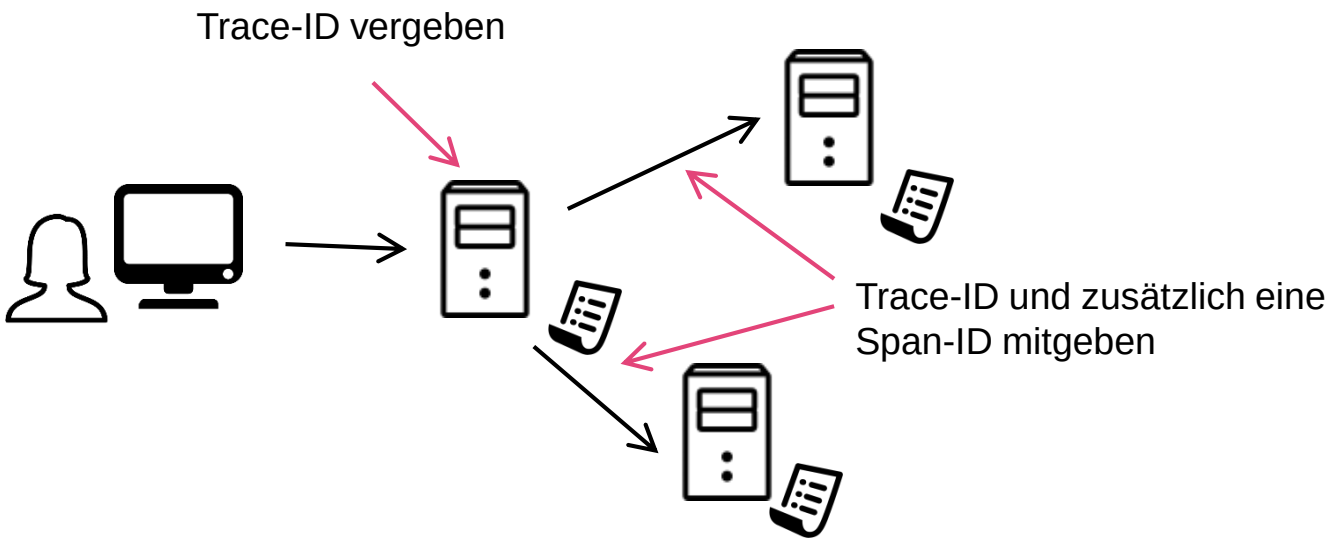
1. Für die Fehlersuche
2. Kontextinformationen bei Log4j
3. Kontext für Webanwendungen
4. **Tracing über Schnittstellengrenzen hinaus**
5. Werkzeuge für Entwicklung und Produktion
6. Recap – Was bietet mir welches Werkzeug?

Nachverfolgung über Systemgrenzen



Ein Aufruf des Nutzers schreibt auf drei verschiedenen Servern Logeinträge

Konzept Dapper in der Umsetzung von Zipkin



Wie es funktioniert

Zusätzliche HTTP Header:

```
GET /api/callback HTTP/1.1
Host: localhost:8080
...
X-B3-SpanId: 34e628fc44c0cff1
X-B3-TraceId: a72f03509a36daae
...
```

Zusätzliche Informationen im Log:

```
09:20:05.840 {
X-B3-SpanId=34e628fc44c0cff1,
X-B3-TraceId=a72f03509a36daae,
..., invoiceId=1} ERROR
d.a.t.d.Invoice - can't load
item ID 4711
```

Was dafür benötigt wird

Zum Selberbauen:

Servlet-Filter zum Auslesen der Informationen bei eingehenden Requests

Wrapper für Aufrufe zu Umsystemen, damit Kontextinformationen gesetzt werden

Für Spring-Boot:

Zusätzliche Depedency spring-cloud-starter-sleuth

Bausteine für andere Projekte:

Zipkin Teilprojekt „brave“ für server- und clientseitige Unterstützung

Logging und Tracing mit Kontextinformationen

1. Für die Fehlersuche
2. Kontextinformationen bei Log4j
3. Kontext für Webanwendungen
4. Tracing über Schnittstellengrenzen hinaus
5. **Werkzeuge für Entwicklung und Produktion**
6. Recap – Was bietet mir welches Werkzeug?

Was ist eigentlich dieses Zipkin?

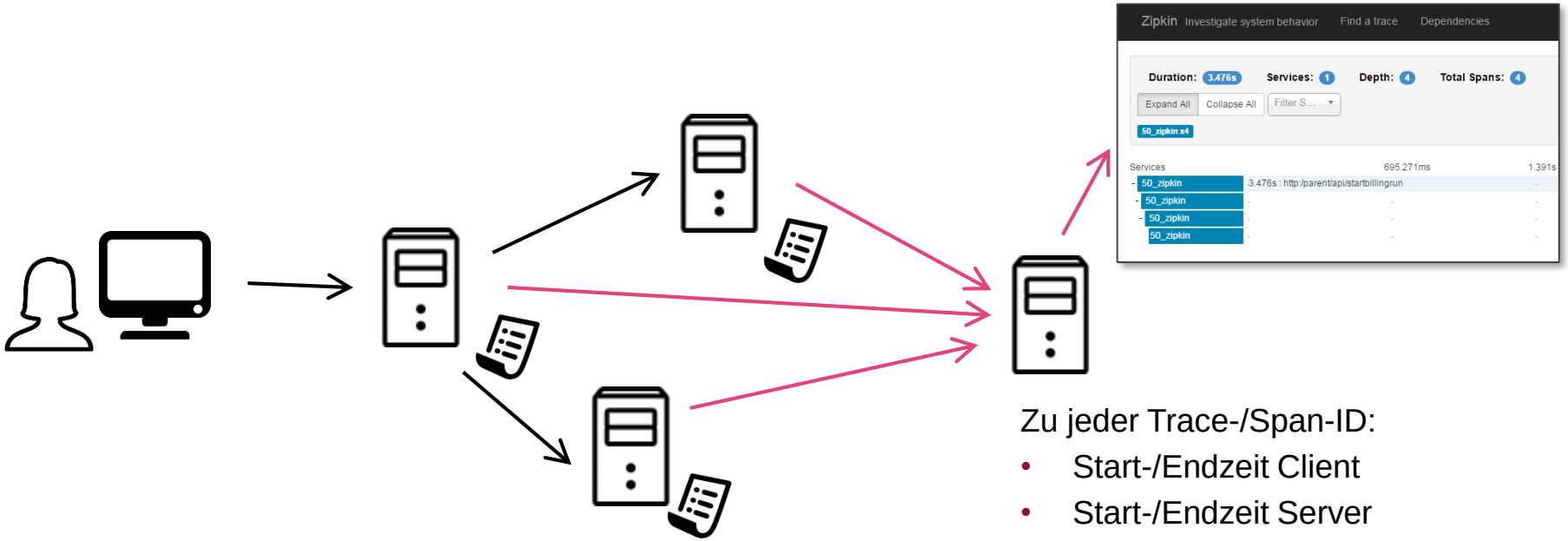
Aufgabe:

- Distributed Tracing
- Sammeln von Kontext- und Timing-Informationen
- Zentrale Sammlung der Informationen für Fehler-, Timing- und Abhängigkeitsanalyse (in Produktion wird meist nur ein kleiner Prozentsatz aller Requests protokolliert)

Konzepte:

- Re-Implementierung der Konzepte von Google Dapper, welches ein internes Closed-Source Werkzeug bei Google ist
- Zuerst umgesetzt als Teil von Finagle bei Twitter, später Open Source

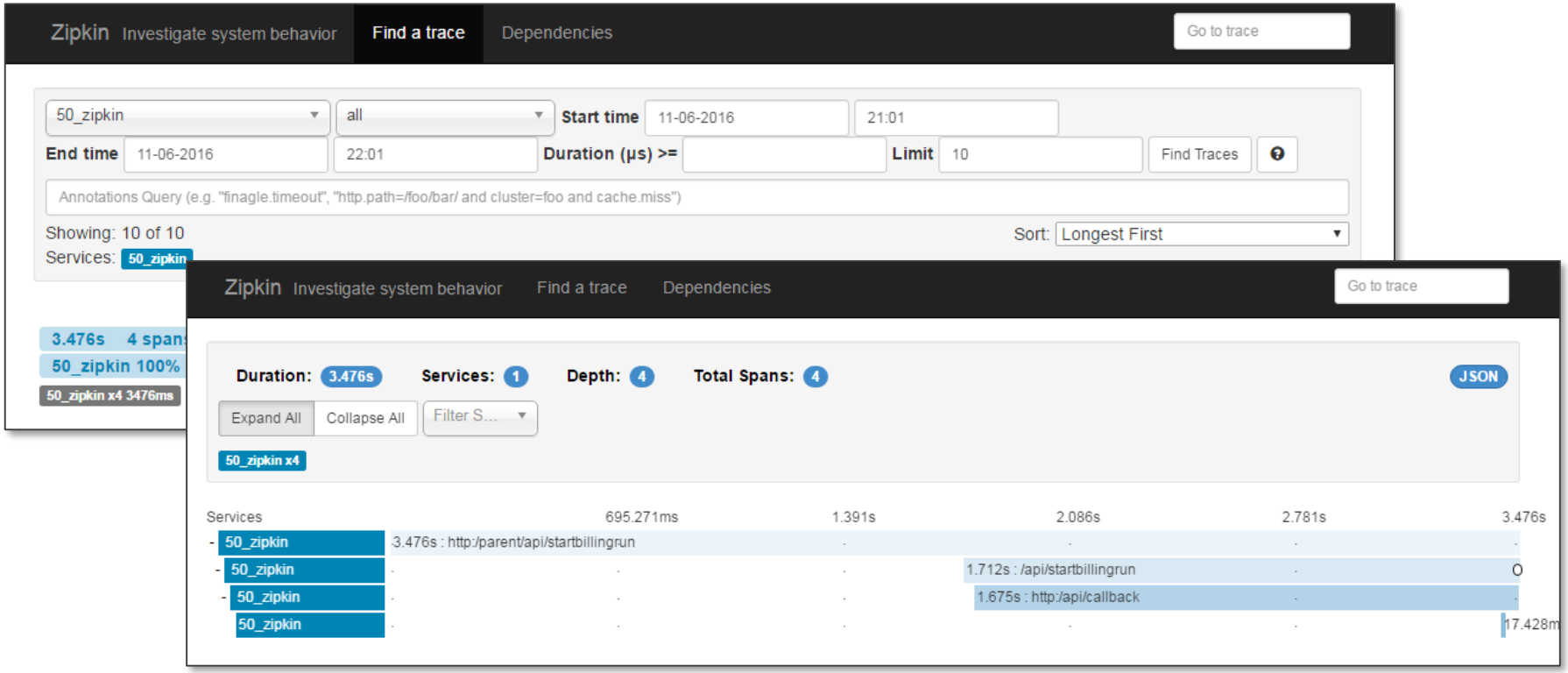
Zipkin Server empfängt Informationen und bietet eine Web-Oberfläche



Zu jeder Trace-/Span-ID:

- Start-/Endzeit Client
- Start-/Endzeit Server
- Tags und Logs

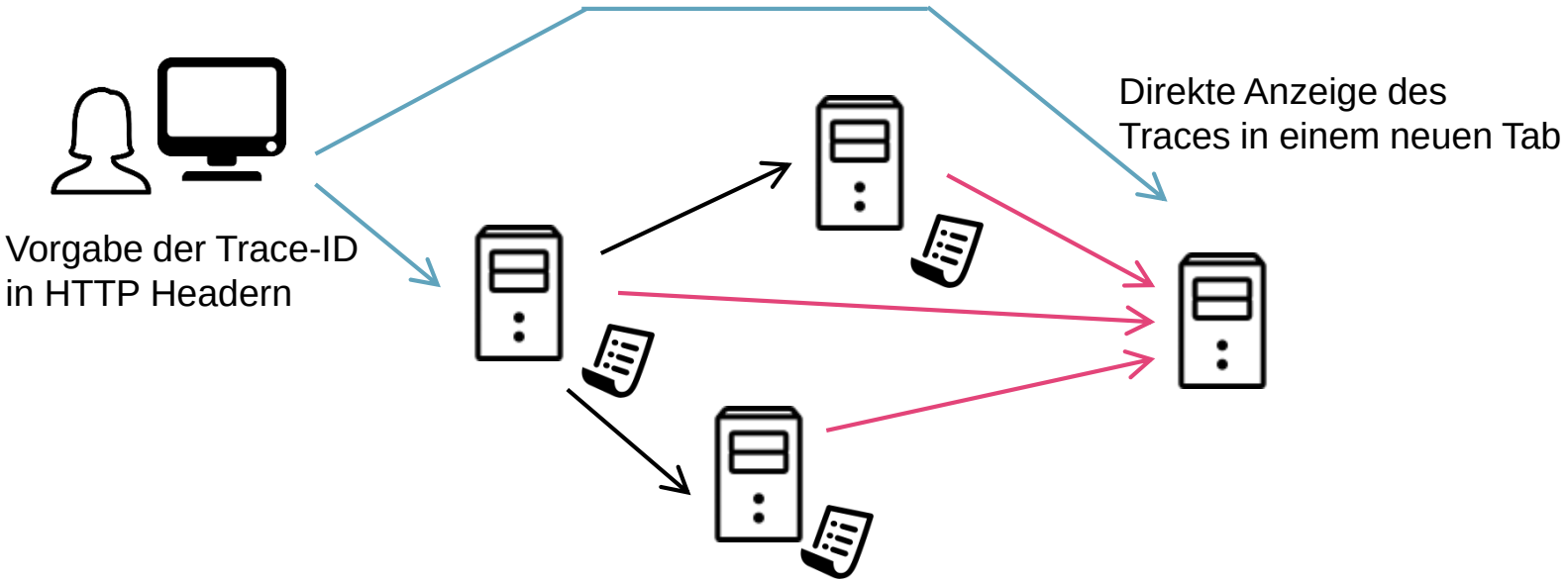
Web-Oberfläche Zipkin



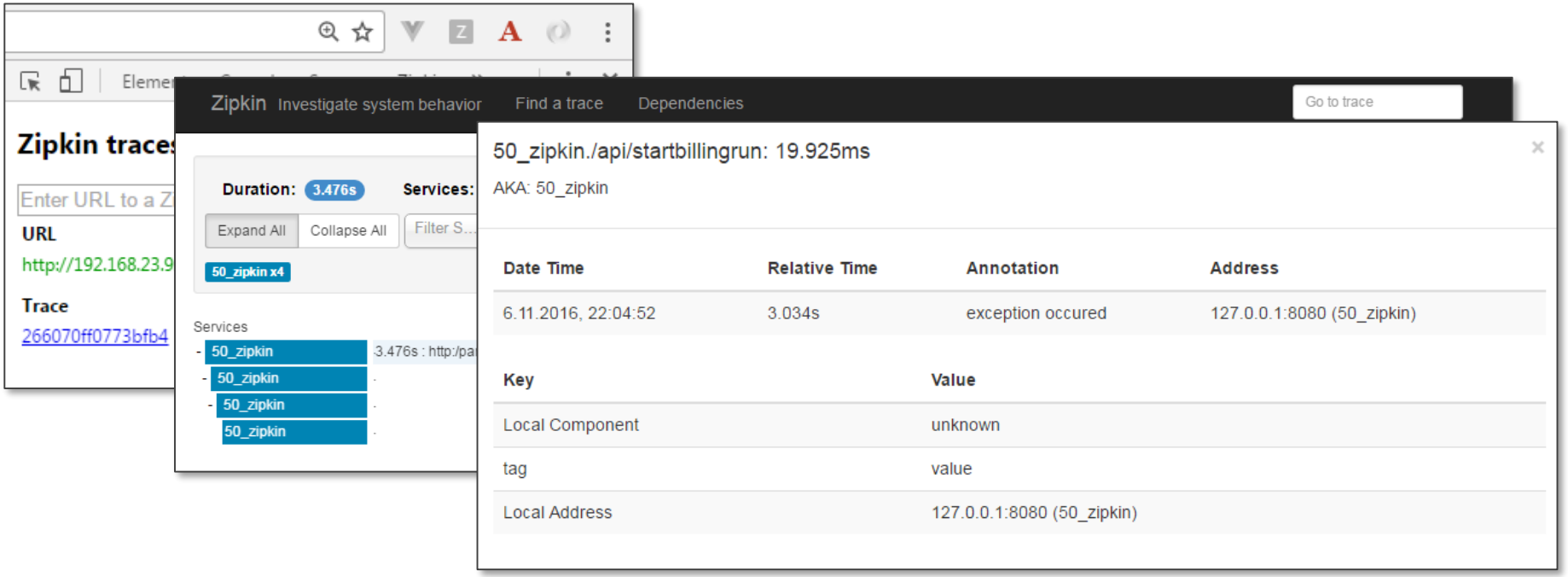
Anreichern mit zusätzlichen Informationen

```
@Component
public class BillingResource {

    @Autowired
    private Tracer tracer;

    public String startBillingRun() {
        /* ... */
        Span span = tracer.getCurrentSpan();
        span.logEvent("log a line");
        tracer.addTag("tag", "value");
        /* ... */
    }
}
```

Zipkin Browser-Plugin für Chrome



Zipkin Browser-Plugin für Chrome



Bonus: Pre-Request-Debugging-Logging

- Zipkin/Sleuth leiten automatisch den Header **X-B3-Flags** weiter, mit dem zusätzliche Informationen übergeben werden können
- Zipkin-Chrome-Plugin setzt den Wert auf „1“ (debug=true)
- Ein eigener Servlet-Filter kann in diesem Fall einen Wert im MDC setzen (z. B. **X-B3-Flags-debug** auf **true**)
- Die folgende log4j2-Konfiguration aktiviert dann das Trace-Level für Requests, die diesen HTTP-Header gesetzt haben.

```
<DynamicThresholdFilter key="X-B3-Flags-debug" onMatch="ACCEPT"
                                defaultThreshold="warn" onMismatch="NEUTRAL">
    <KeyValuePair key="true" value="trace"/>
</DynamicThresholdFilter>
```

Logging und Tracing mit Kontextinformationen

1. Für die Fehlersuche
2. Kontextinformationen bei Log4j
3. Kontext für Webanwendungen
4. Tracing über Schnittstellengrenzen hinaus
5. Werkzeuge für Entwicklung und Produktion
6. **Recap – Was bietet mir welches Werkzeug?**

Was bietet mir welches Werkzeug

Einsatz von…	bietet…
Thread Context / Diagnostic Context	Kontextinformationen in aufgerufenen Funktionen (insbesondere bei Iterationen und Master/Detail Strukturen)
Filter für Web-Aufrufe	Kontextinformationen zum Nutzer und der aufgerufenen URL
Trace-Header á la Dapper mit Zipkin/Brave/Sleuth	Korrelation von Logs über Trace-ID und Span-ID in Entwicklung und Produktion, auf Wunsch auch Pre-Request-Debug-Logs
Zipkin-Server	Durchsuchbare Trace-Information inklusive Timing in Entwicklung und Produktion
Zipkin Browser Plugin	Vorgabe von Trace-IDs, direkte Suche in Entwicklung und Produktion, Erzwingen von Traces (falls in Produktion standardmäßig deaktiviert oder eingeschränkt)

@ahus1de

Links

Log4j 2 – insbesondere NDC/MDC

https://logging.apache.org/log4j/2.x/manual/thread-context.html

Zipkin, Brave

https://github.com/openzipkin/zipkin

https://github.com/openzipkin/brave

Spring Sleuth

https://github.com/spring-cloud/spring-cloud-sleuth

Dapper

http://research.google.com/pubs/pub36356.html

Beispielprojekt

https://github.com/ahus1/logging-and-tracing

@ahus1de

Alexander Schwartz
Principal IT Consultant

+49 171 5625767
alexander.schwartz@msg-systems.com

@ahus1de

msg systems ag (Headquarters)
Robert-Buerkle-Str. 1, 85737 Ismaning
Germany

www.msg-systems.com

.msg .consulting .solutions .partnership