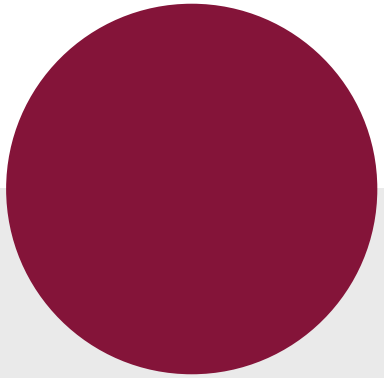
.msg .consulting .solutions .partnership

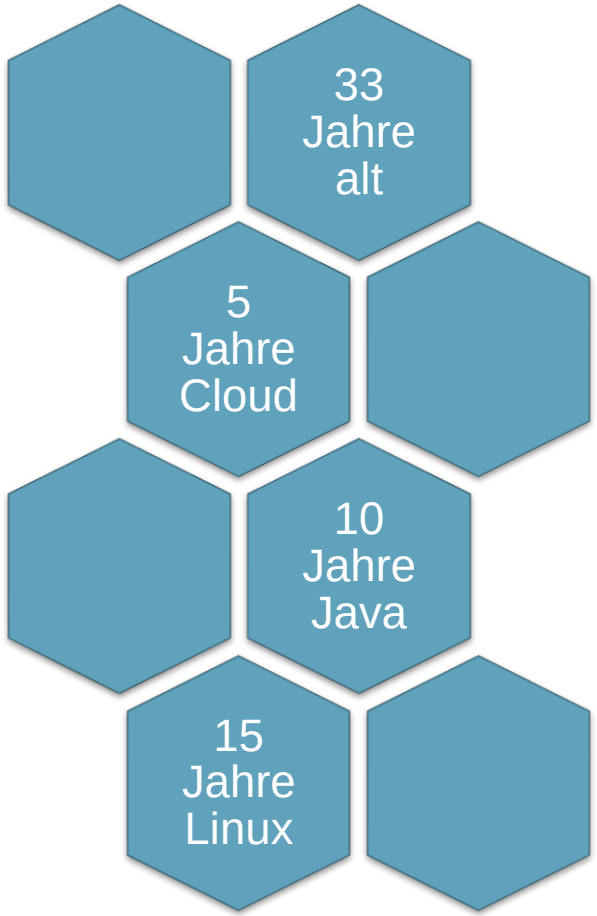


Fährtenlesen für Microservices: Tracing in der Cloud

Björn Kasteleiner, Lead IT Consultant

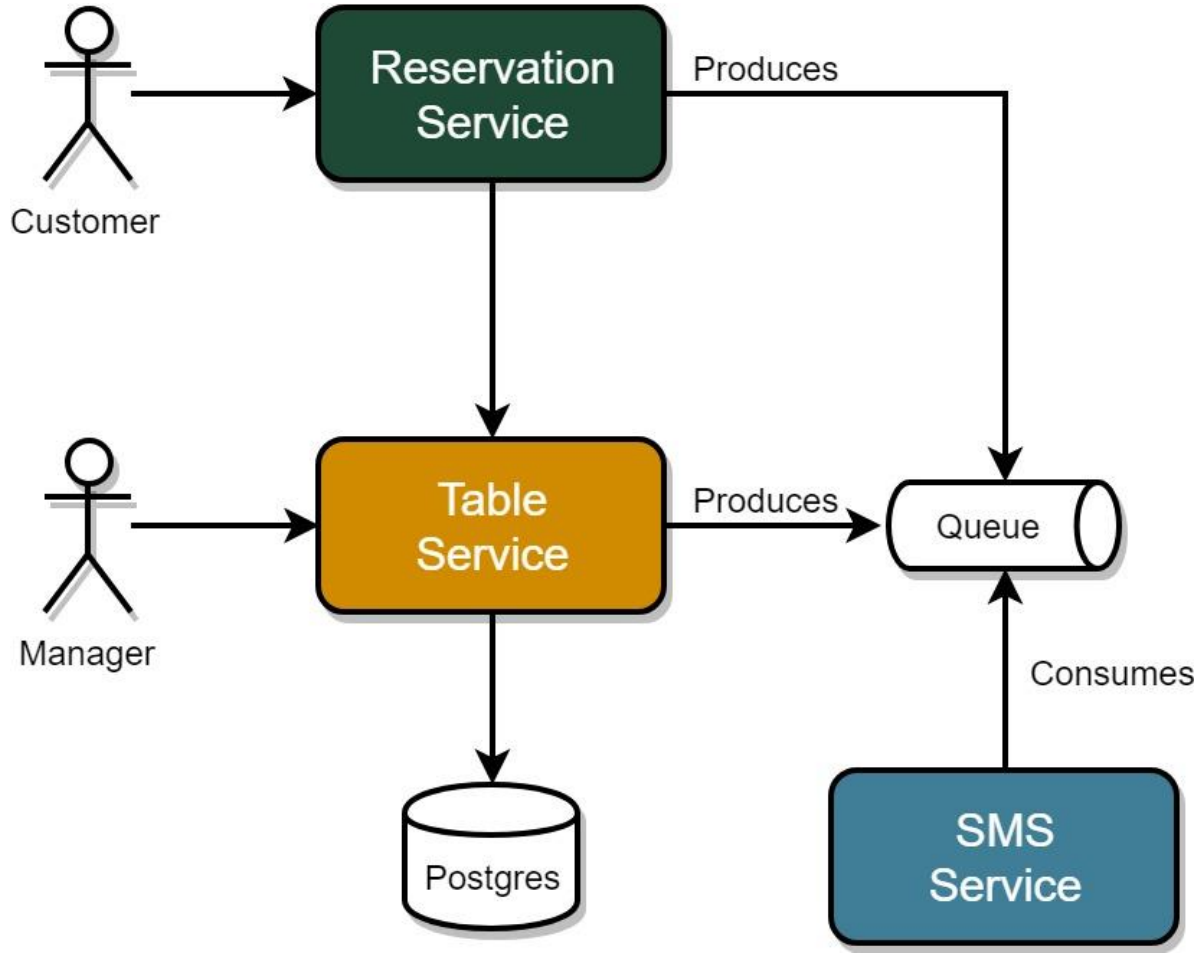
JavaLand 2019

Kurzvorstellung



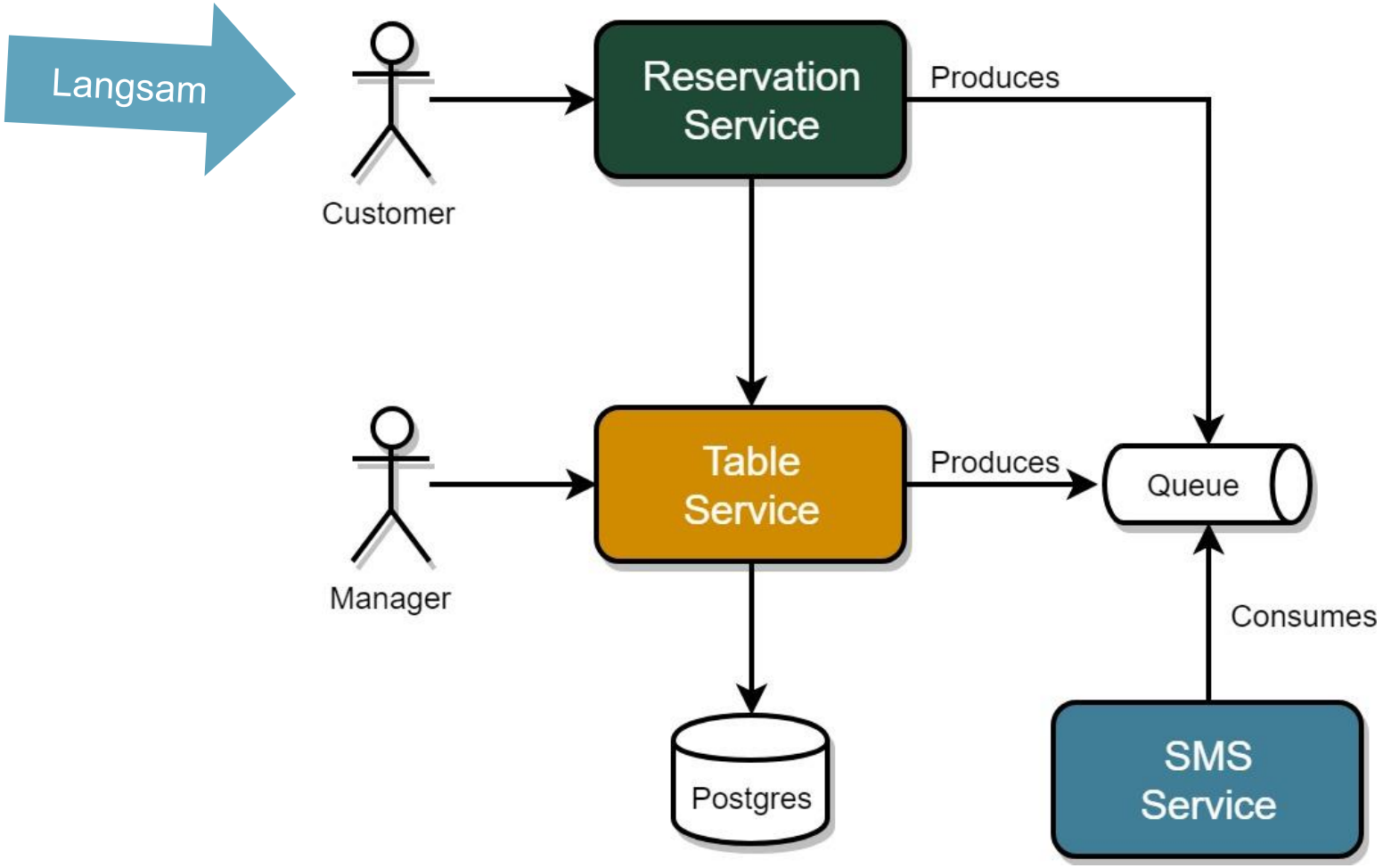
@bjkastel

Unsere Anwendung



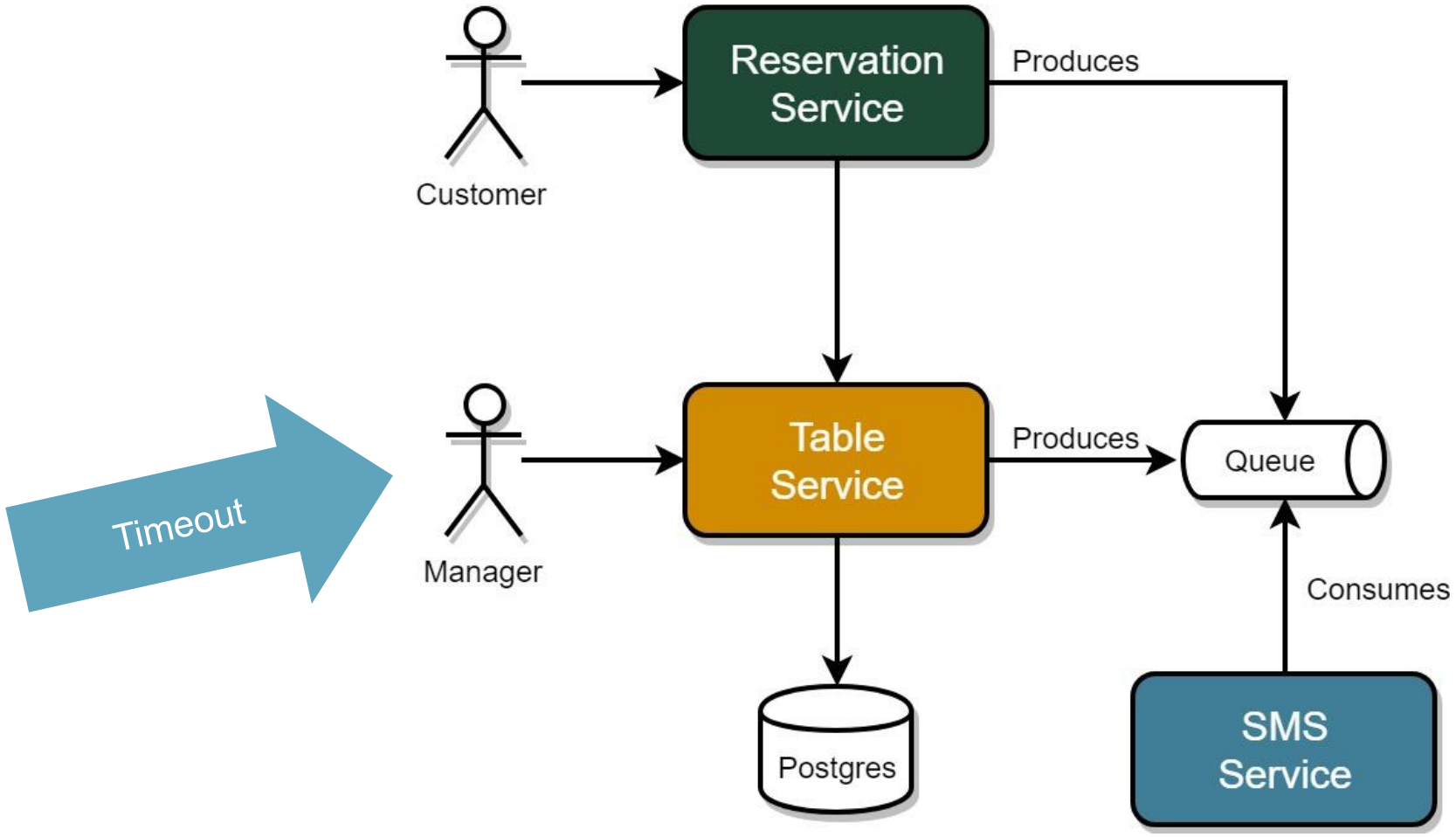
1. https://github.com/bjkastel/sb2-jaeger-sample

Warten…



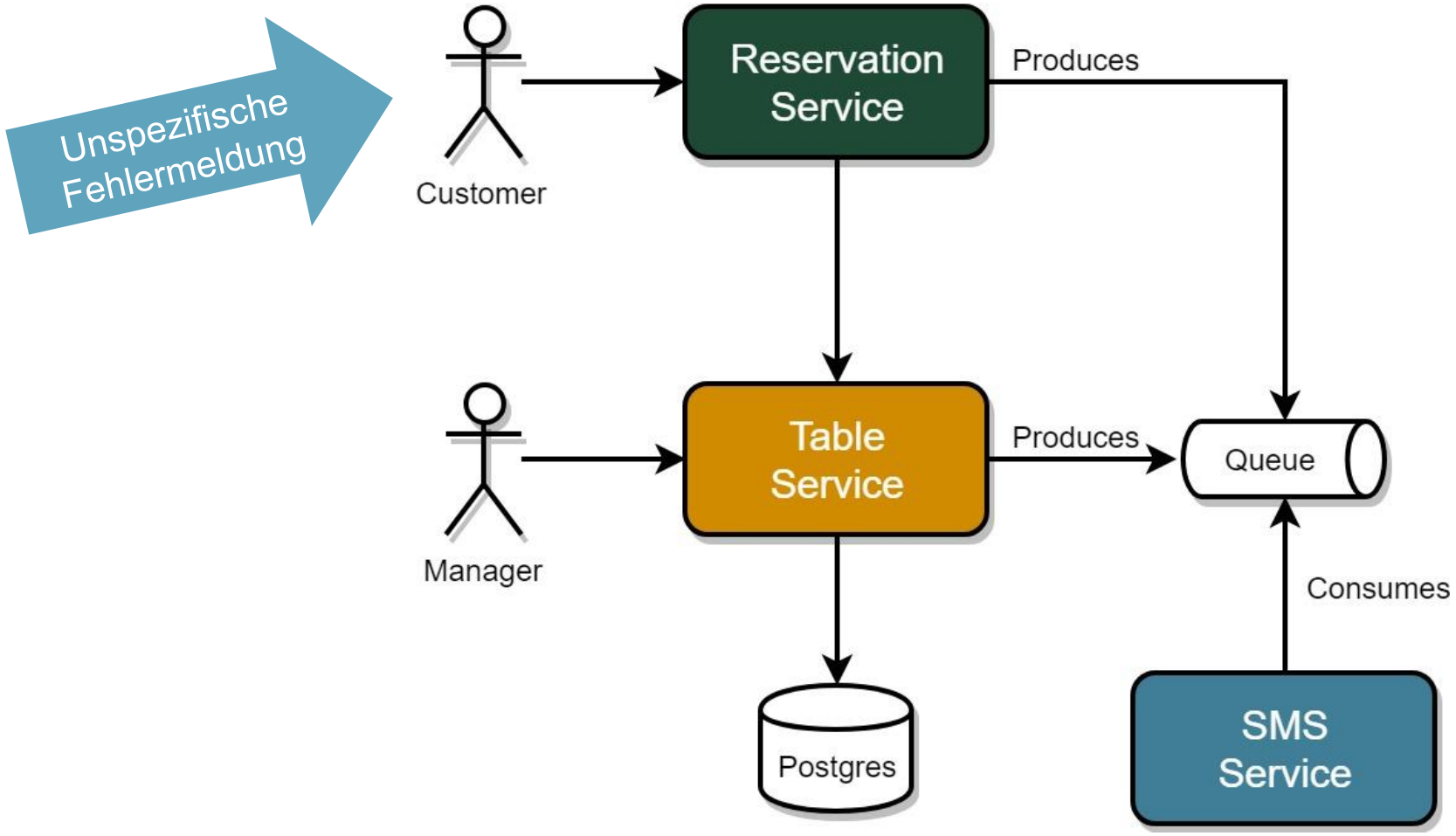
1. https://github.com/bjkastel/sb2-jaeger-sample

Warten…



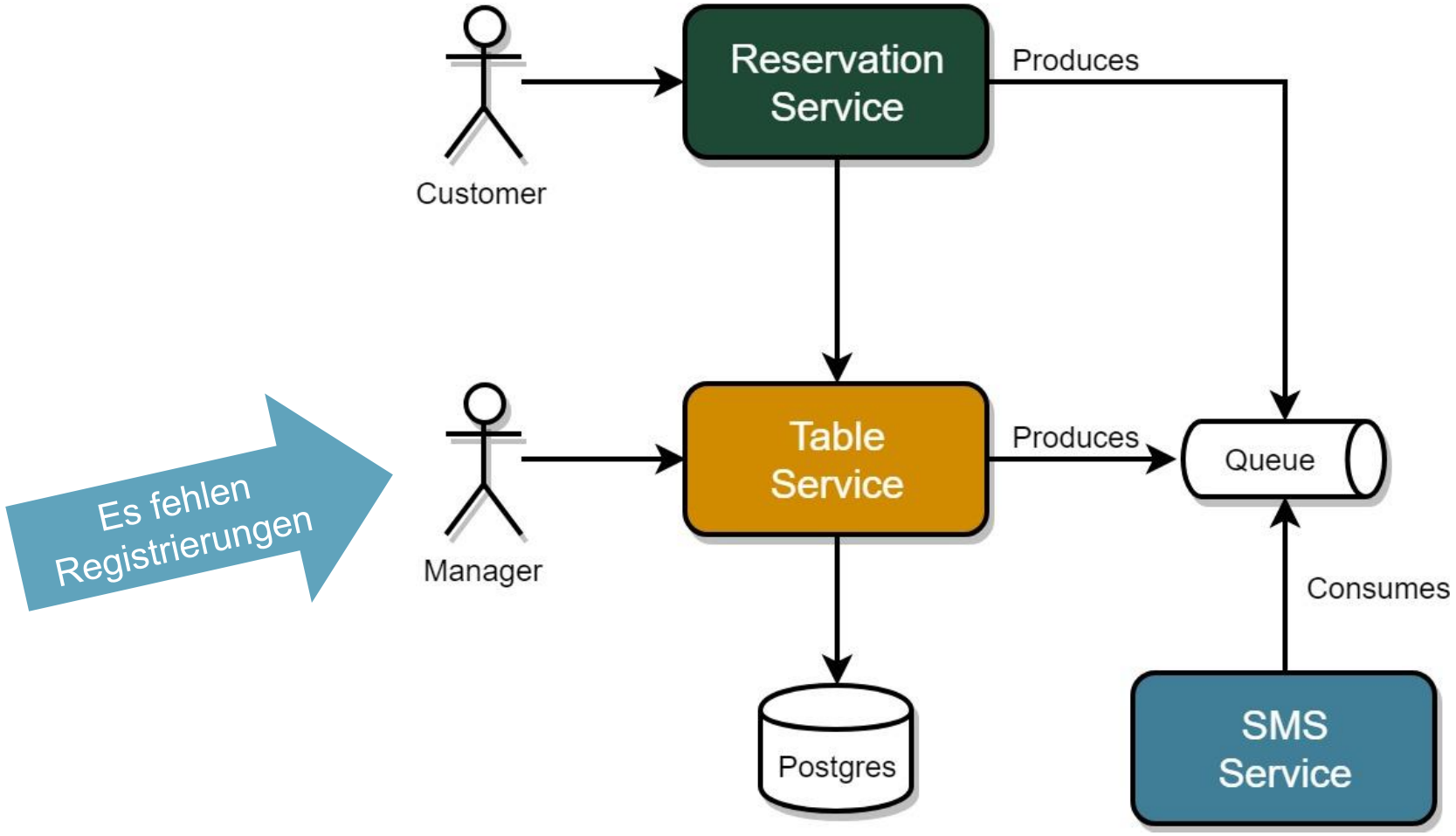
1. https://github.com/bjkastel/sb2-jaeger-sample

Es funktioniert nicht



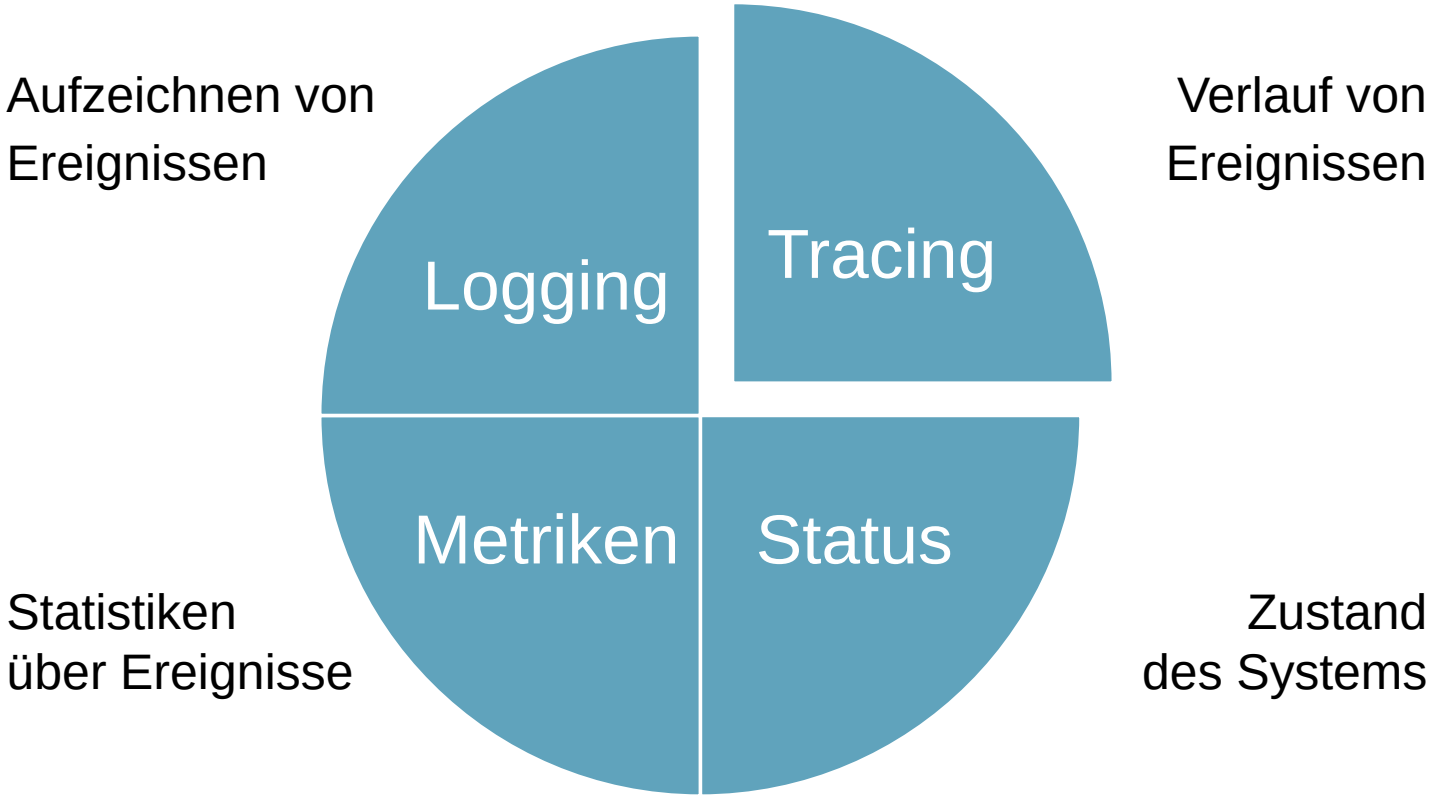
1. https://github.com/bjkastel/sb2-jaeger-sample

Es funktioniert nicht

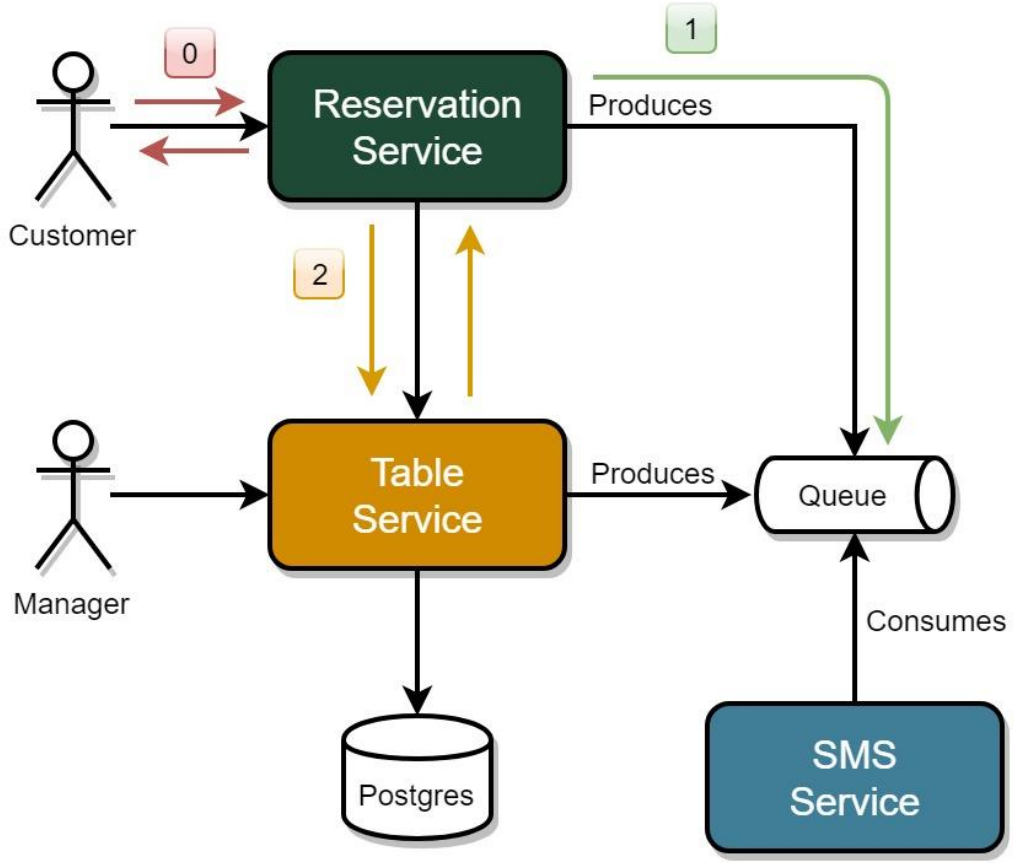


1. https://github.com/bjkastel/sb2-jaeger-sample

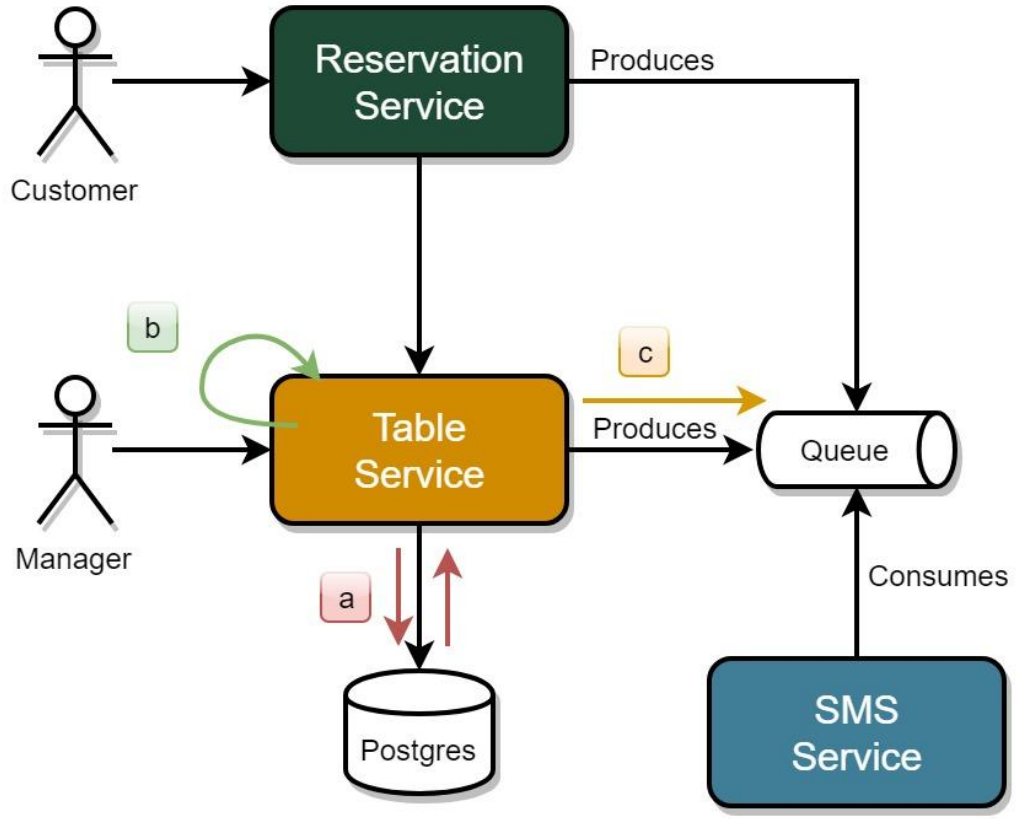
Observability



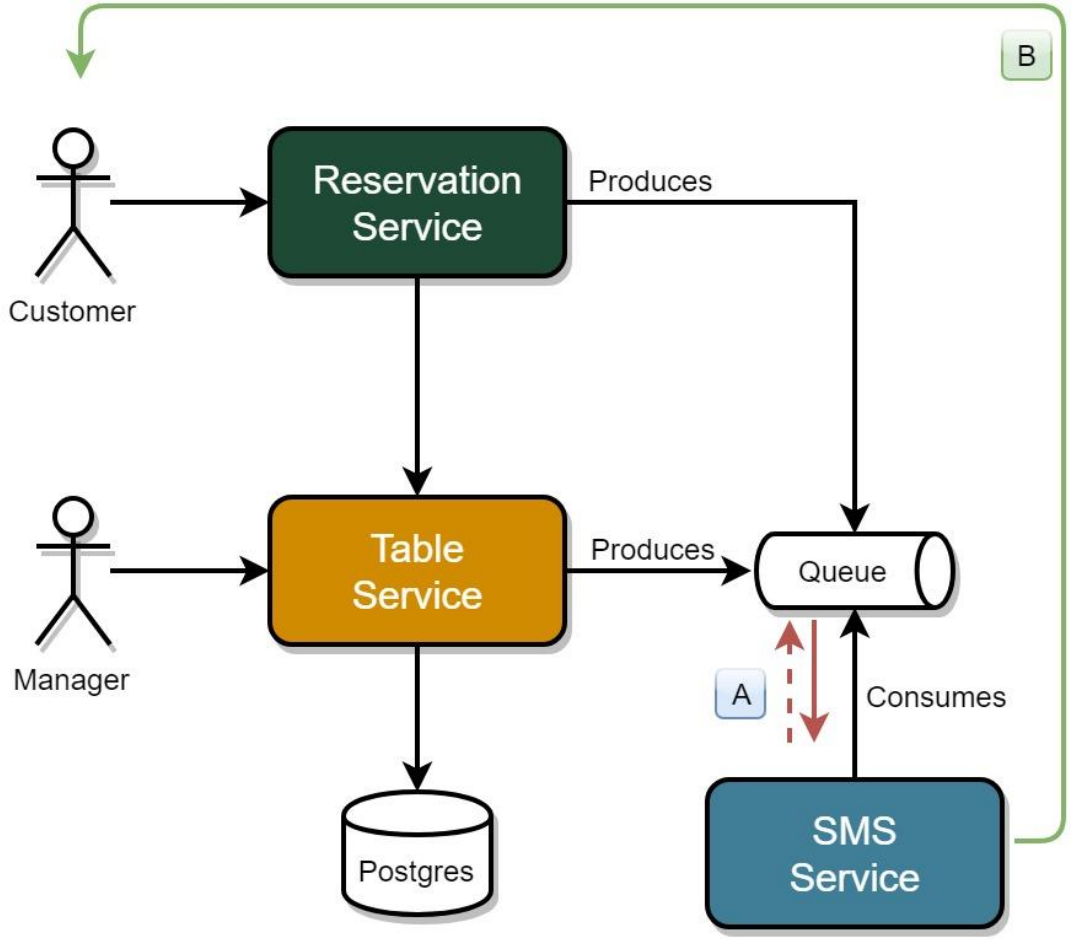
Reservierungswunsch anlegen



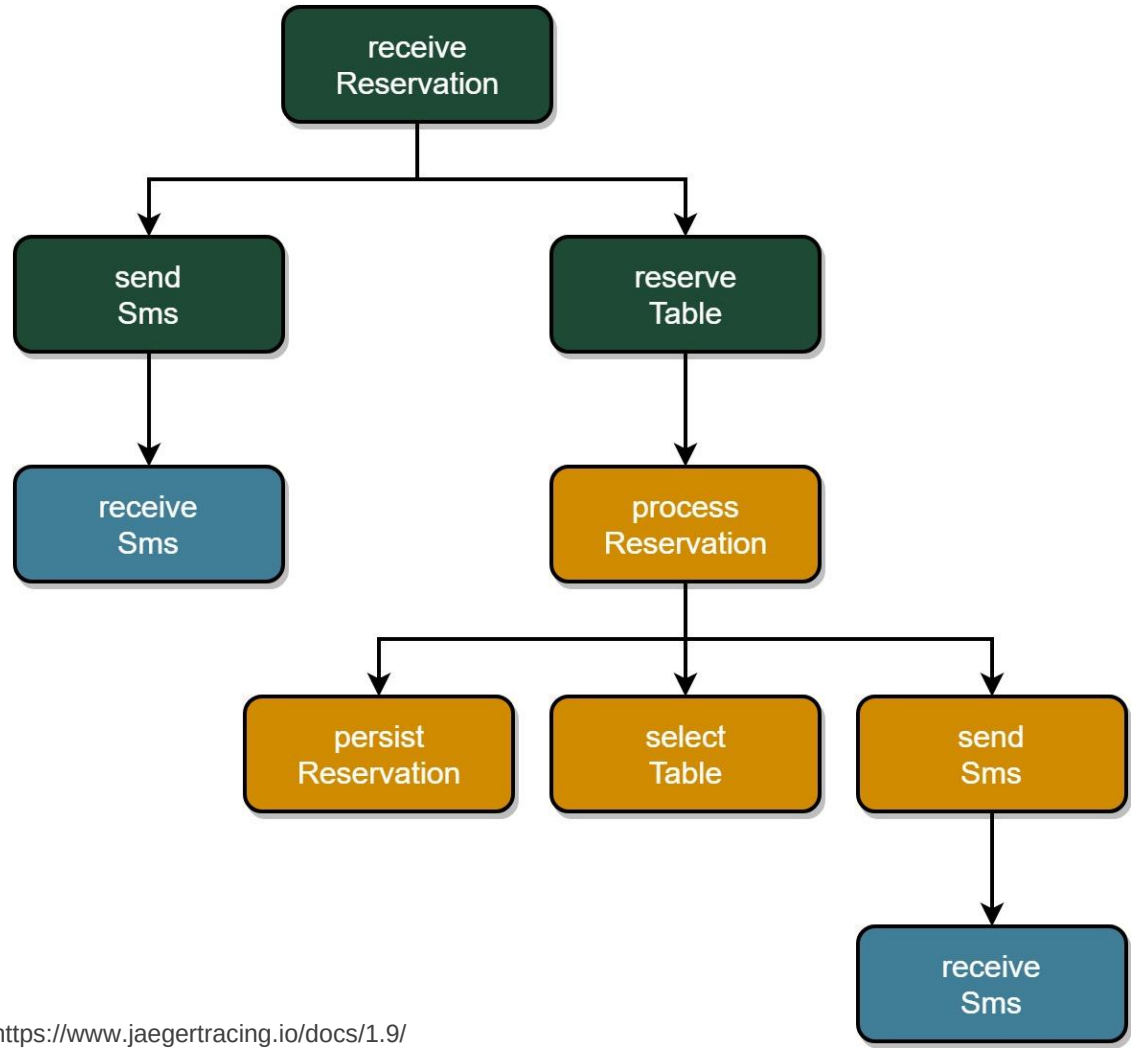
Verarbeitung der Reservierung



Benachrichtigung des Kunden



Trace

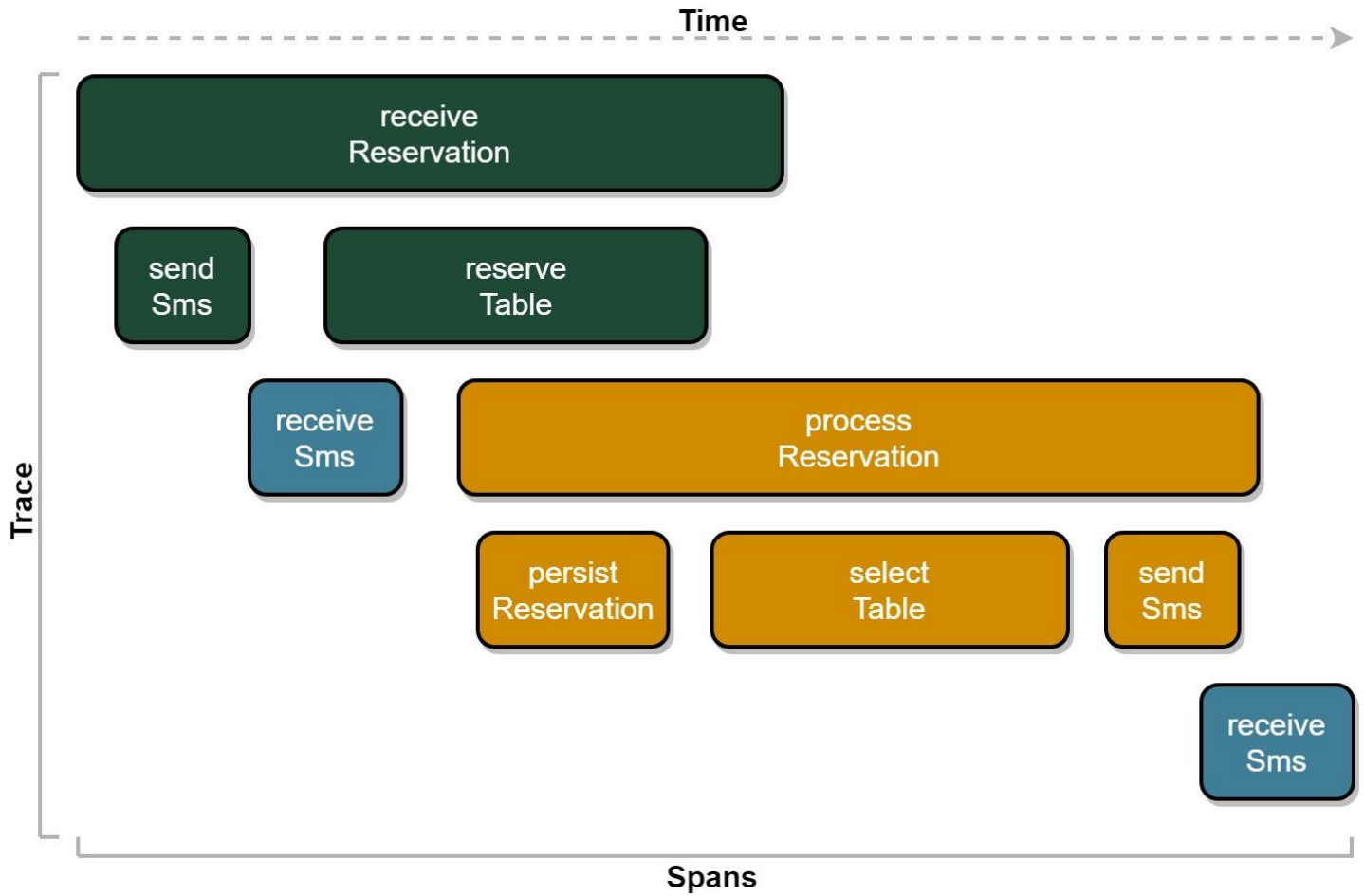


- Beschreibt eine Programmsequenz
- Besteht aus mehreren Operationen
- Koordination erfolgt über Meta-Informationen
  - Trace Id
  - Span Id
  - Parent Span Id
  - Flags

1. https://www.jaegertracing.io/docs/1.9/

Spans

- Einzelne Operationen in einem Aufruf
- Enthält Informationen über
  - Beginn
  - Ende
  - Komponentenname
  - Logeinträge
  - Tags (Key/Value-Paare)



1. https://www.jaegertracing.io/docs/1.9/

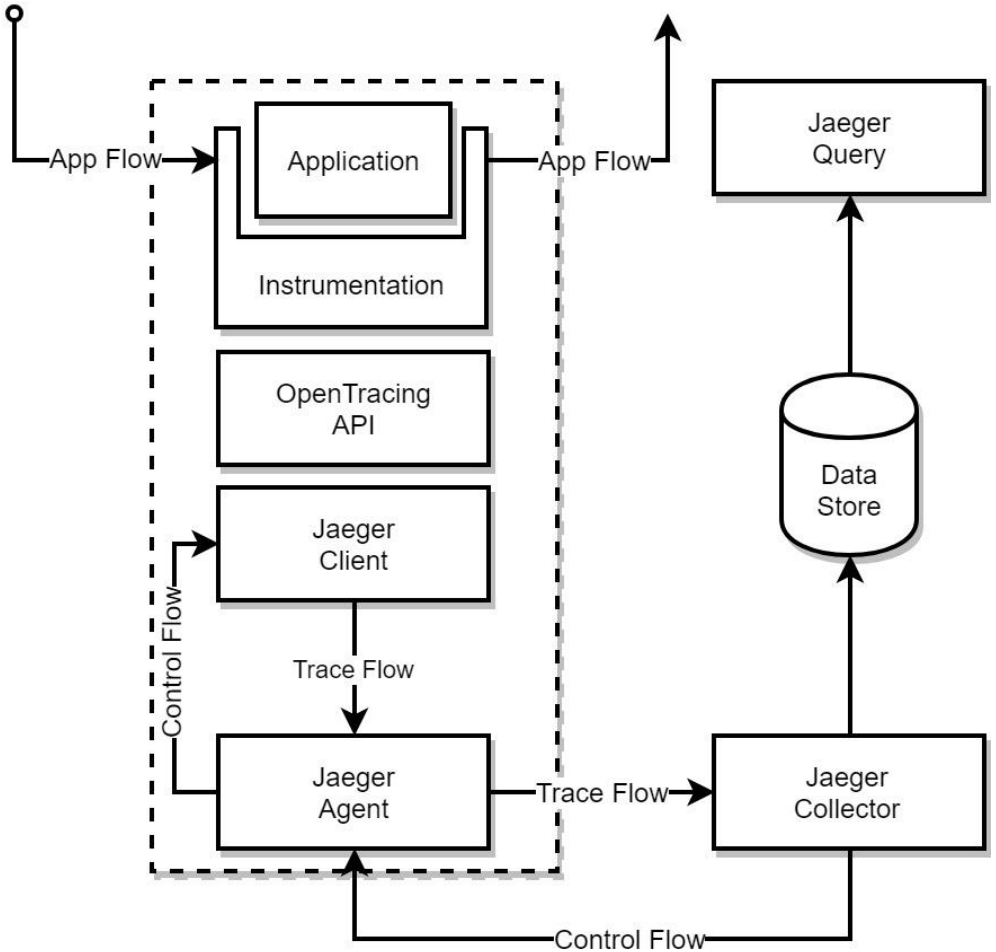
Instrumentierung mit OpenTracing

- Vendor neutrale API für verteiltes Tracing
- Semantische Spezifikation
- Plattform übergreifend
- Anbindung Tracing Implementierungen
- Ermöglicht Instrumentierung des Programmcodes
- Cloud Native Computing Foundation

- Kein Standard
- Überlässt die Weitergabe von Trace-Kontext-Metadaten der Tracing Implementierung

1. https://opentracing.io/
2. Logo: https://github.com/cncf/artwork

Implementierung mit Jaeger Tracing

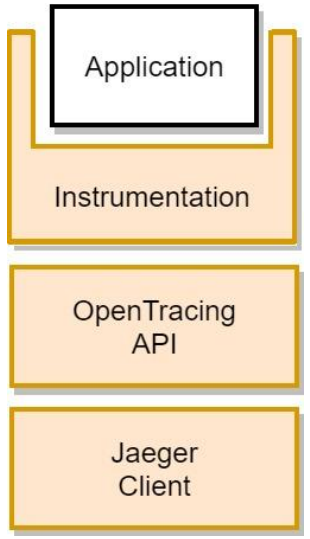


- Implementierung der OpenTracing API
- Cloud Native Computing Foundation
- Ursprünglich von Uber entwickelt
- Datenhaltung
  - Cassandra Database
  - Elastic Search
- Zusätzliche Funktionen
  - Vergleichen von Traces
  - Abhängigkeiten zwischen Services darstellen
  - Unterstützung für Sampling

1. https://www.jaegertracing.io/docs/1.9/architecture/#components
2. Logo: https://github.com/cncf/artwork

Integration mit Spring Boot

- Vorbereitete Spring Boot Starter
  - Vorbereitete Konfiguration
  - Automatische Integration



- Nur ein Maven Dependency Eintrag nötig:

```
<dependency>
    <groupId>io.opentracing.contrib</groupId>
    <artifactId>opentracing-spring-jaeger-cloud-starter</artifactId>
    <version>1.0.1</version>
</dependency>
```

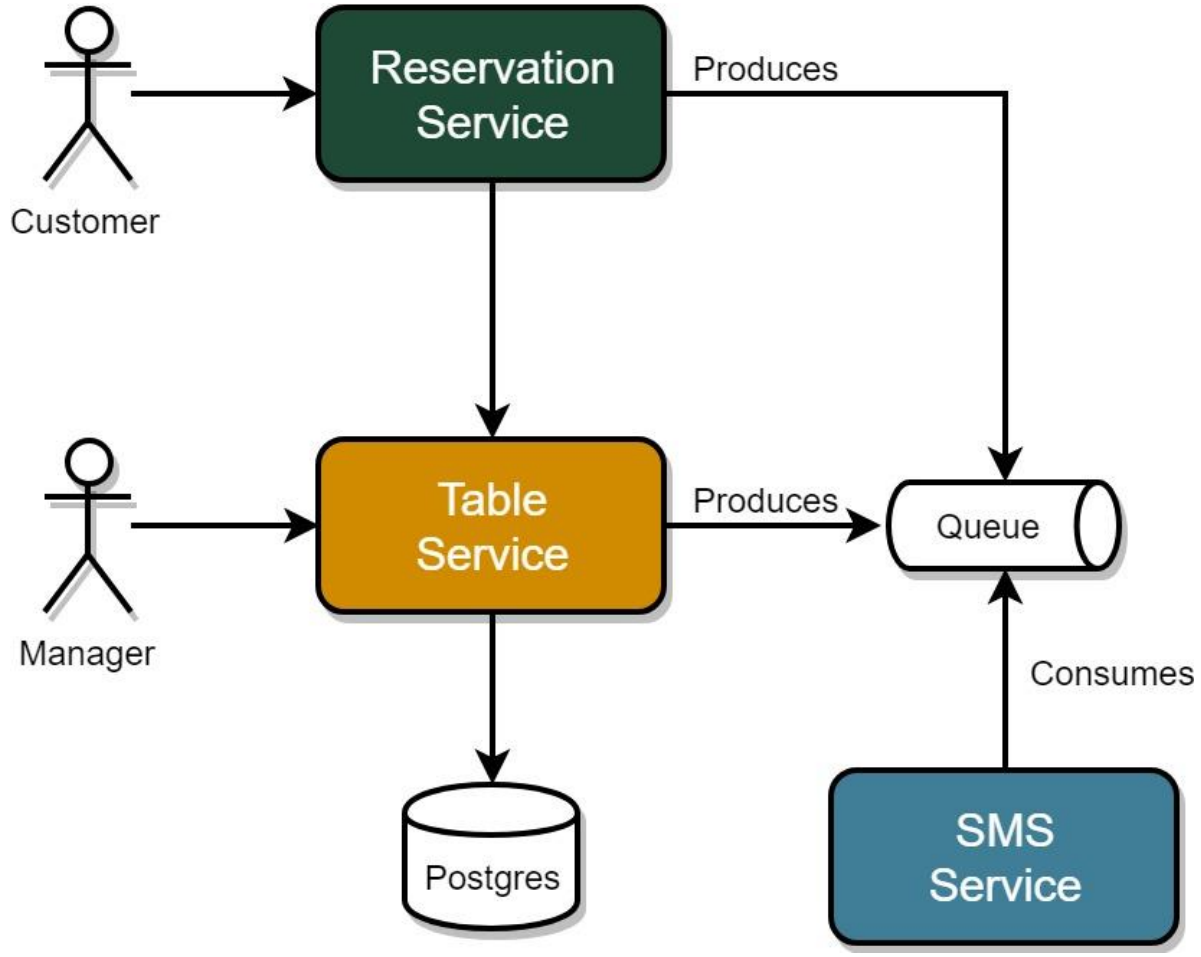
1. https://github.com/opentracing-contrib/java-spring-jaeger
2. https://github.com/opentracing-contrib/java-spring-cloud

Vorhandenen Instrumentierungen

- Spring
  - RestControllers
  - RestTemplates
  - WebAsyncTask
  - @Async
  - @Scheduled
  - Executors
- Messaging
  - JMS
  - RabbitMQ
  - WebSocket STOMP
  - Spring Messaging
- Datenbanken
  - JDBC
  - Mongo
  - Redis
- Standard Logging
- Weitere
  - Zuul
  - RxJava
  - Feign
  - HystrixFeign
  - Hystrix

1. https://github.com/opentracing-contrib/java-spring-jaeger
2. https://github.com/opentracing-contrib/java-spring-cloud

Aufbau der Anwendung



1. https://github.com/bjkastel/sb2-jaeger-sample

DEMO

Tracing Informationen anreichern

Spans um Informationen durch Tags erweitern

```
// ------------------------------------------------------------
// Add waitTime as Tag to current Span
tracer.activeSpan().setTag("waitTime", waitTime);
// ------------------------------------------------------------
```

Eigene Spans mit Informationen über den fachlichen Kontext erzeugen

```
// ------------------------------------------------------------
// Open own Span for the selectTable.
// It's using try-with-resources feature to close the span automatically.
try (Scope scope = tracer.buildSpan(„selectTable").startActive(true)) {
    waitTime = selectTable();
    scope.span().setTag("waitTime", waitTime);
}
// ------------------------------------------------------------
```

Sampling / Adaptives Sampling

- Reduzierung der Rate, mit der Traces aufgezeichnet werden

- Varianten:
  - Constant Sampling
  - Probabilistic Sampling (1 von 10 Traces)
  - Rate Limiting (Traces pro Sekunde)
  - Remote

- Adaptives Sampling
  - Entscheidet auf Basis von bestimmten Operationen (Spans)
  - Mindestrate + Probabilistic Sampling

1. Bild: https://rrze-pp.github.io/rrze-icon-set/tango/gallery.html#filter von Regional Computing Centre of Erlangen (RRZE) ist lizenziert gemäß CC BY-SA

Fazit

- Tracing hilfreich als Ergänzung zur Fehlersuche
- Leichte Integration
- Instrumentierung gängiger Spring Boot Funktionen unterstützt
- Informationen über fachlichen Kontext in Traces integrierbar
- Einigung auf eine Tracing-Implementierung ist (aktuell) notwendig

Links

- Entwurf der Tracing-Header Spezifikation des W3Cs
  https://w3c.github.io/trace-context/
- Webseite des OpenTracing Projekts
  https://opentracing.io
- Webseite des Jaeger Tracing Projekts
  https://www.jaegertracing.io
- Spring Boot Integration von OpenTracing mit Jaeger Tracing
  https://github.com/opentracing-contrib/java-spring-jaeger
  https://github.com/opentracing-contrib/java-spring-cloud

- Code des Beispiel-Projekts
  https://github.com/bjkastel/sb2-jaeger-sample

@bjkastel

Björn Kasteleiner
Lead IT Consultant

bjoern.kasteleiner@msg.group

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

www.msg.group

.msg .consulting .solutions .partnership