TRIDENT

Von Nixdorf zur Cloud City

Oliver Böhm

Software-Entwicklung: Lerne aus der Vergangenheit, träume von der Zukunft und entwickle in der Gegenwart. Frei nach Katharina [1]

Dies ist die Geschichte einer Firma, die auszog, die Rezept-Abrechnung zu vereinfachen, um Ärzten, Apothekern und anderen Leistungserbringern im Gesundheitswesen (wie Physio-/Ergo-Therapeuten, Logopäden, Hebammen) Freiräume für wichtigere Dinge zu geben. Dazu hat das „Optica Abrechnungszentrum", ein Unternehmen der Dr. Güldener Firmengruppe, bereits Ende der 1970er-/Anfang der 1980er-Jahre früh auf Digitalisierung gesetzt. In einem großen Schreibbüro wurden dabei Abrechnungsdaten auf „Datensammelsystemen" – anfangs auf Lochkarten, später auf Magnetbändern (Nixdorf 8850) – gespeichert.

Damals hieß IT noch EDV. Die Firma Nixdorf (später Siemens-Nixdorf) war ein typischer Vertreter der „Mittleren Datentechnik". Lochkarten beziehungsweise Magnetbänder wurden in die EDV-Abteilung gegeben, um daraus die Abrechnungen zu erstellen und in den Druck zu geben. Netzwerke, wie wir sie heute kennen, gab es damals nicht.

Architektonisch waren die Anwendungen für die Nixdorf 8850 nach dem EVA-Prinzip gestrickt: Eingabe – Verarbeitung – Ausgabe, wobei der Fokus auf der Verarbeitung lag (dem „V" in „EDV"). Schöne Oberflächen waren damals kein Thema – Batchverarbeitung, auch als Dunkelverarbeitung bekannt (etwa im Versicherungsbereich), war Stand der Technik. Es galt, den Datenstrom für die Ein- und Ausgabeverarbeitung nicht abreißen zu lassen, um die teuren Maschinen auszulasten. Üblich waren dabei Bänder, von denen gelesen und auf die das Ergebnis geschrieben wurde, Abrechnungslauf für Abrechnungslauf *(siehe Abbildung 1)*.

Mitte/Ende der 1980er-Jahre etablierten sich dann relationale Datenbanken. Diese dienten nicht nur als zentrale Basis zur Ablage, sondern erlaubten auch die Modellierung von Beziehungen. Eine relationale Datenbank der ersten Stunde war Informix *[2]*, die dort auf einem Unix-Server (Sun Microsystems) zum Zuge kam und nach und nach die Nixdorf-Maschine ablösen sollte.

Wie aber bekommt man für die Übergangsphase die neuen Daten in die „alte Welt", die doch nur Datenströme kennt? Durch (digitale) Transformation – man übersetzt die Tabellen aus der relationalen Welt in Records beziehungsweise Dateien der Nixdorf-Welt. In die andere Richtung (Nixdorf nach „neue Welt") übersetzt man Datenströme in Tabellen. Auf diese Art und Weise konnte man die bestehenden Alt-Anwendungen weiterverwenden, gleichzeitig jedoch auch neue Anwendungen auf Basis einer Client-Server-Architektur mit damals zeitgemäßer Oberfläche entwickeln *(siehe Abbildung 2)*.

Mit zunehmender Verbreitung der Unix-Maschinen in den 1990er-Jahren wurde das Internet mit Erfindung des World Wide Web sichtbarer. Damit war die Welt außerhalb des eigenen Netzwerks immer interessanter, sodass man bereits im Jahr 1997 innerhalb der Dr. Güldener-Gruppe mit einer eigenen Homepage präsent war, über die elektronisch Rezepte abgegeben werden konnten.

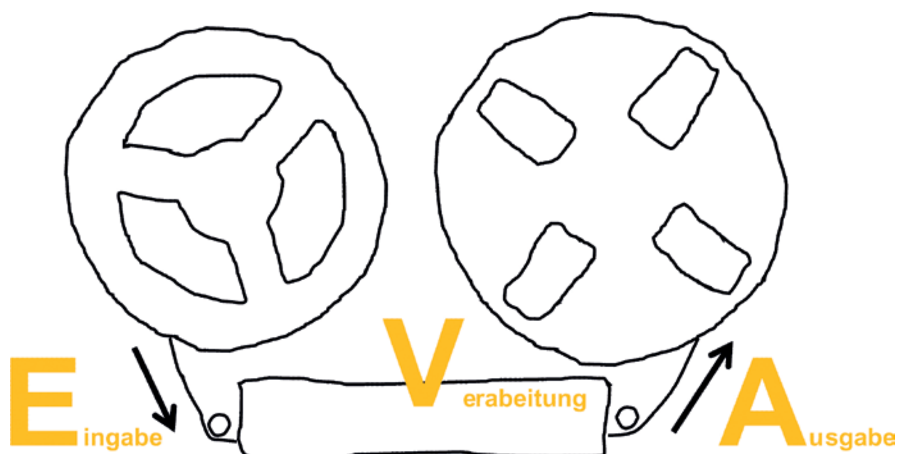

Abbildung 1: Die typische Architektur von damals

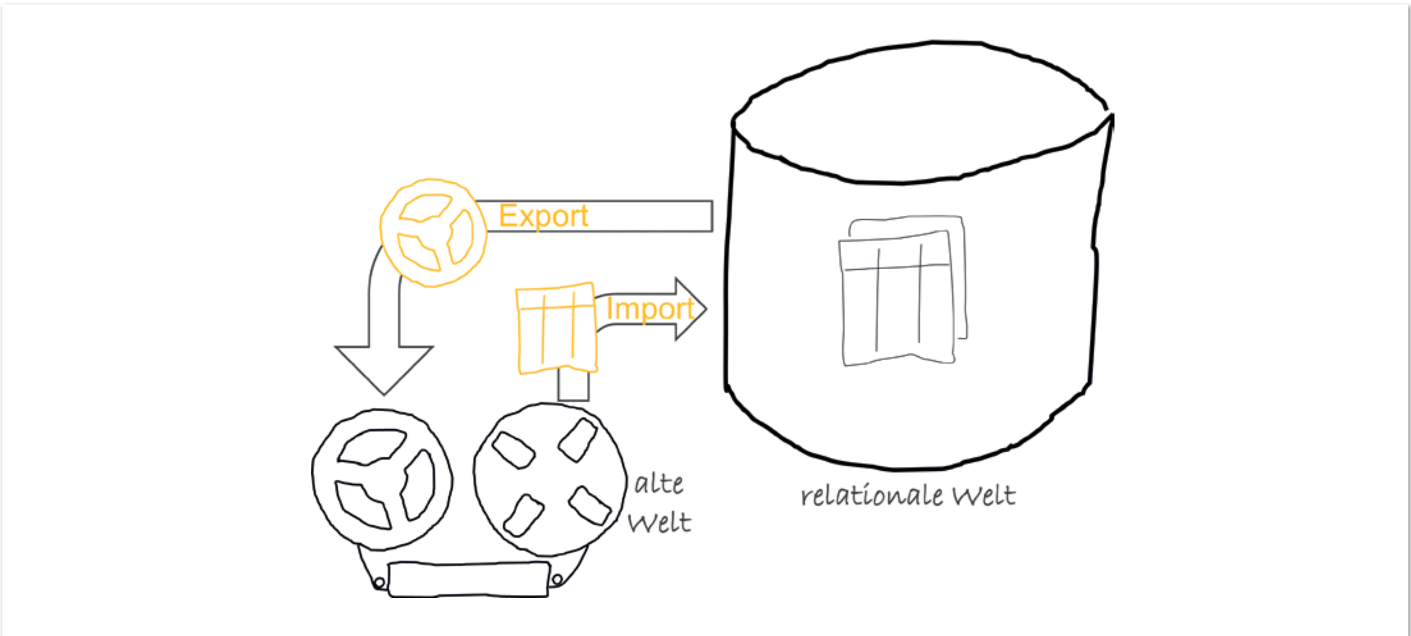

Abbildung 2: Die Verbindung zwischen „alter" und „neuer" Welt

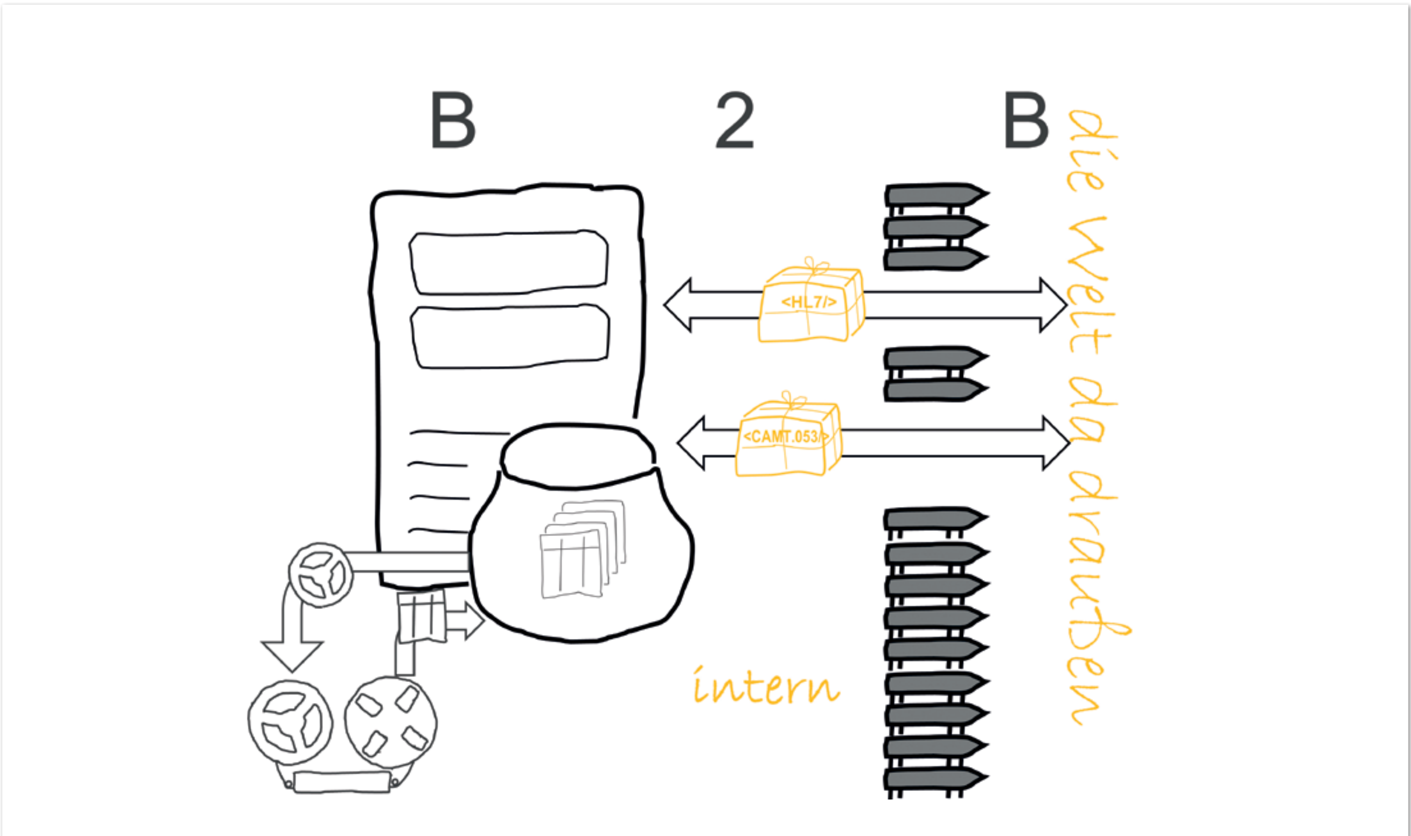

Abbildung 3: Service Oriented Architecture

Mit dem Internet-Hype und Aufstieg des neuen Markts wurden große Hoffnungen in B2C- und B2B-Schnittstellen gesetzt. Auch wenn die Dotcom-Blase Anfang der 2000er-Jahre platzte, hat sich diese Idee in Form von Service Oriented Architecture (SOA) verfestigt und neue Geschäftsfelder erschlossen. Die Datenbank rückte weiter in den Mittelpunkt vieler Anwendungen und wurde ständig erweitert und angepasst *(siehe Abbildung 3)*. Allmählich drohte sie allerdings, zum Flaschenhals zu werden, auch weil der DB-Optimierer an seine Grenzen stieß.

Datenbanken können geclustert werden, was aber nicht zwangsläufig einen Performance-Gewinn bedeutet. Performance-Steigerungen sind insbesondere davon abhängig, ob sich die Daten anhand der Zugriffe aufteilen lassen. Dies führte dann zu grundlegenden Überlegungen, wie eine zukünftige Architektur für die nächsten Jahrzehnte aussehen könnte.

Ein Resultat dieser Überlegungen war die Aufteilung der Anwendungslandschaft nach Domain Driven Design in verschiedene Berei-

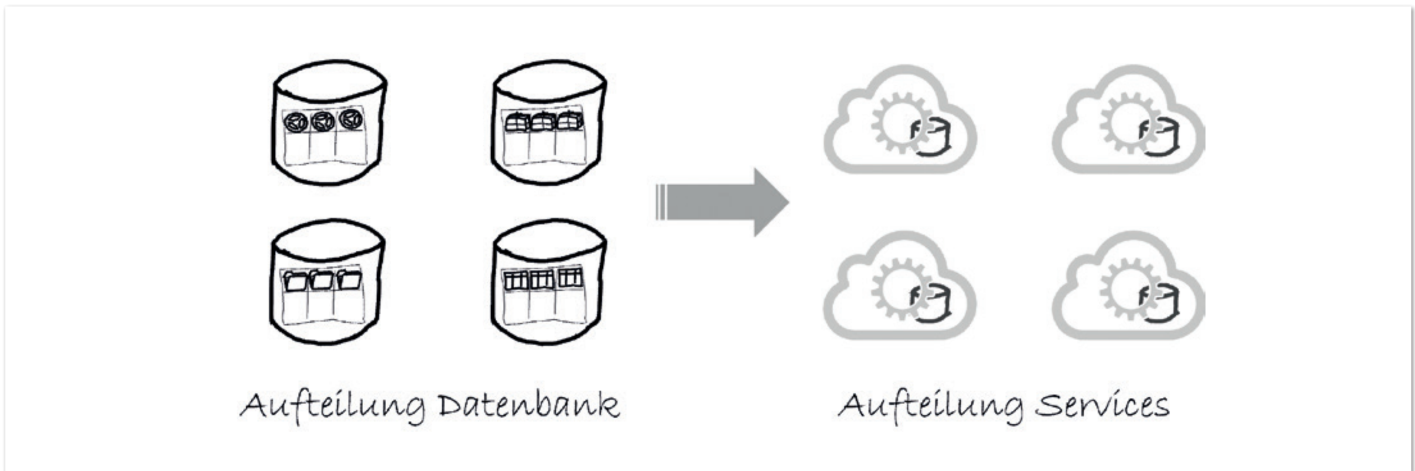

Abbildung 4: Domain Driven Design

che (Stammdaten, Rechnungen, Rezepte), in der jede Domain sein eigenes Datenbank-Schema in Postgres-Datenbanken bekommen sollte. Um damit für die Zukunft gewappnet zu sein, packt man noch einen REST-Service davor, über den alle Anfragen und Abfragen laufen. Damit wird man von der Datenbank unabhängig und kann sie später beispielsweise gegen eine NoSQL-Datenbank austauschen *(siehe Abbildung 4)*.

Eine REST-Schnittstelle hat jedoch noch andere Vorteile: Im Zeitalter der Virtualisierung und Cloud lässt sich die Schnittstelle auch mehrfach starten, wenn die Antwortzeiten zu langsam werden (falls die Datenbank nicht der Flaschenhals ist). Die Architektur ändert sich damit zu einer Microservice-Architektur, die sich einfacher skalieren lässt, allerdings auch höhere Anforderungen an den Betrieb stellt.

Aber nicht nur der Betrieb, auch die Entwicklung wird anspruchsvoller. Nicht umsonst hat Sun Microsystems, Erfinder von Java, aber auch des Slogans „The Network is the Computer", in den 1990er-Jahren vor den Mythen der Netzwerk-Programmierung gewarnt:

1. Netzwerk fällt nie aus
2. Latenz = 0
3. Bandbreite = unendlich
4. Netzwerk ist sicher

Diese Mythen beziehungsweise Irrtümer von Bill Joy, manchen auch als Vater der C-Shell bekannt, wurden im Jahr 1994 von L. Peter Deutsch ergänzt:

5. Topologie ändert sich nicht
6. Es gibt einen Administrator
7. Transport-Kosten = 0

Später hat James Gosling noch hinzugefügt:

8. Netzwerk ist homogen.

Natürlich kann das Netzwerk ausfallen oder zu nicht akzeptablen Wartezeiten führen. Glücklicherweise gibt es dazu bereits einige vorgefertigte Bausteine, die den Umgang mit diesen Herausforderungen vereinfachen. In unserem Fall haben wir uns neben Spring als Basis-Framework für folgende Komponenten entschieden:

- **Keycloak (SSO)**
  Bei Keycloak handelt es sich um eine Single-sign-on-Lösung von Red Hat, die OAuth2 unterstützt und die Anmeldung und Authentifizierung eines Benutzers abnimmt. Die Absicherung der Kommunikation erfolgt dabei über Tokens, die mit jedem Request weitergereicht und auf Gültigkeit geprüft werden.
- **Eureka (Registry)**
  Eureka von Netflix erlaubt die Registrierung von Microservices. Zusammen mit Zuul als API-Gateway (siehe unten) ist es die einzige Adresse, die ein Microservice kennen muss, um auf andere Microservices zugreifen zu können.
- **Zuul (API-Gateway)**
  Zuul dient als API-Gateway und leitet eingehende Anfragen auf den entsprechenden Microservice weiter. Er ist ein wichtiger Baustein für die Skalierung von Microservices, da er Server-seitig Load Balancing mithilfe von anpassbaren Filter- und Routing-Regeln unterstützt.
- **Ribbon + Feign**
  Ribbon ist ein Client-seitiger Load Balancer, der seine Informationen von Eureka bezieht. Feign wiederum erleichtert die Deklaration von REST-Schnittstellen, die dann mit Ribbon kombiniert werden können.
- **Hystrix**
  Hystrix ist die Netflix-Implementierung des Circuit Breaker Pattern [3]. Wenn ein Microservice ausfällt, wird der Aufruf nicht mehr weitergeleitet und auf einen Timeout gewartet, sondern gleich der Fehler (oder ein Fallback) zurückgemeldet.
- **Hystrix-Dashboard/Turbine**
  Das Hystrix-Dashboard zusammen mit Turbine visualisiert die Anzahl von Anfragen und Fehlerfällen – sozusagen das EKG der Microservices.
- **Sleuth + Zipkin**
  Während Sleuth verteiltes Tracing über verschiedene Log-Dateien unterstützt, kann Zipkin diese visualisieren. Dies ist ein wichtiges Hilfsmittel für die Fehlersuche.
- **Spring Boot Actuator**
  Das Actuator-Modul bietet Endpunkte an [4], die über den Gesundheitszustand eines Microservice Auskunft geben.
- **Spring Boot Admin**
  Spring Boot Admin bietet die passende Oberfläche, um sich auf einen Blick eine schnelle Übersicht über die einzelnen Services zu verschaffen.

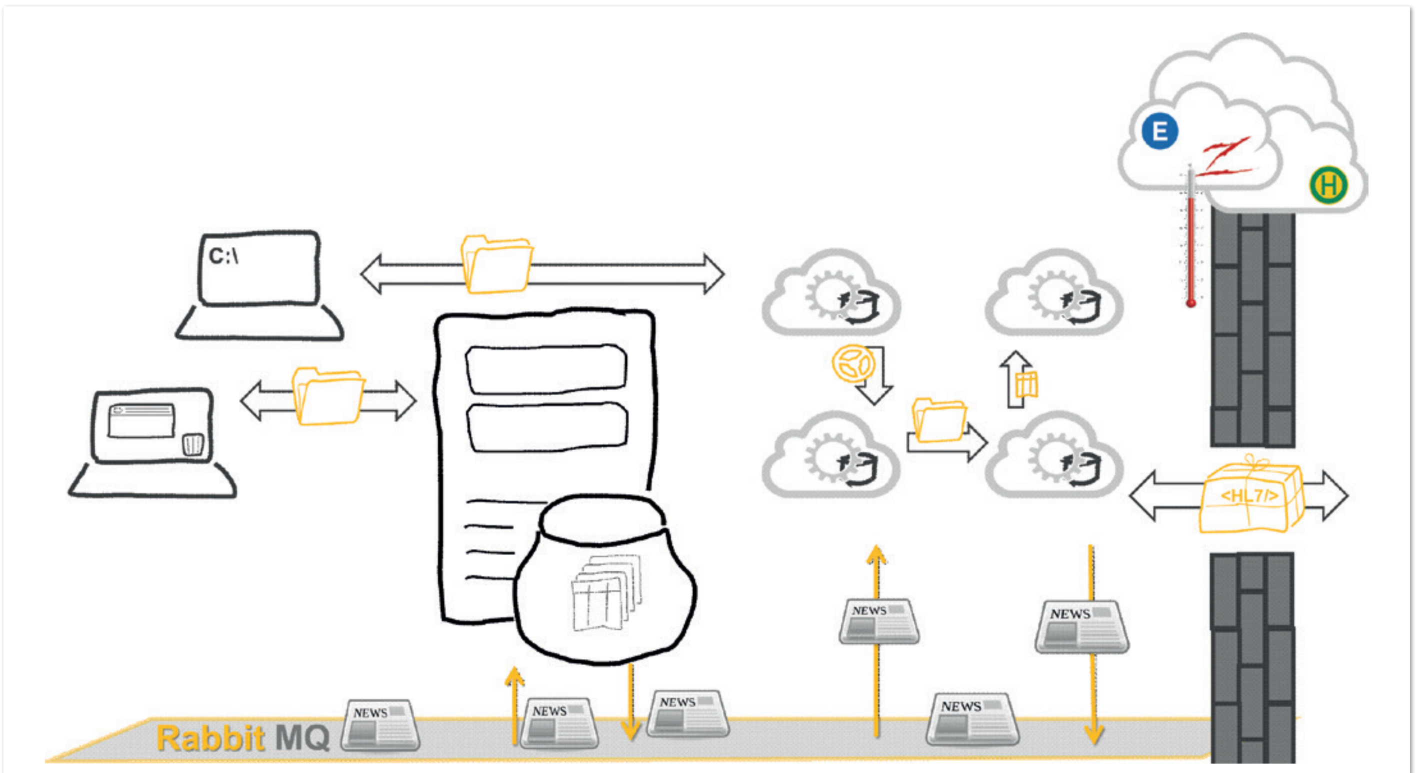

Abbildung 5: Ein Blick in die Zukunft

Dies sind nur einige Bausteine, auf deren Basis aktuell Microservices entwickelt werden, die dann nach und nach bestehende Anwendungen ergänzen oder ersetzen sollen. Zur Ausfallsicherheit werden wichtige zentrale Dienste wie Eureka oder Zuul geclustert sowie durch regelmäßige Last- und Stress-Tests die geforderten Antwortzeiten überprüft. Dies dient auch zur Abschätzung, welche Services mehrfach gestartet werden müssen und wie sich das System beim Ausfall von Services verhält.

Zurück zum Status quo: Während die alte (Nixdorf-)Welt bereits in der Vergangenheit abgelöst wurde (bis auf einige wenige antiquarische Sonderfälle), sind wir jetzt dabei, die mittelalte (Client/Server-)Welt mit ihren starren Strukturen und Zünften zu verlassen, überlieferte Prozesse zu überdenken und nach und nach einzelne Anwendungen auszusortieren oder in die Neuzeit zu überführen. Dies bedeutet in vielen Fällen eine Neuentwicklung von schlankeren Oberflächen auf Basis moderner Web-Technologien, über die die neuen Microservices angesprochen werden. Es bedeutet allerdings auch, dass die Entwickler sich weiterentwickeln müssen und dafür auch Schulungen eingeplant werden.

Spannend wird es während der Übergangsphase: Bis unsere Informix-Datenbank abgeschaltet werden kann, werden wir eine redundante Datenhaltung in beiden Welten haben. Dies ist die größte Herausforderung bei der Modernisierung der Anwendungslandschaft. Technisch wird das über Nachrichten auf Basis von RabbitMQ gelöst, die bei jeder Datenänderung versendet (per Trigger auf der Datenbank und von den Microservices selbst) und von der Gegenstelle (Microservice oder vorgeschalteter Sync-Service für die „alte" Datenbank) empfangen werden *(siehe Abbildung 5)*.

Wer mehr dazu wissen oder gar mitmachen will, darf mich gerne ansprechen. Nächste Gelegenheit dazu wird im September die BED-Con in Berlin sein oder nächstes Jahr wieder im Juli das Java Forum Stuttgart – vielleicht mit einem neuen Wetterbericht aus Cloud City: „Wolkig mit Aussicht auf Fleischbällchen" *[5]*.

Weitere Informationen

[1] *https://www.spruch-archiv.com/completelist/seite-2/action-vote/?query=Vergangenheit+Gegenwart+Zukunft&id=14783&vote=10&pos=15&key=13ljzUhECR&sid=4bac20aa1123325ec08780f21485aa1f#pos15*
[2] *https://de.wikipedia.org/wiki/Informix*
[3] *https://de.wikipedia.org/wiki/Sicherung_(Entwurfsmuster)*
[4] *https://docs.spring.io/spring-boot/docs/current/reference/html/production-ready-endpoints.html*
[5] *https://www.javaland.eu/formes/pubfiles/9974384/2018-nn-holly_cummins-cloudy_with_a_chance_of_meatballs__cloud_surprises_for_the_java_developer_-praesentation.pdf*

Oliver Böhm
ob@jugs.org

Oliver Böhm beschäftigt sich mit Java-Entwicklung unter Linux und Aspekt-Orientierter SW-Entwicklung. Neben seiner hauptberuflichen Tätigkeit als JEE-Architekt bei T-Systems ist er Buchautor, Projektleiter bei PatternTesting und Board-Mitglied der Java User Group Stuttgart.