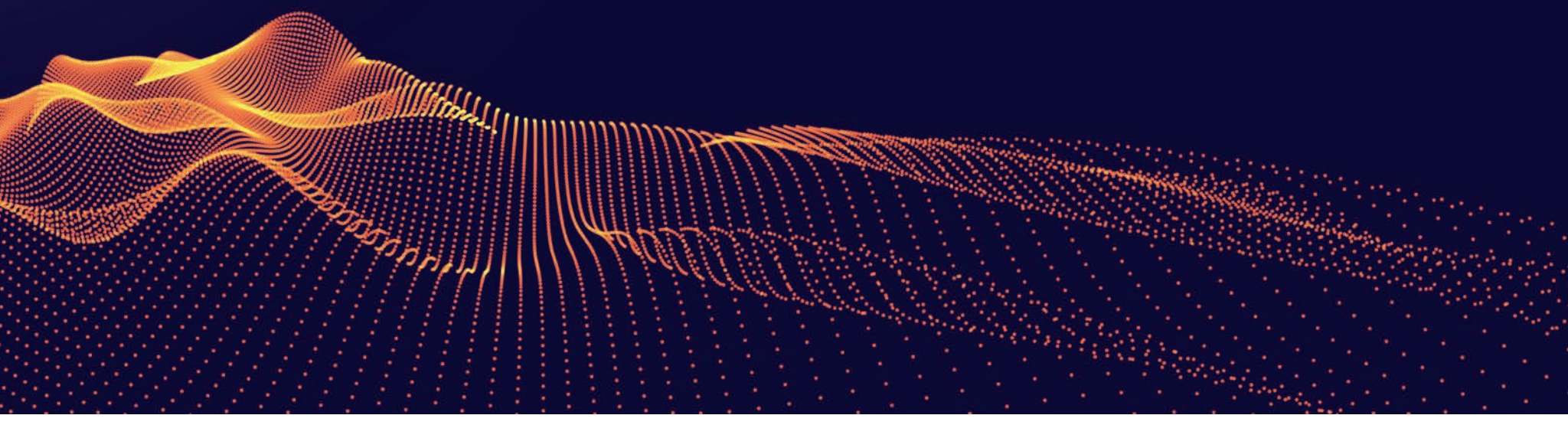

BIG DATA-PROJEKTE: PROBLEMLÖSER SERVICE MESH?

VON SOA ÜBER ESB ZUM SERVICE MESH.

Die meisten IT-Architekten gehen derzeit davon aus, dass Microservices die Antwort auf all die Probleme sind, die sie mit früheren Architekturen wie serviceorientierten Architekturen (SOA) und Enterprise Service Bus (ESB) hatten. Wenn man sich jedoch die aktuellen Microservices-Implementierungen in der Praxis anschaut, stellt man fest, dass auf der Ebene der Microservices häufig nichts anderes als die Funktionalität eines zentralen ESB implementiert wird. Gelöst werden also mehr oder weniger die gleichen grundlegenden Probleme, nur in diesem Fall mit Microservices in verschiedenen Dimensionen. Hier kommt das Konzept des ´Service Mesh´ ins Spiel, das wichtige und oft genutzte Funktionalitäten bündelt und für die einzelnen Microservices bereitstellt.

Microservices-Architekturen im Wandel – Entwicklung und Herausforderungen

Eine Microservices-Architektur ist ein verteiltes System, in dem unterschiedliche Bauteile über das Netzwerk miteinander sprechen. In einem verteilten System kann der Ausfall eines fremden Computers den eigenen Computer lahmlegen. Es sind große Anstrengungen notwendig, um das Risiko für ein solches Ausfall-Szenario im Vorfeld zu minimieren und damit die Vorteile einer Microservices-Architektur nutzbar zu machen. Die Lösung könnte eine technologische Strategie sein, mit der wir die Irrtürmer der verteilten Datenverarbeitung adressieren können. [1]

In der ersten Generation von Microservices-Architekturen wurden die notwendigen Muster zur Adressierung der Probleme in Frameworks in der jeweiligen Programmiersprache umgesetzt. Bekannt wurde der sogenannte „Netflix-Stack", eine Sammlung von Java Libraries, die Netflix in seiner Architektur einsetzt und Open Source stellte. Mittels der Tools Eureka, Zuul und Hystrix werden dabei die Themen Service Discovery (Eureka: Wie finden sich Services bei einer sich dauernd ändernden Netzwerktopologie?, Hystrix: Wie verhindere ich, dass der Ausfall eines Services durch das Netzwerk kaskadiert?) und Sicherheit adressiert.

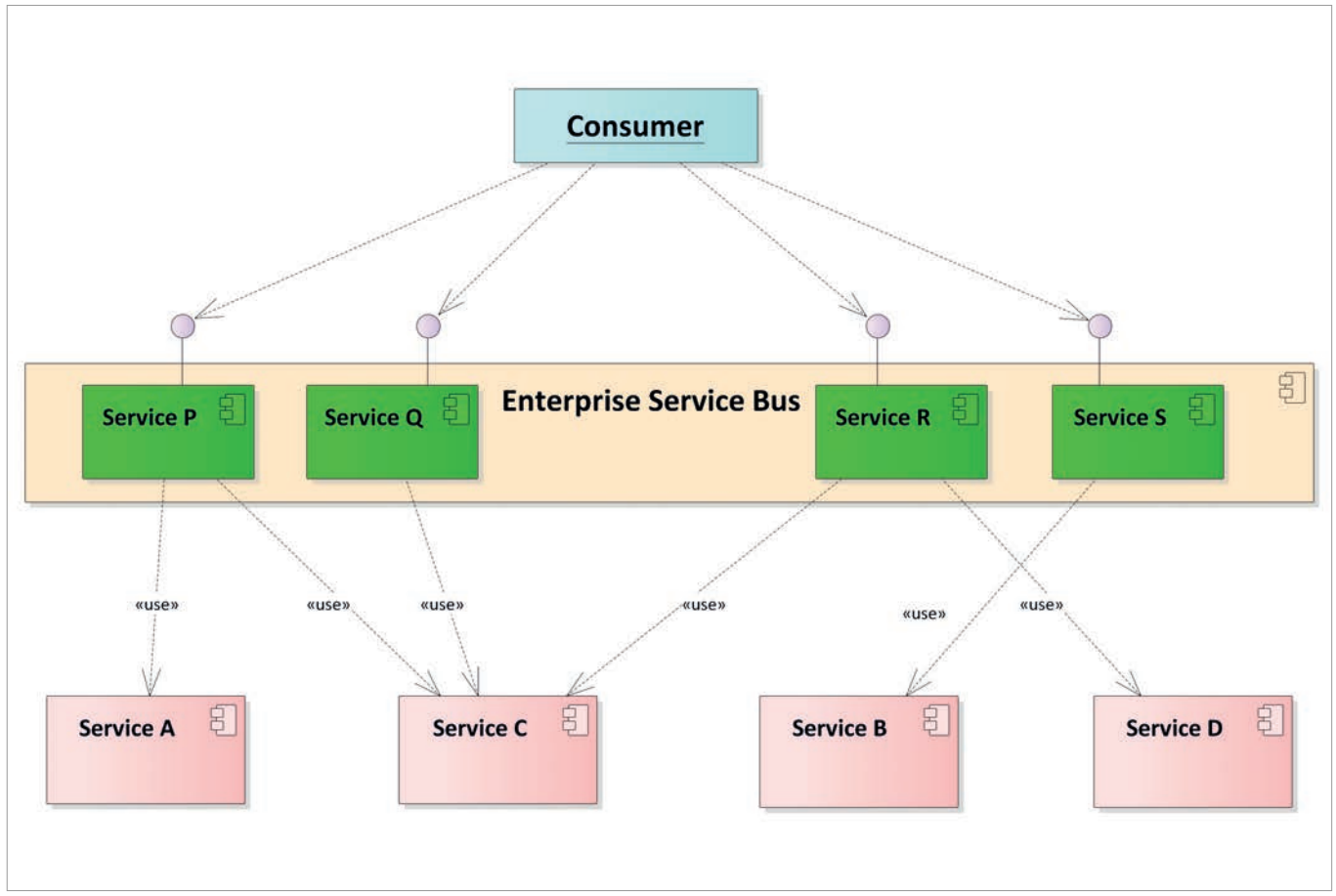

Bild 1: Enterprise Service Bus Architecture.

Microservices versus ESB

Die Themen Service Discovery und Sicherheit gelten auch als typische Aufgaben eines ESB. Für einen ESB gibt es keine eindeutige Definition, aber häufig wird er mit einer zentralen Komponente gleichgesetzt. Und darin liegt der wesentliche Unterschied. Microservices setzen auf Dezentralität und autonome Entscheidungen, ein ESB wird in Unternehmen häufig von einer zentralen Einheit verwaltet. Daher widerspricht der ESB einigen Architekturtreibern, welche die Wahl für eine Microservices-Architektur beeinflussen.

Sidecars und Service Mesh

Mit der nächsten Generation von Microservices-Architekturen änderte sich die Situation etwas. Das Ökosystem, insbesondere um Kubernetes herum, war reifer geworden, und man erkannte, dass man bestimmte Aspekte, die jedem Microservice an die Seite gestellt werden, als Dienste anbieten kann. Dieses Muster nennt sich Sidecar Pattern. Die Kommunikation zwischen den unterschiedlichen Services erfolgt dabei immer über die Sidecars. Diese kümmern sich um alle kommunikationsrelevanten Themen und entlasten so den Anwendungsentwickler bei der korrekten Behandlung dieser Themen und ermöglichen gleichzeitig eine bessere zentrale Durchsetzung von bestimmten Policys und Nachvollziehbarkeitsanforderungen. Die Sidecars und deren Steuerung als Gesamtheit bezeichnet man als „Service Mesh".

Ein Blick unter die Haube

Schauen wir uns Sidecars und Service Mesh einmal genauer an: Nehmen wir zum Beispiel ein Szenario, in dem wir mehrere nachgelagerte Dienste belastbar aufrufen und die Funktionalität als einen weiteren (zusammengesetzten) Dienst darstellen müssen. Wie in Bild 1 dargestellt, können wir mit einer ESB-Architektur die in ESB integrierten Funktionen leicht nutzen, um Funktionalitäten aufzubauen, die für die Inter-Service-Kommunikation nützlich sind, wie Leistungsausfall, Timeouts, Service-Erkennung und so weiter.

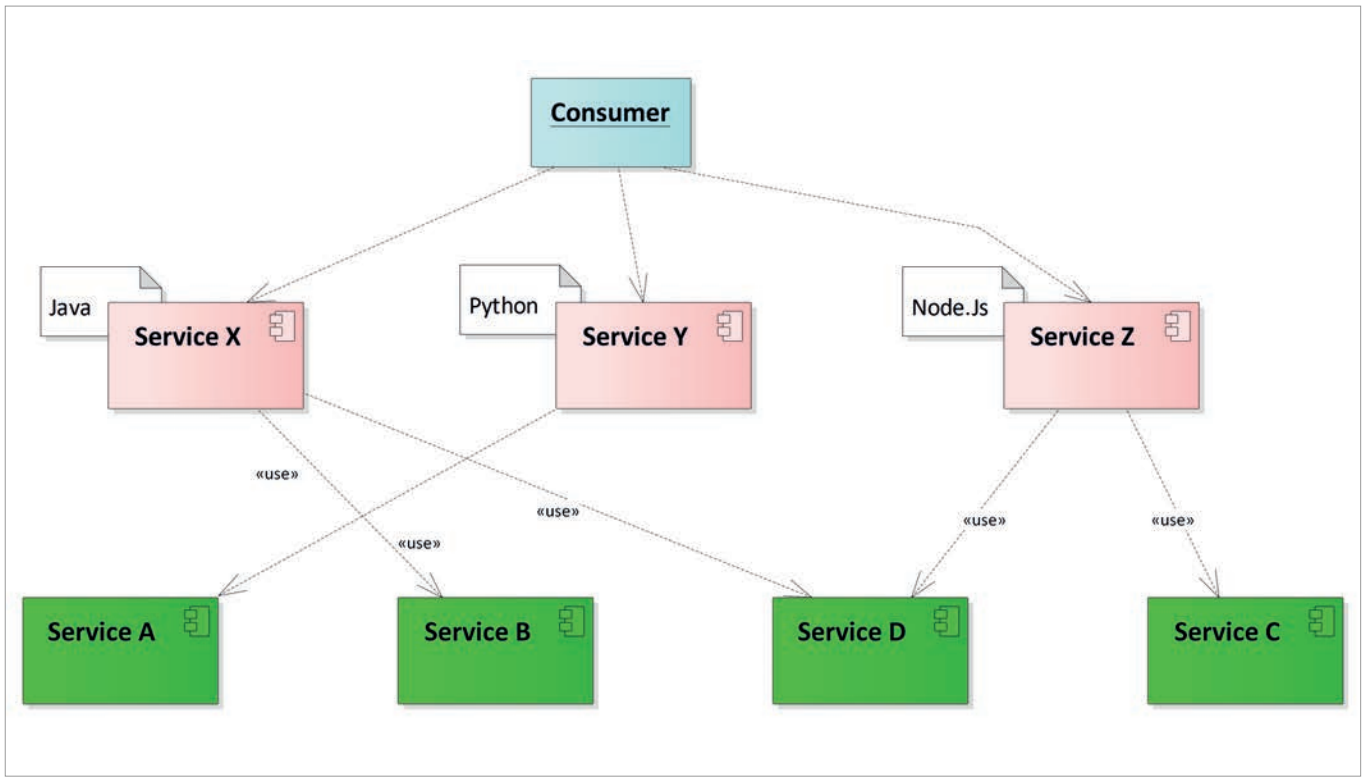

Bild 2: Microservices-Architecture.

Wenn wir das gleiche Szenario mit Microservices implementieren, dann haben wir es nicht mehr mit einer zentralisierte Integrations-/ESB-Schicht zu tun, sondern mit einer Reihe von zusammengesetzten und atomaren Microservices. Daher ist es wichtig, alle diese Funktionalitäten auf der Ebene der Microservices zu implementieren. Doch könnte die Kommunikation nicht einfach weiterhin über einen ESB laufen?

Dagegen spricht der Wunsch der Entwicklerteams nach Autonomie. Die Loslösung von einem zentralen Team und von den mit diesem verbundenen Abhängigkeiten und Bottlenecks führt unvermeidlich zu einem dezentralen Architekturansatz wie Microservices.

Daher umfasst ein bestimmter Microservice, der mit anderen Diensten kommuniziert, auch a) die eigentliche Geschäftslogik und b) die Netzwerkfunktionen, die sich um die Kommunikationsmechanismen zwischen den Diensten kümmern (Bild 3).

Babylonische Sprachverwirrung

Der Aufwand für die Implementierung eines Microservice, der die funktionalitätsbezogenen Service-zu-Service-Kommunikation enthält, ist für viele Architekten ein Alptraum. Anstatt sich auf die Geschäftslogik zu konzentrieren, verbringen sie viel Zeit damit, die Funktionen für die Inter-Service-Kommunikation aufzubauen. Noch aufwändiger wird es, wenn mehrere Technologien zum Aufbau von Microservices verwendet werden, zum Beispiel mehrere Programmiersprachen wie in Bild 2 dargestellt. Dann fällt der gleiche Aufwand in verschiedenen Sprachen an. Zwar gibt es für die einzelnen Programmiersprachen Bibliotheken, die diese Netzwerkfunktionalitäten bereitstellen, doch diese müssen aus Sicherheitsgründen regelmäßig aktualisiert werden, was einen hohen administrativen Aufwand bedeutet.

Protokollierung und Tracing

Eine wichtige Frage, die sich bei verteilten Systemen aus Entwicklersicht stellt, lautet: Wie lassen sich Service-Aufrufe im Fehlerfall debuggen, loggen oder verfolgen? Dies war und ist trotz einer zentralen Instanz wie einem ESB auch in SOA eine Herausforderung. Verteilte Protokollierung über Services hinweg ist auch heute nur mit Diensten wie Graylog [2] oder Splunk [3] möglich. Für verteiltes Tracing werden sehr oft APM-Tools wie AppDynamics [4] verwendet, da diese von Haus aus Traces aufzeichnen können.

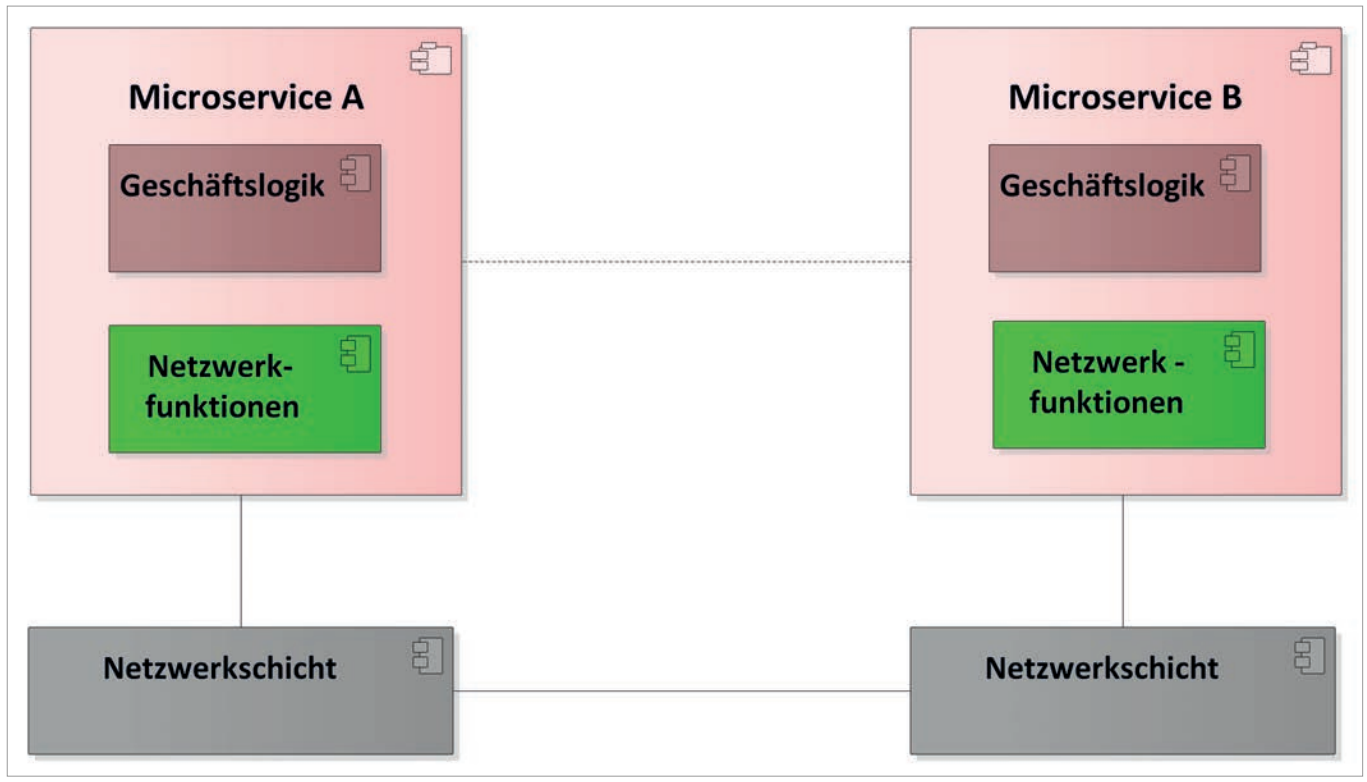

Bild 3: Microservices mit Service-zu-Service-Kommunikation.

Die komplexeste Herausforderung bei der Realisierung von Microservices-Architekturen besteht tatsächlich nicht im Aufbau der Dienste selbst, sondern in der Kommunikation zwischen den Diensten.

Inter-Service-Kommunikation

Die meisten Anforderungen an die Inter-Service-Kommunikation für die Microservices sind recht allgemein. So besteht meist der Wunsch, dass all diese Aufgaben in eine Komponente verlagert werden, damit sich die Microservices-Implementierungen auf die Geschäftslogik konzentrieren können.

In diesem Fall kommuniziert ein bestimmter Microservice nicht direkt mit den anderen Microservices. Vielmehr findet die gesamte Service-zu-Service-Kommunikation über eine Softwarekomponente statt: das Service Mesh.

Service Mesh – Hilfe für verteilte Systeme

Ein Service Mesh bietet integriert die Unterstützung für einige Netzwerkfunktionen wie Ausfallsicherheit oder Diensterkennung (Bild 4). So wird der größte Teil der Arbeit im Zusammenhang mit der Netzwerkkommunikation auf das Service Mesh übertragen und die Entwickler können sich stärker als bisher auf die Geschäftslogik konzentrieren.

Das Service Mesh ist sprachunabhängig: Da die Kommunikation der Microservices über den Service Mesh Proxy immer mithilfe von Standardprotokollen wie HTTP1.x/2.x oder gRPC abläuft, funktionieren die Microservices mit dem Service Mesh in jeder Technologie.

Wie sehen nun die Verantwortlichkeiten innerhalb der Service-Interaktionen aus:

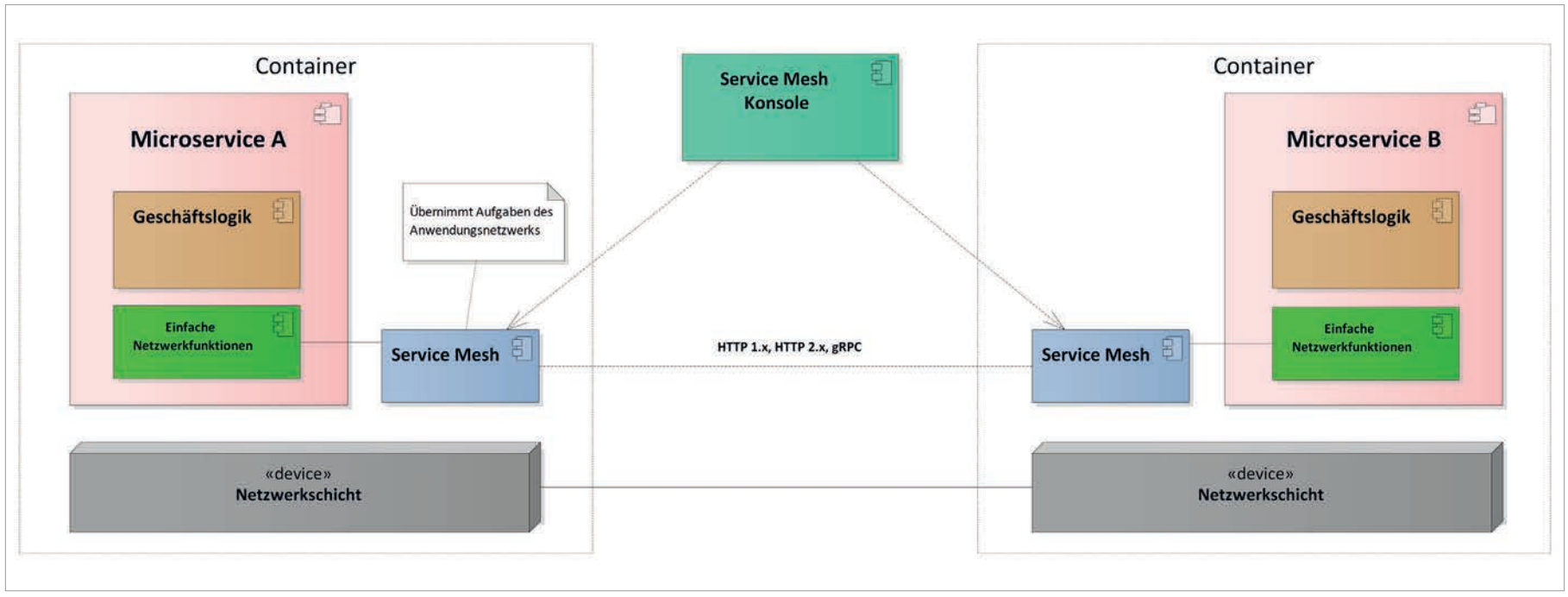

Bild 4: Service-zu-Service-Kommunikation mit dem Service Mesh.

1. Geschäftslogik

Die Service-Implementierung sollte die Realisierung der Geschäftsfunktionalitäten beinhalten. Dazu gehören die Logik für Geschäftsfunktionen, Berechnungen, Integration mit anderen Diensten/Systemen (einschließlich Legacy, proprietär und SaaS) sowie Service-Kompositionen, komplexe Routing-Logik, Mapping-Logik zwischen verschiedenen Nachrichtentypen und so weiter.

2. Einfache Netzwerkfunktionen

Obwohl wir die meisten Netzwerkfunktionen ins Service Mesh auslagern, muss ein bestimmter Dienst die grundlegenden High-Level-Netzwerkinteraktionen enthalten, um eine Verbindung mit dem Service Mesh herzustellen. Daher muss eine bestimmte Service-Implementierung eine bestimmte Netzwerkbibliothek verwenden.

In den meisten Fällen bettet das Microservices Development Framework die erforderlichen Netzwerkbibliotheken ein, die für diese Funktionen verwendet werden sollen. Anders als in der ESB-Welt verwendet man hier eine sehr einfache Abstraktion, um auf Services zuzugreifen.

3. Aufgaben des Anwendungsnetzwerks

Es gibt Anwendungsfunktionalitäten, die eng mit dem Netzwerk verbunden sind, wie Leistungsausfall, Timeouts, Serviceerkennung und so weiter. Diese sind explizit von der Service-Code/Business-Logik getrennt und werden vom Service Mesh übernommen, das diese Funktionalitäten sofort nach der Installation bereitstellt.
Microservice-Implementierungen haben die Komplexität der Netzwerkfunktionen, die von einer zentralen ESB-Schicht angeboten werden, zu Beginn häufig noch unterschätzt. Sie implementierten diese Funktionalitäten auf jeder Mikroservice-Ebene von Grund auf neu. Mittlerweile hat man erkannt, dass gemeinsam genutzte Funktionalitäten hier ähnlich wichtig sind, wie bei einem verteilten Netz.

4. Steuerebene

Alle Service Mesh Proxys werden zentral über eine Konsole verwaltet. Das ist sehr sinnvoll, wenn es darum geht, Service-Mesh-Funktionen wie Zugriffskontrolle, Beobachtungsfähigkeit oder Serviceerkennung zu unterstützen.

DER AUFWAND FÜR DIE IMPLEMENTIERUNG EINES MICROSERVICE, DER DIE FUNKTIONALITÄTSBEZOGENEN SERVICE-ZU-SERVICE-KOMMUNIKATION ENTHÄLT, IST FÜR VIELE ARCHITEKTEN EIN ALPTRAUM.

Klaus Kramer
Senior Consultant, Opitz Consulting
www.opitz-consulting.com

Funktionalitäten eines Service Mesh

Wie wir bereits gesehen haben, bietet das Service Mesh eine Reihe von Anwendungsnetzwerkfunktionen, während einige (primitive) Netzwerkfunktionen noch auf Microservices-Ebene implementiert sind. Die Funktionalitäten eines Service Mesh sind nicht fest definiert. Häufig bietet es diese Funktionen:

- Ausfallsicherheit für die dienstübergreifende Kommunikation: Leistungsunterbrechung, Wiederholungen und Timeouts, Fehlerinjektion, Fehlerbehandlung, Lastausgleich und Ausfallsicherung
- Service-Erkennung: Erkennung von Service-Endpunkten durch eine dedizierte Service-Registrierung
- Beobachtbarkeit: Metriken, Überwachung, verteilte Protokollierung, verteiltes Tracing
- Sicherheit: Transport Level Security (TLS)
- Zugangskontrolle: Listen-basierte Zugriffskontrolle
- Bereitstellung: Native Unterstützung für Container. Docker und Kubernetes
- Kommunikationsprotokolle zwischen den Diensten: HTTP1.x, HTTP2, gRPC

Vorteile

Die allgemeinen Netzwerkfunktionen werden außerhalb des Microservice-Codes implementiert und sind so wiederverwendbar. Ein Service Mesh behebt die meisten Probleme in der Microservices-Architektur, die wir früher mit Ad-hoc-Lösungen hatten wie Verteilte Verfolgung, Protokollierung, Sicherheit, Zutrittskontrolle und so weiter. Zusätzlich bekommen wir mehr Freiheit bei der Auswahl einer Implementierungssprache für die Microservices, das heißt, wir müssen uns keine Gedanken mehr darüber machen, ob eine bestimmte Sprache Bibliotheken besitzt, um Funktionen für die Netzwerkkommunikation zu erstellen oder die Funktionen von Grund auf neu zu implementieren.

Nachteile

Da wir mehr Laufzeitkomponenten in unserem System benötigen, wird es insgesamt komplexer. Des Weiteren fügen wir eine weitere Komponente in unsere Netzkommunikation hinzu: Jeder Service-Aufruf durchläuft nun zusätzlichen einen Service Mesh.

Der Service Mesh adressiert nur eine Teilmenge von Kommunikationsproblemen zwischen den Diensten. Viele umfängliche Probleme wie komplexes Routing, Transformation/Typ-Mapping, Integration mit anderen Diensten und Systemen, müssen weiterhin in der Geschäftslogik der Microservices gelöst werden.

Service-Mesh-Implementierungen

Die bekanntesten Service-Mesh-Implementierungen sind Envoy [5] oder das darauf basierende Istio-Framework [6] sowie Linkerd [7] und consul-connect []. Bei diesen handelt es sich um Open-Source-Projekte, die sich mit dem Thema Service Mesh beschäftigen. Alle Service-Mesh-Technologien sind relativ neu und werden aber bereits für voll produktiv erklärt.

Fazit

Zusammenfassend lässt sich sagen, dass die Service-Mesh-Technologie einige der wichtigsten Herausforderungen bei der Realisierung von Microservices-Architekturen angeht. Architekten profitieren damit von einer größeren Freiheit bei der Auswahl von Technologien zur Implementierung von Microservices. Außerdem benötigen sie weniger Zeit für Netzwerkfunktionen zwischen den Services und können sich mehr auf die Geschäftslogik konzentrieren. Allerdings löst das Service Mesh keine Probleme, die mit der Geschäftslogik oder mit der Service-Integration oder -Komposition verbunden sind.

Klaus Kramer

Quellen

[1] https://de.wikipedia.org/wiki/Fallacies_of_Distributed_Computing
[2] https://www.graylog.org/overview
[3] https://www.splunk.com/en_us/solutions/solution-areas/log-management.html
[4] https://www.appdynamics.de/
[5] https://www.envoyproxy.io/
[6] https://istio.io/
[7] https://linkerd.io/
[8] https://www.consul.io/docs/connect/index.html