Smart Commerce

Monolithen vs. Microservices

Ein Leitfaden zu Architekturvorteilen, Migrationsstrategien, Tools und Technologien

Inhalt.

Gerd Valdeig

1. State of the Art eCommerce Architekturen

Im eCommerce dominieren monolithische Middleware-Lösungen den Markt. So hilft bspw. die SAP Commerce Cloud Unternehmen, Kosten zu senken, Zeit zu sparen und Komplexität zu reduzieren, um ein hervorragendes Kundenerlebnis zu generieren. Doch bei monolithischen Anwendungen nimmt der Entwicklungsumfang mit der Zeit zu. Infolgedessen wächst nicht nur die Codebasis, sondern auch die Komplexität, wodurch sich Prozesse mühsamer gestalten. Eine agile Entwicklung und schnelle Bereitstellung werden so zu echten Herausforderungen.

Um diese zu lösen, wandeln einige Unternehmen ihre monolithische Middleware in individuelle Microservices um.

Die Microservice-Architektur lässt sich einfach testen, warten und ist eine lose gekoppelte Lösung, bei der jeder Service autonom bereitgestellt werden kann. Sie besitzt zudem den Vorteil, dass sie sich an den Business Capabilities eines Unternehmens orientiert. So kann der Entwicklungsaufwand auf mehrere Teams verteilt werden, die jeweils für einen oder mehrere Services verantwortlich sind. Ein jedes Team kann unabhängig von den anderen Teams entwickeln, testen und skalieren.

> **„Riesige Firmen wie Amazon und Netflix basieren nicht umsonst auf Microservices. Sie bieten den Unternehmen eine große Flexibilität. Weil sie so viele Nutzer haben, dürfen deren Systeme nicht einfach so ausfallen. Und weil es ihnen dank der Microservice Architektur gelingt, die Anzahl und Dauer von Ausfällen zu reduzieren, gewinnen sie auch eine Menge Geld. Denn mit dieser hohen Verfügbarkeit und Zuverlässigkeit kommt auch eine starke Kundenbindung.“**
>
> Gerd Valdeig, Senior Consultant Smart Commerce

Doch Unternehmen dürfen sich nicht der Illusion hingeben, dass Microservices alle ihre Probleme mit einem Mal lösen oder einfacher funktionieren als Monolithen. In vielen Fällen gilt es abzuwegen, was im individuellen Fall die beste Lösung ist. Das kann auch ein Zusammenspiel beider Architekturen sein.

Welche Vorteile bieten also die jeweiligen Architekturen Unternehmen? Welche Migrationsstrategie ist die effektivste für Microservices, in welchen Fällen sollte man sie überhaupt einsetzen und welche Technologien und Tools eignen sich für deren Aufbau? Wir haben unsere Experten zum Thema befragt und ihre Einschätzungen für Sie zusammengefasst.

1.1 Monolithische Struktur

Die monolithische Architektur galt jahrelang als das traditionelle Softwaredesign. Denn auch wenn beide Ansätze etwa gleichalt sind, überwogen beim Microservices-Ansatz für lange Zeit die Nachteile die Vorteile. Von potenziellen Veränderungen unabhängig wird ein Monolith häufig exhaustiv bereitgestellt. In Monolithen wird alles in einer Codebasis konstituiert. In den meisten Fällen setzen sie sich aus drei Bestandteilen zusammen:

- Präsentationsschicht,
- Geschäftslogik-Schicht und
- Datenbank

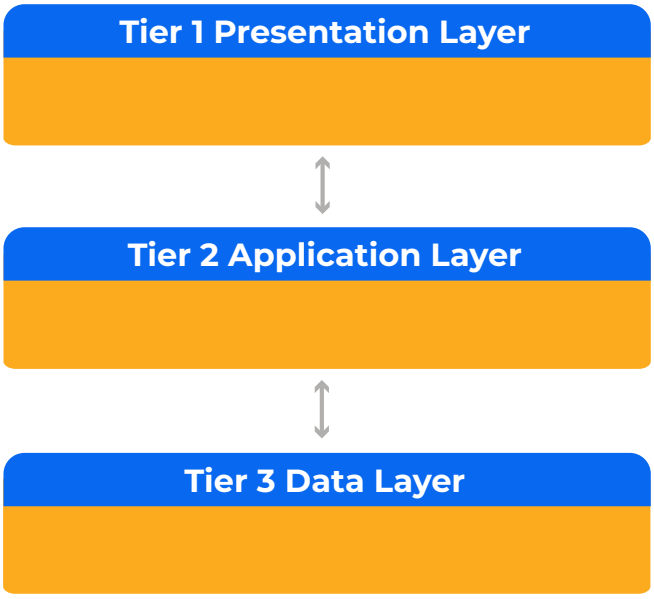


Abb.: Einer von mehreren Ansätzen: Die traditionelle dreischichtige monolithische Architektur

1.2 Microservices

Microservices ermöglichen als Architekturmuster den Aufbau einer Gruppe von lose gekoppelten Services. Diese sind logisch und physisch getrennt und jeder einzelne Service hält das Single-Responsibility-Prinzip ein. Das wiederum ermöglicht parallele Entwicklung, Testing und unabhängige sowie kontinuierliche Bereitstellung.

Jeder einzelne Microservice lässt sich individuell skalieren. Während monolithische Anwendungen eine einzige logische Datenbank für persistente Daten verwenden, verfügen Microservice-Architekturen über ein dezentrales Datenmanagement. Die Grenzen einzelner Microservices werden von individuellen Bounded Contexts definiert. Deshalb sollten letztere jeweils mit einer einzelnen, klaren Business Capability übereinstimmen.

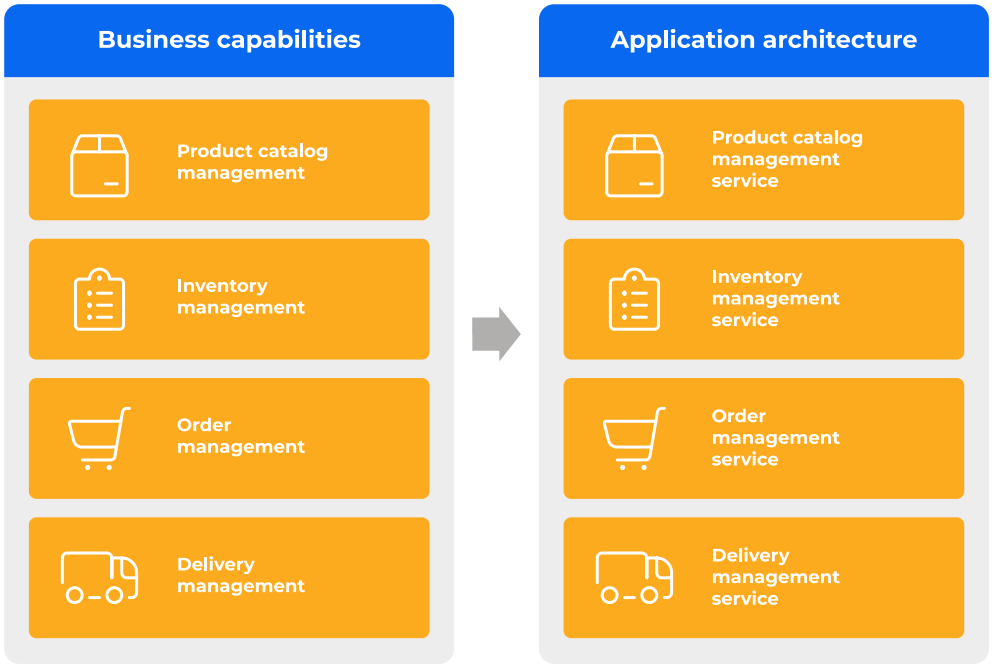


Abb.: Die Microservice-Architektur würde über Services verfügen, die den einzelnen Business Capabilities entsprechen

> **„Any organization that designs a system (defined more broadly here than just information systems) will inevitably produce a design whose structure is a copy of the organization's communication structure."**
>
> Gesetz von Conway

2. Migrationsstrategie

Welche Migrationsstrategien eignen sich, um Monolithen in Microservices zu segmentieren?

Diese Frage bezieht nicht nur die Codebasis ein, sondern betrifft auch die (Aufteilung der) Datenhaltung. Denn die dezentrale Datenhaltung ist eines der Hauptmerkmale der Microservice-Architektur und jeder einzelne Service sollte über eine eigene Datenhaltung verfügen. Es sollte allerdings klar sein, dass eine Migration, die einen Monolithen gänzlich mittels Microservices nachbaut, einen riesiegen Ressourcenaufwand erfordert. Deshalb setzen Unternehmen, die Microservices einsetzen wollen, in der Regel auf bereits vorhandene Ansätze wie die commercetools Technologie oder lösen nur einzelne Funktionen raus.

2.1 Aufspalten der Monolithen

Für den Erfolg des Prozesses ist entscheidend, dass die neue extrahierte Funktion genauso funktioniert wie jene im Monolithen.

Es ist deshalb wichtig, die Entwicklung im Monolithen so weit wie möglich einzuschränken, bevor neue Services extrahiert werden können. Erweitert man einen großen, komplexen Monolithen und hat während der Implementierung neuer Funktionen die Wahl, mehr Code hinzuzufügen, sollte man besser die Alternative wählen und sie als separate Services implementieren.

2.1.1 Auswahl der zu extrahierenden Funktionen

Extrahierung nach Business Capabilities

Durch die Extraktion von Business Capabilities entsteht eine stabile Microservice-Architektur, in der die Services lose miteinander gekoppelt sind. Um dies zu erreichen, gilt es einige Bedingungen zu erfüllen:

- die Entwicklungsteams sollten funktionsübergreifend agieren,
- sie sollten ein tiefes Verständnis für das gesamte Geschäft haben und
- sich auf die Bereitstellung von Business Value statt auf technische Funktionen konzentrieren.

Extraktion von Services gemäß den Subdomänen des Domain-Driven Designs

Eine alternative Möglichkeit besteht darin, Services gemäß den Subdomänen des Domain-Driven Designs zu extrahieren. Dabei wird das Domänenmodell in einzelne Subdomänen aufgeteilt. Dieser Ansatz eignet sich besonders dann, wenn die zugrundeliegenden Services klar abgegrenzte Grenzen haben. In der Regel ist kein aufwendiges Umschreiben des vorhandenen Codes erforderlich.

Vom Monolithen leicht abkoppelbare Services, die an vielen Clients keine Änderungen benötigen:

Nach dieser Strategie empfiehlt sich eine Hierarchisierung nach Änderungsnotwendigkeit. Zunächst sollten die einfachen Edge-Services entkoppelt werden. Anschließend werden andere Services, die tiefer im Monolithen verankert sind, extrahiert, damit die Entwickler Erfahrung im Umgang damit sammeln können und erste erfolgreiche Veröffentlichungen bereits früh erfolgen.

Einige Fragen, die dabei helfen, zu entscheiden, mit welchem Service man beginnen sollte:

- Welcher Service könnte die Entwicklung vorantreiben, wenn er extrahiert wird?
- Welcher Service wirkt sich am meisten auf das Problem der Leistung, Skalierung und Zuverlässigkeit aus?
- Welcher Service verringert den Extraktionsaufwand für andere Services?

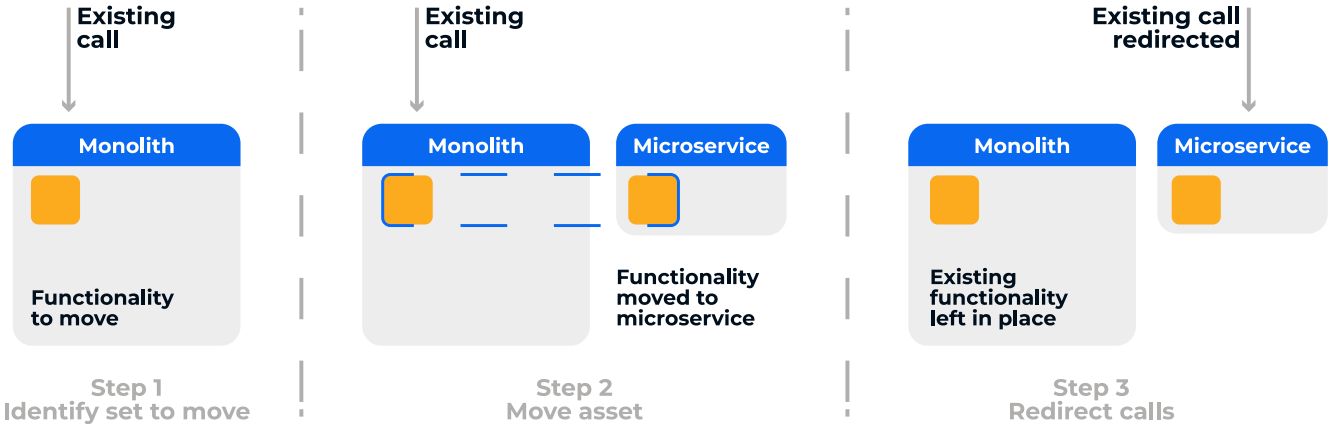


Abb.: Ein Überblick über die Strangler-Fig-Anwendung

2.1.2 Das Strangler Fig Application Pattern

Gegenüber der vollständigen Substitution eines Systems findet durch die Stranger-Fig-Anwendung eine inkrementelle Migration bei gleichzeitiger Nutzung des bestehenden Monolithen zur Risikominderung und schnelleren Auslieferung von Funktionalitäten statt.

Die Umsetzung des Musters erfolgt in drei Schritten:

1. Zunächst müssen die Funktionalitäten, die migriert werden sollen, im bestehenden Monolithen identifiziert werden. Dabei sollten die in 2.1.1 erläuterten Kriterien zur Auswahl vorrangiger Teile des Systems herangezogen werden.
2. Anschließend müssen diese Funktionalitäten im neuen Microservice implementiert werden.
3. Sobald die neue Funktionalität einsatzbereit ist, sollten Aufrufe vom Monolithen auf den neuen Microservice umgeleitet werden.

2.1.3 Parallel Run Pattern

Zusätzlich wird ein parallellaufendes Muster angewandt, um Leistungsvergleiche zwischen der neuen und der monolithischen Funktionalität ziehen zu können, außerdem ermöglicht dies die Reaktionszeit sowie Fehlerquoten zu ermitteln. Das parallele Muster erkennt nur eine Quelle als Wahrheit an, auch wenn es beide Funktionalitäten aufruft und die Ergebnisse gegenüberstellt. Bis die neue Implementierung ihre Zuverlässigkeit beweist, wird dies die monolithische sein.

2.2 Datenbankmigration

Während Unternehmen oft eine einzelne Datenbank für eine Reihe von Anwendungen bevorzugen, präferieren monolithische Anwendungen eine einzige logische Datenbank für persistente Daten. Die Entscheidungen hängen oft mit den kommerziellen Lizenzmodellen der Anbieter zusammen. Microservices stellen einen Gegensatz dazu dar, weil hier idealerweise jeder Dienst seine eigene Datenhaltung verwaltet – entweder verschiedene Instanzen derselben Datenbanktechnologie oder sogar komplett verschiedene Datenbanksysteme.

Es gibt verschiedene Muster und Methoden, um während der verschiedenen Migrationsstufen mit der Datenbank umzugehen.

Shared Database Pattern

Mehrere Microservices oder Artefakte können bei gemeinsam genutzten Datenbankmustern dieselbe Datenbank als „Quelle der Wahrheit" ausschöpfen. Der geteilte Zugriff mehrerer Services auf eine einzige Datenbank erschwert die Entscheidungen, was kollektiv genutzt wird oder nicht und was verborgen bleiben oder zugänglich gemacht werden soll. Eine gemeinsam genutzte Datenbank ist hingegen schnell bei der Datenmigration und zeigt eine starke Konsistenz.

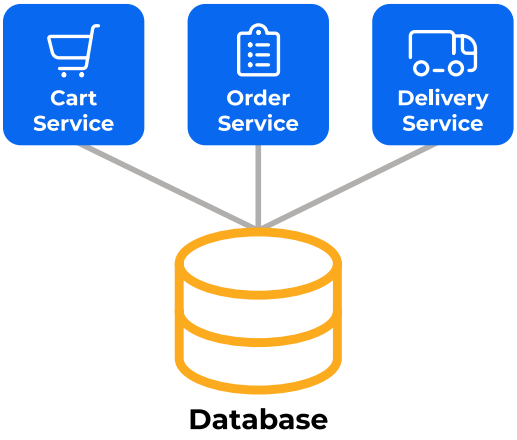


Abb.: Shared Database

Database View

Einerseits kann eine Datenbank-View als Resultat für eine gespeicherte Abfrage interpretiert werden, anderseits kann man sie als logische Darstellung einer oder mehrerer Tabellen begreifen.

Dadurch ist die Voraussetzung geschaffen, das Modell auf eine Weise abzubilden, die für ein bestimmtes Artefakt und einen bestimmten Anwendungsfall im Besonderen geeignet ist.

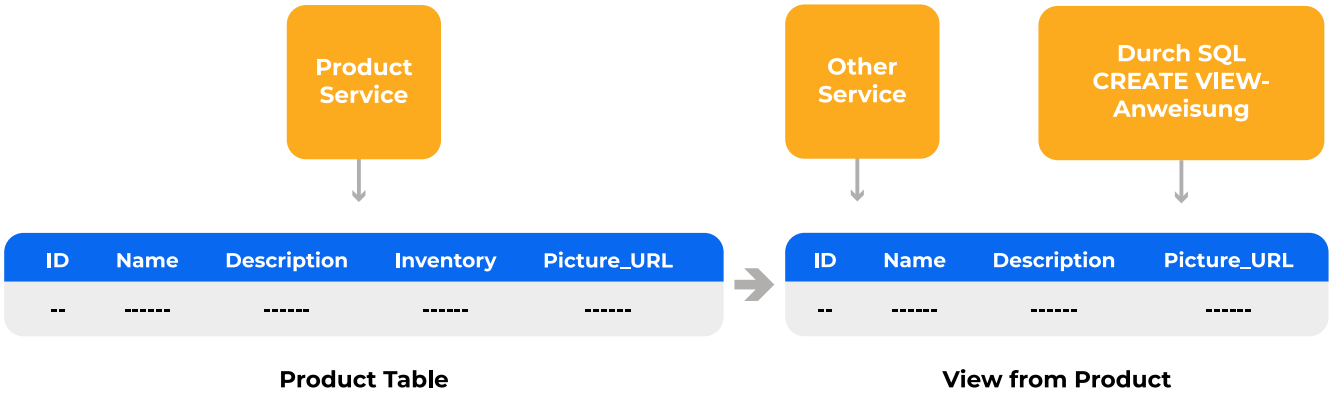


Abb.: ein Beispiel für eine View-Datenbank

Die Implementierung eines Datenbank-Views ist einfach und zeichnet sich durch hohe Konsistenz aus. Vorteil von Datenbank-Views ist die Möglichkeit, Informationen nur begrenzt aus der zugrunde liegenden Quelle zu projizieren. Dadurch kann kontrolliert werden, welche Informationen geteilt und welche versteckt werden sollen.

Database Wrapping Service

Mit dem Database Wrapping Service versteckt man die Database hinter einem Service, der als Ummantelung fungiert und Datenbankabhängigkeiten in Serviceabhängigkeiten umwandelt. Dies bietet Kontrolle darüber, was freigegeben wird und was nicht. Der Vorteil gegenüber dem Datenbank-View ist die Möglichkeiten, Code in den Wrapping-Service zu schreiben, um eine wesentlich bessere Projektion zu präsentieren sowie API-Aufrufe zu überschreiben.

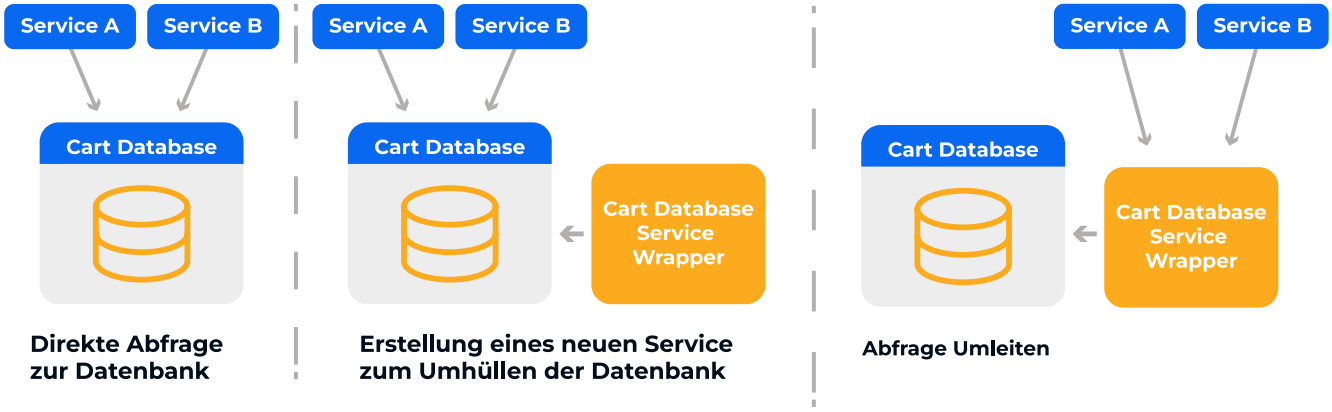


Abb.: Schritte zur Umhüllung einer Datenbank mit einem Wrapper-Service

Split Table

Die Split Table Methode ordnet die vertikale Anordnung in Tabellenspalten in eine oder mehrere Tabellen um.

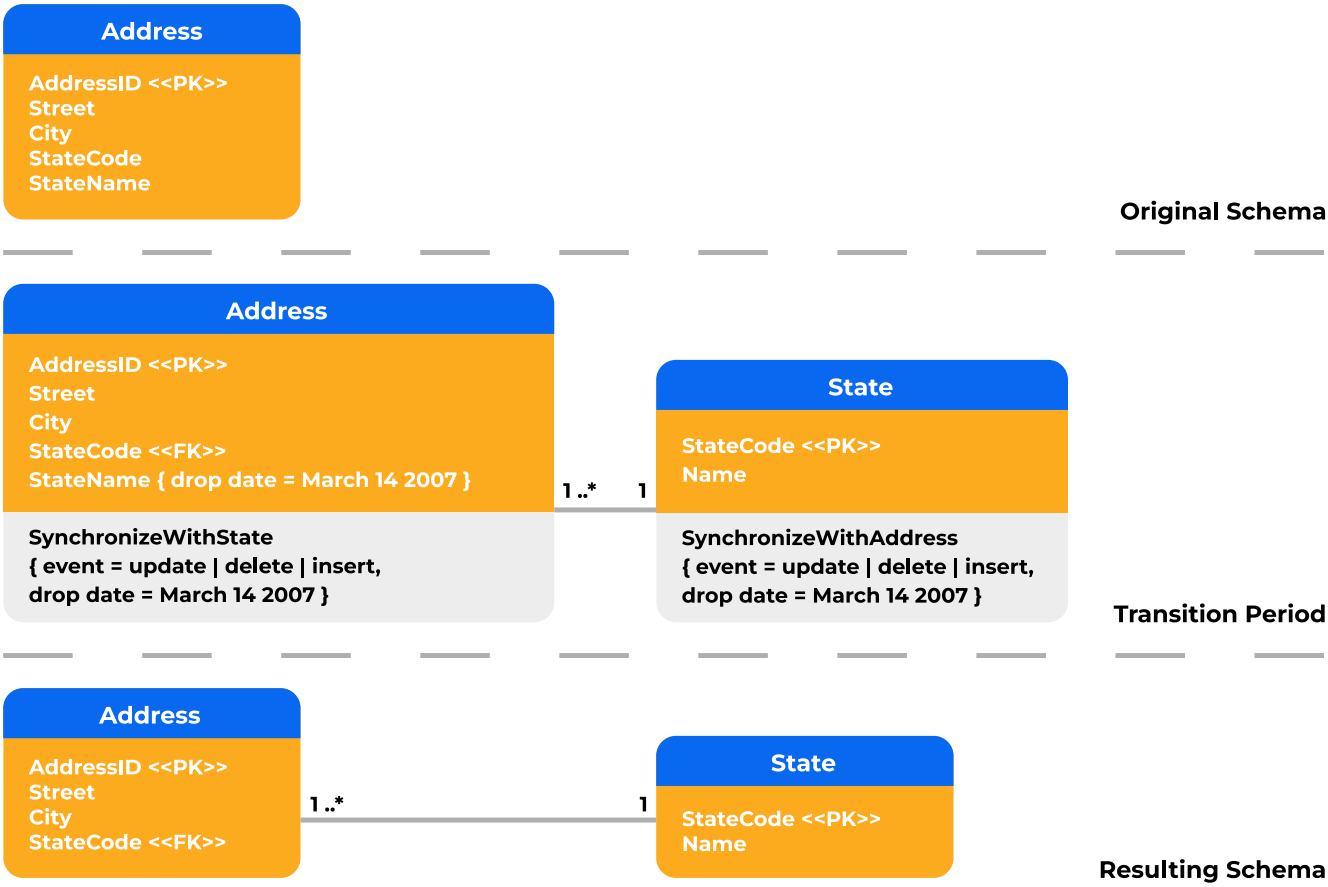


Abb.: Schritte zur Aufteilung der Adresstabelle

3. Tools und Technologien

Für den Aufbau von Microservices können verschiedene Technologien und Betriebsumgebungen eingesetzt werden. Die folgenden Tools und Technologien stellen eine Übersicht dar und haben keinen Anspruch an Vollständigkeit.

Spring Boot

Spring Boot ist der de facto Standard für Java™-Microservices. Es ermöglicht Microservices, klein anzufangen und schnell zu iterieren. Das eingebettete Servermodell von Spring Boot bietet den Vorteil, dass kein Server in der Bereitstellungsumgebung vorinstalliert werden muss. Der standardmäßig eingebettete Server ist Tomcat.

Docker

Docker ist eine offene Plattform für die Entwicklung, Bereitstellung und Ausführung von Anwendungen. Es ermöglicht, Anwendungen von der Infrastruktur zu trennen, sodass die Softwarebereitstellung beschleunigt wird. Zudem bietet es Container, die zur Verpackung und Ausführung einer Anwendung in einer losen, isolierten Umgebung befähigen. Docker vermag, Microservices zu containerisieren und die Bereitstellung und Verwaltung dieser Microservices zu vereinfachen. Die Containerisierung hält für die einzelnen Microservices ihre eigenen isolierten Umgebungen bereit und macht sie unabhängig voneinander einsetz- und skalierbar.

REST API

Clients verwenden Anwendungsprogrammierschnittstellen (APIs) zur Kommunikation mit Webservices. REST APIs sind Web-APIs, die dem Architekturstil ‚Representational State Transfer' entsprechen, angewandt auf einen WebService machen sie diesen RESTful. APIs müssen über eine klar definierte Semantik und Versionsschemata verfügen, damit Aktualisierungen andere Services nicht in ihrer Funktion behindern.

REST bietet die Möglichkeit, Domänenmodelle durch Ressourcen auszudrücken. Auf der Grundlage von HTTP-Verben definiert es eine Schnittstelle, verfügt über eine eindeutige Semantik in Bezug auf Idempotenz, Seiteneffekte sowie Antwortcodes und arrangiert eine Stateless-Kommunikation, was die Skalierbarkeit verbessert

Keycloak

Keycloak ist eine Open Source Identity und Access Management Solution für moderne Anwendungen und Services. Die Besonderheit beruht auf dem Single-Sign-On, Single-Sign-Out. Die Benutzenden authentifizieren sich mit Keycloak und nicht über einzelne Anwendungen, was den klaren Vorzug hat, dass diese Anwendungen sich nicht mit Anmeldeformularen, der Authentifizierung der User oder der Speicherung von Benutzerdaten befassen müssen. Einmal bei Keycloak eingeloggt, müssen sich die Nutzer nicht erneut einloggen, um auf eine andere Anwendung zuzugreifen. Keycloak bietet auch andere Features wie Benutzer-Registrierung, Social Login und Zwei-Faktor-Authentisierung.

Scientist

Scientist ist ein Open-source Framework, das von Github entwickelt wurde und zum Testen von Produktionsdaten und Verhalten verwendet wird. Mit Scientist lässt sich die Korrektheit einer neu implementierten Funktionalität sicherstellen und Legacy-Code mit der neu implementierten Funktionalität vergleichen. Ziel des Einsatzes ist, eine leichtgewichtige Abstraktion, ein Experiment, um den zu ersetzenden Quellcode herum zu erstellen.

Cronjob Service

Die Cronjob-Funktionalität in SAP Commerce Cloud wird für die kontinuierliche und wiederholte Ausführung von Aufgaben zu bestimmten Zeitpunkten verwendet. Zentraler Faktor bei der Anwendung von Cronjobs ist der Start eines langen oder periodischen Hintergrundprozesses mit der Option, jeden Lauf zu protokollieren und das Ergebnis leicht zu überprüfen.

4. Vor- und Nachteile der Architekturen

4.1 Bietet der Microservice Ansatz die gewünschten Vorteile gegenüber der bisherigen Architektur?

Basierend auf Tests und unseren eigenen Projekten zeigen Microservices einige Vorteile.

1. Bereitstellung

Microservices versetzen uns in die Lage, schnell neue Werte zu liefern, weil sie sofort nach der Extraktion verwendet werden können. Die vollständige Extraktion der am meisten genutzten Services (bspw. Produkt- und Warenkorbservices) trägt dazu bei, den Monolithen zu entlasten. Wenn ein Microservices ausfällt, hat dies keinen Einfluss auf die anderen. Kleine Services sind einfacher zu installieren und aufgrund ihrer Autonomie weniger anfällig für Systemausfälle, wenn sie fehlschlagen

> **„Mit Microservice Architekturen ist es möglich, automatisch auf wechselnde Arbeitslasten zu reagieren. Stark ausgelastete Microservices können dynamisch auf Cloud-Infrastrukturen repliziert werden.“**
>
> Ingo Körber, COO

2. Skalierung

Wenn Entwickler-Teams Services aus dem Monolithen herausschleusen und umgestalten, durchdringen sie die Domäne, an der sie arbeiten, substantiell und erwerben bessere Fähigkeiten im Hinblick auf den Migrationsprozess. Dies ermöglicht eine agile Entwicklung und verbessert die Qualität ihrer Arbeit. Gemäß dem IKEA-Effekt legen Menschen größeren Wert auf Produkte, die sie selbst erstellt haben.

4.2 Herausforderungen bei der Zerlegung Monolithischer eCommerce Lösungen in Microservices

…sind u. a. die Handhabung und Durchdringung von Monolithen angesichts ihrer großen Komplexität und die zentrale Entscheidung darüber, welche Services oder Funktionalitäten extrahiert werden sollen. Auch ist nicht klar, wo die Grenzen zwischen potenziell unabhängigen Domänen im Monolithen liegen. Große Monolithen gänzlich zu zerlegen, ist ein enorm ressourcenforderndes Unterfangen und macht nur in wenigen Fällen Sinn.

Eng gekoppelte Domänenmodelle können zu erheblichen Herausforderungen führen, insbesondere bei Veränderungen. Sie können zu Konflikten mit anderen Klassen und Modulen des Monolithen beitragen oder zu einem hybriden Konstrukt aus Monolith und Microservices führen, das die Nachteile von beiden Architekturen besitzt. Der Aufwand, einen einzigen Service aus dem Monolithen zu extrahieren, kann enorm sein. Daher empfehlen wir, den Migrationsprozess nur schrittweise mit kleinen Teilen des Monolithen durchzuführen und jeden neuen Service nach dem bewährten DDD-Prinzip zu entwickeln.

Hierbei sollte jeder Service mit einem eigenen Domänenmodell erstellt und über ACL geschützt werden. Durch die schrittweise Extraktion wird der Monolith durch den bewährten Strangler Fig Pattern ersetzt. Die Aufteilung des Monolithen in unabhängige Microservices erfordert eine klare Abgrenzung der Daten für jeden Service. Um dies zu erreichen, empfehlen wir die Verwendung von Events. Doch für kleinere bis mittelgroße Unternehmen stellt ein solches Projekt eine große Herausforderung dar.

Die verschiedenen Herausforderungen bedeuten mehr Komplexität als bei der monolithischen Architektur der Fall ist und müssen vor Einsatz von Microservices gut abgewogen werden.

4.3 Hat der Einsatz von Microservices Nachteile?

Die kurze Antwort ist: Ja.

Neben den bereits erwähnten Herausforderungen während der Zerlegung müssen Teams sich mit Problemen wie Datenkonsistenz und der Synchronisierung zwischen dem Monolithen und den neu extrahierten Services beschäftigen. Je mehr Services extrahiert werden, desto komplexer wird die Kommunikation, sodass eine Latenzzeit emergiert, die berücksichtigt werden muss.

Tritt dann noch ein Fehler auf, ist durch den Prozessfluss bereits der Schritt herauszufinden, wo der Fehler liegt und wie dieser kommuniziert wurde, problematisch. Grund hierfür sind moderne Clusterstrukturen, durch welche man nicht weiß, wo Anfragen gerade beantwortet wurden.

4.4 Hat es einen Grund, dass Monolithen lange als die primäre Architektur galten?

Monolithen bieten einige klare Vorteile, die Microservices nicht haben.

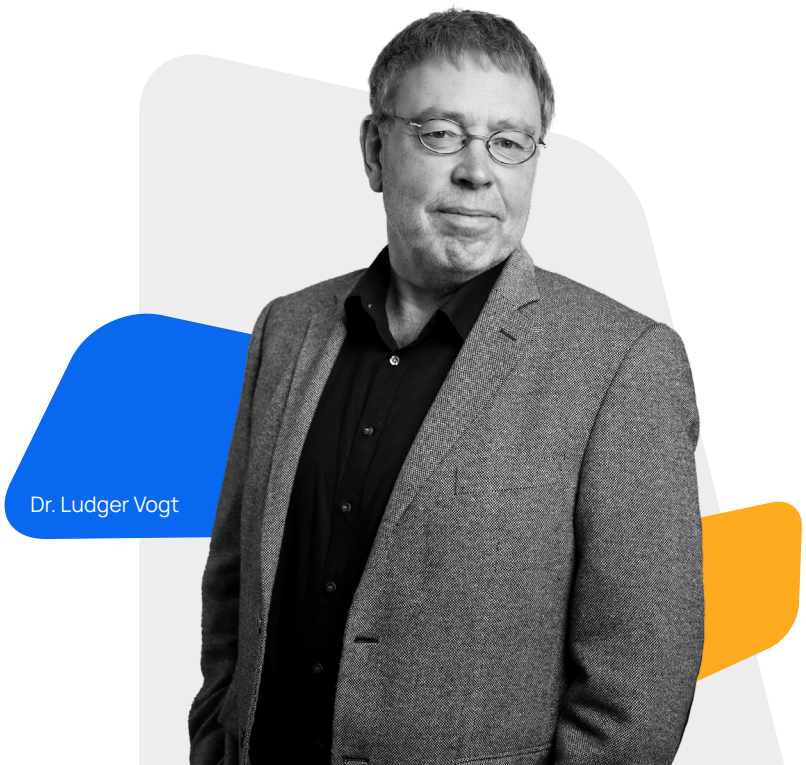
1. Einfachheit in Bereitstellung und Verwaltung

Monolithische Systeme sind unkompliziert zu implementieren, da alle ihre Komponenten Teil einer einzigen, einheitlichen Plattform sind. Diese Einfachheit erstreckt sich auf die Verwaltung, bei der administrative Aufgaben, Updates und Überwachung zentralisiert sind, wodurch die Komplexität, die normalerweise mit der Verwaltung mehrerer Systeme oder Dienste verbunden ist, reduziert wird.

2. Integrierte und einheitliche Umgebung

In einer monolithischen Architektur sind alle Komponenten der eCommerce-Plattform, einschließlich Frontend, Backend, Datenbank und serverseitigen Anwendungen, eng integriert. Diese Integration stellt sicher, dass die Komponenten nahtlos zusammenarbeiten, was eine konsistente und zuverlässige Benutzererfahrung bietet. Es vereinfacht auch Aufgaben wie Datenmanagement und abteilungsübergreifende Arbeitsabläufe.

> **„Einem ‚Standard-Shop' ist es erstmal egal, ob er auf Basis monolithischer Architektur oder auf Microservices fußt. Bei solchen ‚Standard-Shops' und bei Projekten, die schnell Live gehen wollen, überwiegen im Sinne von (Kosten-)effizienz deshalb meist monolitische Lösungen."**
>
> Dr. Ludger Vogt, CEO



3. Kosteneffizienz (kurzfristig)

Anfangs können monolithische Architekturen aufgrund niedrigerer Entwicklungs- und Bereitstellungskosten wirtschaftlicher sein. Die einheitliche Natur der Architektur bedeutet, dass weniger Zeit und Ressourcen für die Integration verschiedener Systeme oder Dienste aufgewendet werden müssen. Für Start-ups und kleine Unternehmen, oder Projekte unter Zeitdruck kann dies ein bedeutender Vorteil sein, da sie damit eine Online-Präsenz mit einem geringeren Ressourcenaufwand etablieren können.

4.5 Nachteile der monolithischen eCommerce Architektur

1. Skalierbarkeitsherausforderungen

Wenn das Geschäft stark wächst, werden die Grenzen einer monolithischen Architektur deutlich. Das Skalieren einer monolithischen Anwendung bedeutet oft, das gesamte System zu skalieren, auch wenn nur bestimmte Komponenten eine erhöhte Last bewältigen müssen. Dies kann zu Ineffizienzen und erhöhten Kosten führen, da zusätzliche Ressourcen verbraucht werden, ohne vollständig genutzt zu werden.

2. Schwierigkeiten bei der Implementierung von Updates und neuen Funktionen

Die Implementierung neuer Funktionen oder Updates in einer monolithischen Architektur kann umständlich und riskant sein. Da alle Komponenten eng miteinander verbunden sind, können Änderungen in einem Bereich unbeabsichtigte Konsequenzen in anderen Bereichen haben, was die Komplexität und Dauer von Entwicklungszyklen erhöht. Dies kann die Fähigkeit eines Unternehmens, schnell auf Marktveränderungen zu reagieren und zu innovieren, behindern.

3. Potenzial für systemweite Ausfälle

In einem monolithischen System kann ein Fehler in einer Komponente das gesamte System beeinträchtigen und zu potenziellem Ausfall und Dienstunterbrechungen führen. Das Risiko systemweiter Ausfälle erfordert umfassende Tests und Qualitätssicherungsprozesse, die ressourcenintensiv sein können. Diese Verwundbarkeit steht im Gegensatz zu modulareren Architekturen, bei denen Probleme in einem Bereich weniger wahrscheinlich das gesamte System beeinflussen.

5. Unsere Empfehlungen für Ihr Projekt

Welchen Unternehmen verschafft der Einsatz von Microservices Vorteile und bei welchen Arbeitsprozessen sind sie besonders hilfreich?

> **„Das hängt stark von dem Projekt ab. Die Art von Architektur ist sehr nützlich für Enterprise Projekte. Doch man sollte nicht anfangen, sein Projekt auf Basis von Microservices aufzubauen oder „per default“ Microservices nutzen, da diese Architektur auch neue Komplexität und Aufwände mit sich bringt. Vielmehr ist es ratsam, Projekte als Monolith anzufangen und wenn es einen echten Nutzen gibt, geeignete Funktionalitäten zu migrieren. Auch wenn Skalierungsprobleme auftreten, kann es sinnvoll sein Microservices einzusetzen. Wenn es den echten Nutzen am Anfang eines Projektes gibt, kann man natürlich auch mit einzelnen Microservices starten. Wichtig ist, dass das Ganze bewusst passiert.“**
>
> Thomas Ernst, Head of Backend

5.1 Kostenabwägung

Vor der Entscheidung, auf eine Microservice-Architektur zu migrieren, sollten Kosten und Vorteile abgewogen worden sein. Es ist klar, dass die Microservice-Architektur Vorteile gegenüber der monolithischen Architektur mit sich bringt, aber die Migration kann teuer werden, wenn sie nicht angemessen durchgeführt wird oder unvorhergesehene Schwierigkeiten auftreten. Weiterhin müssen die erwähnten Herausforderungen bei der Bestimmung der Größe des Service und seiner Grenzen sowie bei der Wahrung der Konsistenz berücksichtigt werden.

5.2 Fehlerresponse

Bei der Verwendung von Microservices müssen die Anwendungen so konzipiert sein, dass sie den Ausfall einzelner Services tolerieren können. Es ist wichtig, dass die Fehler schnell erkannt und die Services nach Möglichkeit automatisch wiederhergestellt werden können, da sie jederzeit ausfallen könnten. Monitoring gewährleistet ein Frühwarnsystem, wenn etwas schiefläuft, und alarmiert die Entwicklungsteams, damit sie sich mit den Fällen befassen.

5.3 Komplexität

Unter dem Gesichtspunkt der Komplexität steht die Kosten-Nutzenrechnung im Zentrum. Sollen abseits vom klassischen eCommerce Applikationen eingesetzt werden, eignet sich die Ergänzung mittels Microservices möglicherweise. Ein praktisches Beispiel wäre eine komplexe Produktkonfiguration. Unternehmen, die ihre Produktdaten nicht im PIM abliegen haben, sondern diese für ihre Kunden zuschneiden müssen, können dies über ein Microservice integrieren. Ein weiteres Beispiel sind Konfiguratoren im Bestellprozess von Autos, die verschiedenen Bestandteile wie Sitze, Heizungen, Motor etc. zu konfigurieren lassen. Dies ist häufig dank Microservices möglich. Soll ein Shop jedoch nur eine Reihe von Produkten verkaufen, dann ist es einfacher, mit einem Monolithen zu starten. Die Komplexität liegt am Ende bei den Monolithen im Code. Bei den Microservices liegt sie in der Netzwerkebene. Worauf am meisten Wert gelegt werden sollte, hängt stark von den Anforderungen des jeweiligen Projekts ab.

5.4 Teamzusammenstellung

Die erfolgreiche Migration kann nur mit einem gut ausgebildeten Team durchgeführt werden. Es erfordert, dass die Teammitglieder ein gutes Verständnis hinsichtlich des Monolithen und dessen Domänenmodell haben. Dies garantiert man am besten mit einem Pilotprojekt, anhand dessen der neue Ansatz bewertet wird und dem Team die Möglichkeit geboten wird, Erfahrungen mit dem Migrationsprozess zu sammeln.

Die vertikale Aufteilung der Organisationsstruktur in funktionsübergreifende Teams analog zur Microservice-Struktur ermöglicht die Skalierung der Entwicklungskapazitäten entsprechend den veränderlichen Geschäftsanforderungen. Die Teams sollten in hohem Maße unabhängig sein und aus Mitgliedern aller Rollen und unterschiedlicher Fähigkeiten bestehen, die für die Erstellung und Wartung ihrer Microservices erforderlich sind. Microservices verstärken die modulare Struktur, was besonders für größere Teams wichtig ist. Die Entkopplung von Teams ist ebenso wichtig wie die Entkopplung von Softwaremodulen.

5.5 Fazit

Microservices bieten eine Vielzahl an Vorteilen für Unternehmen, die auf der Suche nach Flexibilität und Effizienz in der Entwicklung und Bereitstellung ihrer eCommerce-Anwendungen sind. Die Möglichkeit, schnell zu testen, bereitzustellen, zu skalieren und agile Entwicklungsmethoden in unabhängigen Teams zu nutzen, macht Microservices in solchen Fällen besonders attraktiv. Auch wenn sich Unternehmensprozesse nicht nahtlos in die Struktur eines Monolithen integrieren lassen, oder wenn das IT-Projekt über eine einfache eCommerce Lösung hinausgeht, kann der Einsatz von Microservices sich lohnen.

Allerdings kommt die Verwaltung dieser Vorteile nicht ohne Herausforderungen. Einige Beispiele für solche Herausforderung sind das Orchestrieren der verschiedenen Services und die Fehlersuche über mehrere Schnittstellen hinweg. Zudem wächst die Komplexität mit der Anzahl der Services.

Die Entscheidung zwischen der Implementierung eines Microservice-Architektur Modells oder eines traditionellen Monolithen hängt stark von den spezifischen Anforderungen und Zielen Ihres Unternehmens ab. Ein monolithischer Ansatz kann vorteilhaft sein, wenn Ihre Anwendung eng mit etablierten Prozessen verknüpft ist und Softwareanpassungen unkompliziert ermöglicht. Dank ihrer längeren Entwicklungszeit bieten Monolithen hierfür in der Regel ein reichhaltigeres Spektrum an Funktionalitäten. Letztendlich kann auch ein hybrider Ansatz, der die Stärken beider Architekturen vereint, für Ihr Unternehmen die ideale Lösung sein. Eine solche Strategie ermöglicht es, von der robusten Funktionalität eines Monolithen zu profitieren, während gleichzeitig die Flexibilität und Skalierbarkeit von Microservices genutzt werden. Das hilft dabei, neue Geschäftsfelder zu erschließen oder bestehende Systeme effizient zu ersetzen.

Doch egal wie Sie sich entscheiden:
Die Wahl des richtigen Ansatzes beginnt immer mit einer gründlichen Evaluation Ihrer geschäftlichen Anforderungen und technischen Kapazitäten und sollte niemals ohne die notwendige Erfahrung getroffen werden.

Smart Commerce.

Bei der Smart Commerce SE kombinieren wir die jahrelange Erfahrung unserer eCommerce & Digital Consulting Expert:innen mit technischer Expertise in den Kernbereichen: eCommerce Plattformen, eCommerce Cloud, CMS, CRM und Digital Marketing. So können wir nach umfassenden Analysen problemspezifische Lösungen in Kombination mit einer individuellen und ganzheitlichen Strategie für unsere Kund:innen anbieten.

Unser kompetentes TEC-Team bestehend aus Senior Project Consultants, erfahrenen Software-Architekt:innen, Software-Entwickler:innen, Web-Entwickler:innen sowie Betriebs-Expert:innen mit tiefgehender eCommerce-Erfahrung bietet ein umfassendes Dienstleistungspaket für High End-eCommerce Plattformen und die digitale Unternehmenstransformation. Dabei hat für uns oberste Priorität, unsere Kund:innen von Anfang an mitzunehmen und Projekte gemeinsam aufzubauen, damit sie zu jeder Zeit den kompletten Weg und das Ziel transparent im Blick haben. Denn wir haben uns dem langfristigen Online-Erfolg unserer Geschäftspartner:innen verschrieben. Das heißt für uns, nachhaltige, nutzerorientierte und datenzentrierte Lösungen bereitzustellen.

Seit über 10 Jahren folgen wir dem Smart Way, der unsere Unternehmenskultur von Beginn an prägt: Wir sind ein mitarbeiter:innenzentriertes Unternehmen. Unsere 120+ Mitarbeiter:innen sind Mitunternehmer:innen und Mitgestalter:innen. Sie halten die Aktien und gestalten die Unternehmung von arbeitsplatzspezifischen Entscheidungen bis hin zu unternehmerischen Werten, Zielen und Führungsgrundsätzen aktiv mit.

Technologiepartner.

intershop · SAP · room · Spryker · FACT FINDER · PIWIK PRO · SITECORE · commercetools

Referenzkunden.

ALTANA · baur. · Town & Country HAUS · MISTER SPEX · Linde Material Handling · SEEFELDER · medi · CALUMET PHOTOGRAPHIC · FORCH · fischer · GEHE · MERCK · KION GROUP · MIVO · HELUKABEL · FIBRO · nexmart

Impressum

Smart Commerce SE
Steinweg 10 · 07743 Jena · Germany
Telefon: +49 3641 3 16 10 20
Telefax: +49 3641 3 16 10 22

E-Mail: hello@smartcommerce.de
Internet: www.smartcommerce.de

Vertretungsberechtigte Vorstände:
Dr. Ludger Vogt, Frank Schneider, Ingo Körber
Registergericht: Amtsgericht Jena
Registernummer: HRB 507999
USt-IdNr: DE284448889
Inhaltlich Verantwortlicher gemäß
§ 55 Abs. 2 RStV: Frank Schneider
Autor: David Bredenbeck

Interessiert an Ihrer zukünftigen Erfolgsstrategie?
Einfach per E-Mail: **hello@smartcommerce.de**

Smart Commerce

Kontakt Mail:
hello@smartcommerce.de

Standort Jena (HQ):
Steinweg 10 · 07743 Jena, Germany
Tel.: +49 3641 3 16 10 20

Standort Böblingen:
Otto-Lilienthal-Straße 36 · 71034 Böblingen, Germany
Tel.: +49 711 18 42 01 00

Standort Leipzig:
Naumburger Straße 25 · 04229 Leipzig, Germany
Tel: +49 341 99 15 36 00

www.smartcommerce.de