Microservices

# THINK BIG, START SMALL: MIT GROSSEN SCHRITTEN KLEINE SERVICES ENTWICKELN

**Haben Sie schon einmal einen Berg bestiegen? Keinen Achttausender, sondern einen weniger anspruchsvollen Alpengipfel, der ohne spezielle Ausrüstung zu erklimmen ist. Im Tal ist Ihnen die Aufgabe klar: Sie wollen dort oben ankommen. Da Sie aber keine Seilbahn nehmen möchten, geht das nicht in einem Anlauf. Vielmehr sind viele kleine Schritte notwendig, die für sich genommen jeweils ein Mikroprojekt sind: Einen Fuß vor den anderen setzen, nicht abrutschen, Luft holen und den nächsten Schritt machen.**

Dieser Vergleich umschreibt in etwa das Prinzip von Microservices in der Anwendungsentwicklung: Ein in kleinere Projekte bzw. Services zerlegte Applikation sorgt unter anderem für verbesserte Skalierung und eröffnet mehr Freiräume bei der Wahl einer Hosting-Plattform wie Cloud-Anbieter oder das eigene Rechenzentrum. Es entsteht also keine monolithische Applikation, die zentral auf einem Server läuft, sondern eine modulare und verteilte Gesamtanwendung. Dies allein ist natürlich noch keine Besonderheit. Vielmehr sind bei einer Microservices-Architektur die fachlichen Services, die hier als Domänen bezeichnet werden, unabhängig voneinander lauffähig und werden auch separat entwickelt.

Aus Anwendersicht ist das Ergebnis nach wie vor eine einzige Applikation. Aus Perspektive eines Entwicklers gleicht die Infrastruktur einer Service-orientierten Architektur (SOA), mit dem Unterschied, dass kein zentraler Enterprise Service Bus (ESB) mehr benötigt wird.

Die Kommunikation zwischen den einzelnen und verteilt laufenden Microservices erfolgt über etablierte und leichtgewichtige Protokolle, die nur geringe Ressourcen und Systemleistung benötigen. Dazu zählt beispielsweise REST (Representational State Transfer), das auf Basis des Internet-Protokolls HTTP (Hypertext Transfer Protocol) arbeitet.

## Warum wird eine Microservice-Architektur benötigt?

Da Microservices eigenständig ablaufen und nur lose über Standardschnittstellen gekoppelt sind, lassen sich einzelne Funktionen einer Applikation freier skalieren. Damit bleiben beispielsweise Antwortzeiten von Online-Shops auch bei hohen Besucherzahlen konstant. Darüber hinaus erhalten Unternehmen mehr Spielraum bei der Organisation ihrer Entwicklerteams: Je nach Relevanz eines fachlichen Services können Verantwortliche die benötigte Menge an Experten oder anderen Ressourcen flexibel reservieren. Auch die verwendeten Technologien lassen sich je nach Entwicklerteam und fachlicher Anforderung variieren. Ebenfalls denkbar: Ein Technologiewechsel zu einem späteren

Zeitpunkt, um beispielsweise von kommerziellen Lösungen auf Open Source-Produkte zu wechseln.

Ein weiterer wichtiger Aspekt ist die Möglichkeit, einen Service bewusst mit einem kleinen Funktionsumfang zu entwickeln. Damit wird eine kurze Entwicklungszeit für eine Komponente erreicht. Der Vorteil: Ist ein Fehler im Service vorhanden, die Skalierung nicht ausreichend oder machen veränderte Marktbedingungen eine Neuprogrammierung notwendig, kann das Team zeitnah eine neue Version entwickeln. Das Risiko für die Gesamtapplikation ist gering, da nur ein Teil der Lösung verändert wird. Bei monolithischen Systemen ist ein wesentlich höherer Aufwand notwendig, im Extremfall muss die komplette Applikation als Update neu ausgerollt werden.

## Für wen sich Microservices eignen

Es gibt einige Szenarien, für die sich eine Microservices-Architektur besonders gut eignet. Dazu zählen Märkte mit hohen Wachstumsraten und starken Veränderungen, wodurch immer wieder Anpassungen der Software notwendig sind. Aber auch Firmenübernahmen, neue Produkte und die Expansion in neue Märkte verlangen nach Software-Änderungen. In diesen dynamischen Umgebungen kann eine kontinuierliche und agile Anwendungsentwicklung signifikant zum Unternehmenserfolg beitragen, beispielsweise durch Effizienzsteigerungen auf Basis neuer Applikationen.

## Bei einem Tausendfüßler die Schritte zählen

Auf der anderen Seite verlangt der Einsatz von Microservices neue Fähigkeiten innerhalb der IT-Organisation. So entsteht aus IT-Sicht durch die Vielzahl an eigenständigen Komponenten ein komplexes und verteiltes System. Zudem ändert sich in einer Microservice-Architektur dauernd etwas. Es ist deshalb wichtig, systemweit die Verfügbarkeit der Dienste zu monitoren und Fehlerkaskaden zu vermeiden, die es ansonsten erschweren, den wahren Grund eines Fehlers zu identifizieren. Durch die zahlreichen verteilten Systeme gibt es entsprechend mehr potenzielle Fehlerquellen. So sollten die Überwachung und das Monitoring der Software-Landschaft deutlich intensiver und engmaschiger betrieben werden. Zusätzlich zu dem feingranularen Monitoring ist eine entsprechende Reaktionsfähigkeit der IT-Experten notwendig, um Ausfallzeiten bei den Geschäftsprozessen zu minimieren. Außerdem verlangt ein effizienter Systembetrieb eine Automatisierung vieler Abläufe innerhalb der IT-Organisation.

Unternehmen scheuen oftmals den Umstieg auf Microservices, da die eigenen IT-Experten mit solchen Architekturen kaum Erfahrung haben und auch nicht die passenden IT-Tools für die Automatisierung und das Monitoring verwenden. Hier können externe Dienstleister dabei helfen, die eigene IT-Organisation effizienter aufzustellen. Lösungen wie von BMC oder ServiceNow bieten entsprechende Funktionen, um den Automatisierungsgrad innerhalb der IT zu steigern. Ebenfalls wichtig: Beim Monitoring und dem Systembetrieb ist darauf zu achten, dass die ausgewählten Tools auch tatsächlich für alle Technologien und Betriebsmodelle von On-premise bis Multi-Cloud geeignet sind – schließlich ermöglicht das Prinzip der Microservices, dass jedes Entwicklungsteam theoretisch einen eigenen Technologie-Stack verwendet.

## So gelingt der Start

Im ersten Schritt sollten Unternehmen die fachlichen Aspekte analysieren. In Hinblick auf die weitere Software-Entwicklung erfolgt an dieser Stelle auch die Definition von Domänen, also eine Aufteilung von Prozessen in einzelne Services. Für Microservices eignen sich insbesondere Funktionen, die besonders stark nachgefragt werden und eine entsprechend hohe Last innerhalb der IT-Infrastruktur erzeugen können. Für den Start sollten Unternehmen einen überschaubaren und risikoarmen Prozess auswählen und diesen als Microservice definieren.

In der zweiten Phase wird die für den Betrieb von Microservices benötigte technologische Infrastruktur aufgebaut und in die bestehende Betriebslandschaft integriert. Für einen ersten Test lassen sich Cloud-Ressourcen nutzen, um einen Microservice schnell und ohne größere Investitionen aufzusetzen. Container-Technologien wie Docker auf Kubernetes eignen sich ideal für eine Microservices-Architektur, da diese einen hohen Grad an Skalierbarkeit und Portabilität bieten.

## Richtige Oberfläche auswählen

Die aus einzelnen Diensten zusammengesetzte Applikation muss schließlich dem Anwender auch präsentiert werden. Hierfür wird ein Konzept zur Realisierung der Frontends bzw. Oberflächen benötigt. Möglich ist, dass jeder Microservice sein eigenes Frontend mitbringt, sodass die Benutzeroberfläche erst bei Aufruf des Dienstes im Internet-Browser

des Anwenders aufgebaut wird. Die Alternative ist eine zusätzliche Integrationsschicht, die die Services zusammenführt und dem Anwender als zusammenhängende Anwendung präsentiert. Bei der Konzeption sollte auch die mobile Nutzung der Software auf verschiedenen Endgeräten berücksichtigt werden.

## Jetzt wird in die Hände gespuckt

Nach dieser Konzeptionsphase beginnt die eigentliche Entwicklung des Microservices. Hierfür existieren eine Vielzahl bewährter Technologien, wie das Spring Framework für die Java-Welt. Hier hat sich die Lösung Spring Boot als Entwicklungsumgebung für Enterprise Java-Applikationen etabliert, um schnell einen fertigen Microservice aufzubauen. Bei einem Modernisierungsprojekt kann der neue Service nun bereits losgelöst von der Legacy-Anwendung betrieben werden.

## In die psychologische Trickkiste greifen

Für den Entwicklungsprozess ist es entscheidend, dass sich die Beteiligten von Beginn an auf eine agile Arbeitsweise einstellen. Wichtig sind kurze Feedbackschleifen, um Probleme zu korrigieren oder geänderte Anforderungen von Kundenseite zu berücksichtigen. Damit wird auch ein zentrales Problem vieler Entwicklungsprojekte adressiert: Die Angst davor, Fehler zu machen. Bei sehr kurzen Release-Zyklen ist ein Fehler schnell korrigiert. Anders bei einer monolithischen Applikation, wenn ein Fehler vielleicht erst nach dem globalen Rollout sichtbar wird. Ein zusätzlicher Effekt ist, dass bei Entwicklern auch die Kreativität zunimmt, denn sie trauen sich eher einmal, einen neuen Weg zu gehen, wohl wissend, dass sich Fehler schnell wieder korrigieren lassen.

Die kurzen Zyklen lassen sich durch einen hohen Automatisierungsgrad erreichen, wie ein automatisiertes Deployment. Diese Vorgehensweise schlägt sich übrigens nicht auf die Qualität der Software nieder. Dafür sind entsprechende Verfahren innerhalb des Konzepts für Continuous Integration und Continous Delivery (CI/CD) vorhanden, das bei der agilen Entwicklung ohnehin genutzt werden sollte.

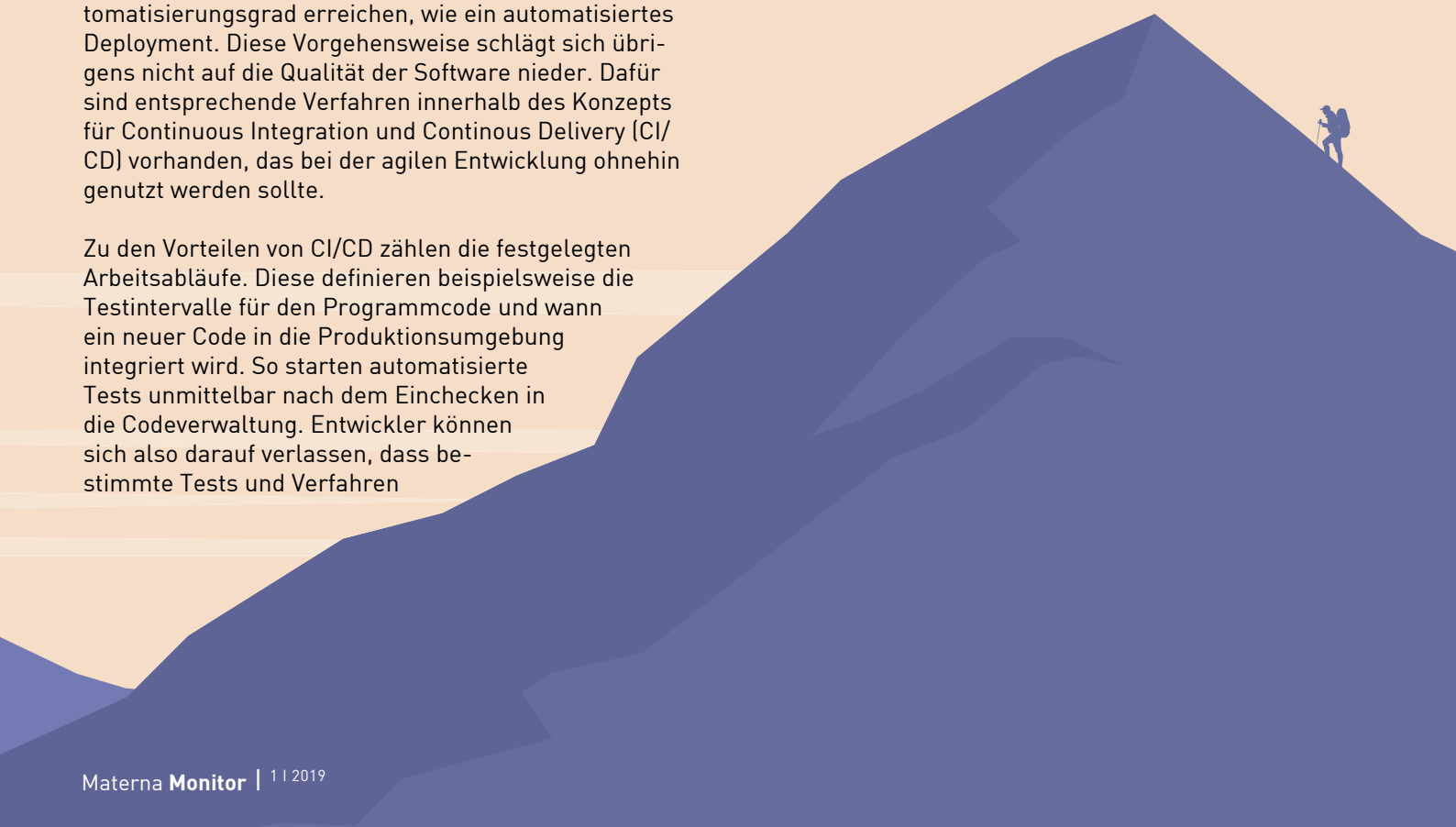
Zu den Vorteilen von CI/CD zählen die festgelegten Arbeitsabläufe. Diese definieren beispielsweise die Testintervalle für den Programmcode und wann ein neuer Code in die Produktionsumgebung integriert wird. So starten automatisierte Tests unmittelbar nach dem Einchecken in die Codeverwaltung. Entwickler können sich also darauf verlassen, dass bestimmte Tests und Verfahren zur Qualitätssicherung automatisiert ablaufen. Darauf aufbauend erfolgt mit Continuous Delivery die Bereitstellung neuer Komponenten in einer produktiv genutzten Applikation. CD erfolgt ebenfalls komplett automatisiert und befreit somit Entwickler von fehleranfälligen manuellen Routineaufgaben. Das Ergebnis ist eine Deployment-Pipeline, die neue Features schnell in den Code integriert. Entsprechend zeitnah kommt Rückmeldung von den Anwendern zu den neuen Funktionen, die dann wieder in den Entwicklungsprozess einfließen.

## Funktioniert auch in der Praxis hervorragend

Die hier beschriebenen theoretischen Grundlagen wurden von vielen Unternehmen, wie beispielsweise von Netflix, Amazon oder Twitter, bereits erfolgreich in der Praxis umgesetzt. Aber auch ein großer deutscher Online-Händler hat seine monolithische Shop-Anwendung in eine Microservice-basierende und agile Anwendungslandschaft überführt. Materna unterstützte hierbei durch verschiedene Beratungsleistungen und einen kontinuierlichen Know-how-Transfer.

## Fazit

Neue Technologien verändern derzeit rasant bestehende Geschäftsmodelle. Unternehmen müssen auf das Tempo reagieren, indem sie schneller neue Software-Anwendungen entwickeln. Diese müssen die Anforderungen einer globalen Internet-Wirtschaft unterstützen: Dazu zählen eine hohe Skalierbarkeit, um auf Nachfrageschwankungen reagieren zu können, aber auch die effiziente Nutzung von Cloud-Ressourcen und die Integration über offene API-Schnittstellen. Letzte sind besonders wichtig, möchte ein Unternehmen seine Leistungen in globale Marktplätze integrieren.



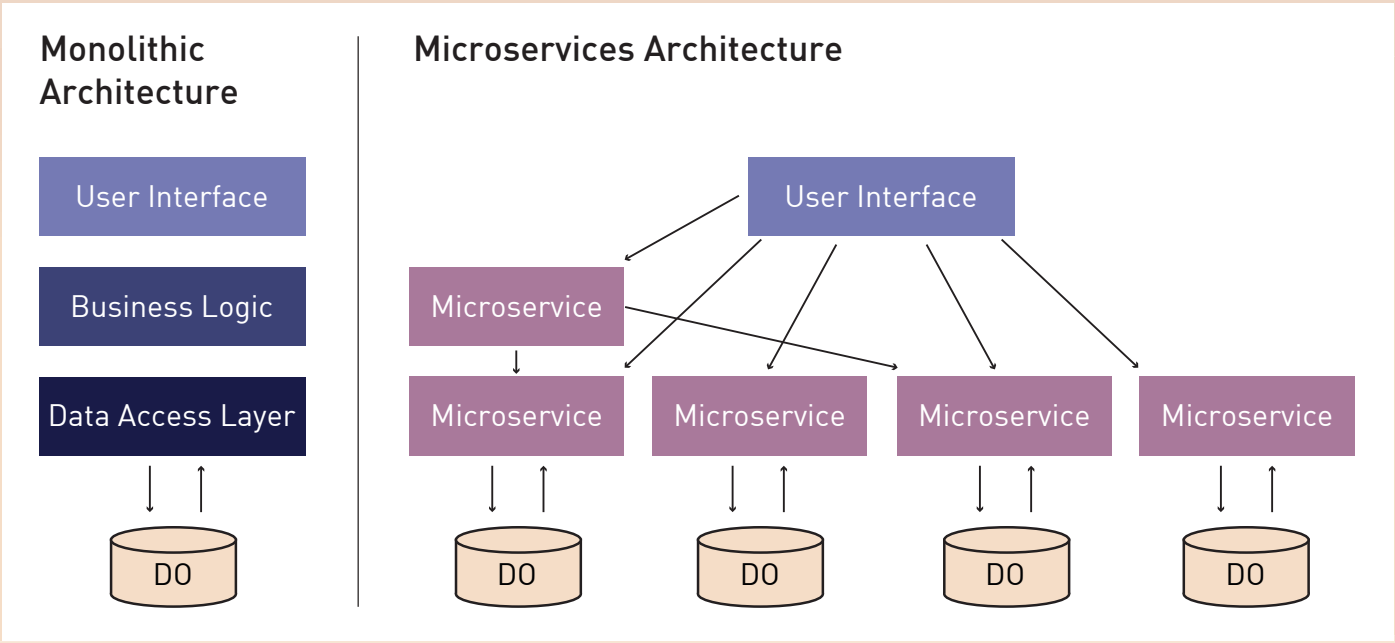
Mit einer Microservices-Architektur wird Software flexibel veränderbar, hochgradig skalierbar und evolutionär wachsend. Der Umstieg sollte jedoch zuvor analysiert werden, da eine höhere Komplexität im laufenden Betrieb entstehen kann. Unternehmen wählen eine Microservices-Architektur insbesondere dann, wenn die Flexibilität bei der Entwicklung einen tatsächlichen Mehrwert bringt oder die Marktsituation eine permanente Anpassung von Anwendungen erfordert. Wer Microservices effizient nutzen möchte, benötigt zudem einen höheren digitalen Reifegrad, da hierbei beispielsweise Konzepte wie Continuous Delivery und Infrastructure Automation dringend empfohlen werden.

## Leistungen Materna

- Beratung und Aufbau von Microservices-Architekturen
- Architektur-, Code- und Entwicklungsprozess-Reviews
- Optimierung und Automatisierung des DevOps-Prozesses
- Beratung, Konzeption und Umsetzung von Applikationen mit Java, JavaScript und .NET sowie Cloud-native Anwendungen für AWS, Azure, IBM Cloud, Kubernetes, Docker
- Migration bestehender Anwendungen in die Cloud
- Continuous Integration and Delivery (CI/CD)
- Agile Vorgehensweisen (Scrum und Kanban)

## Über den Autor

André Briskorn ist Senior Software Architekt bei Materna und spezialisiert auf die Entwicklung von Microservices-Architekturen. Er hat in verschiedenen agilen Software-Projekten als Architekt und Full-Stack-Entwickler mitgewirkt. Seine Schwerpunkte sind Java-Entwicklung, Microservices, agile Methoden, CI/CD-Pipelines, DevOps und Cloud Computing.




Die Abbildung zeigt den Unterschied zwischen einer monolithischen Software-Architektur und einer Microservices-Architektur. In der Microservices-Architektur ist auch die Datenhaltung getrennt, sodass die einzelnen Microservices auch wirklich eigenständig werden.