1 Vorwort

Microservices sind ein neuer Begriff – aber sie verfolgen mich schon lange. 2006 hielt Werner Vogels (CTO, Amazon) einen Vortrag auf der JAOO-Konferenz, wo er die Amazon Cloud und Amazons Partnermodell vorstellte [1]. Dabei erwähnte er das CAP-Theorem – heute Basis für NoSQL. Und dann sprach er von kleinen Teams, die Services mit eigener Datenbank entwickeln und auch betreiben. Diese Organisation nennen wir heute DevOps und die Architektur Microservices.

Später sollte ich für einen Kunden eine Strategie entwickeln, wie er moderne Technologien in seine Anwendung integrieren kann. Nach einigen Versuchen, neue Technologien direkt in den Legacy-Code zu integrieren, haben wir schließlich eine neue Anwendung neben der alten Anwendung mit einem völlig anderen modernen Technologie-Stack aufgebaut. Die neue und die alte Anwendung waren nur über HTML-Links gekoppelt – und über die gemeinsame Datenbank. Bis auf die gemeinsame Datenbank ist auch dieses Vorgehen im Kern ein Microservices-Ansatz. Das war 2008.

Ein anderer Kunde hatte schon 2009 seine komplette Infrastruktur in REST-Services aufgeteilt, die jeweils von einzelnen Teams weiterentwickelt wurden. Auch das nennen wir heute Microservices. Viele andere Unternehmen aus dem Internet-Bereich hatten damals schon ähnliche Architekturen.

In letzter Zeit wurde mir außerdem klar, dass Continuous Delivery [2] Auswirkungen auf die Software-Architektur hat. Auch in diesem Bereich haben Microservices viele Vorteile.

Und das ist der Grund für das Buch: Microservices sind ein Ansatz, den einige schon sehr lange verfolgen; darunter auch viele sehr erfahrene Architekten. Wie jeder Architekturansatz löst er sicher nicht alle Probleme – aber er kann eine interessante Alternative darstellen.

1.1 Überblick über Microservices

Microservice: vorläufige Definition

Im Mittelpunkt des Buchs stehen Microservices – ein Ansatz zur Modularisierung von Software. Modularisierung ist nichts Neues. Schon lange werden große Systeme in kleine Module unterteilt, um Software einfacher zu erstellen, zu verstehen und weiterzuentwickeln.

Das Neue: Microservices nutzen als Module einzelne Programme, die als eigene Prozesse laufen. Der Ansatz basiert auf der UNIX-Philosophie. Sie lässt sich auf drei Aspekte reduzieren:

- Ein Programm soll nur eine Aufgabe erledigen, und das soll es gut machen.
- Programme sollen zusammenarbeiten können.
- Nutze eine universelle Schnittstelle. In UNIX sind das Textströme.

Der Begriff Microservice ist nicht fest definiert. Kapitel 4 zeigt eine genauere Definition. Als erste Näherung dienen folgende Kriterien:

- Microservices sind ein Modularisierungskonzept. Sie dienen dazu, ein großes Software-System aufzuteilen – und beeinflussen die Organisation und die Software-Entwicklungsprozesse.
- Microservices können unabhängig voneinander deployt werden. Änderungen an einem Microservice können unabhängig von Änderungen an anderen Microservices in Produktion gebracht werden.
- Microservices können in unterschiedlichen Technologien implementiert sein. Es gibt keine Einschränkung auf eine bestimmte Programmiersprache oder Plattform.
- Microservices haben einen eigenen Datenhaushalt: eine eigene Datenbank – oder ein vollständig getrenntes Schema in einer gemeinsamen Datenbank.
- Microservices können eigene Unterstützungsdienste mitbringen, beispielsweise eine Suchmaschine oder eine spezielle Datenbank. Natürlich gibt es eine gemeinsame Basis für alle Microservices – beispielsweise die Ausführung virtueller Maschinen.
- Microservices sind eigenständige Prozesse – oder virtuelle Maschinen, um auch die Unterstützungsdienste mitzubringen.
- Dementsprechend müssen Microservices über das Netzwerk kommunizieren. Dazu nutzen Microservices Protokolle, die lose Kopplung unterstützen. Das kann beispielsweise REST sein – oder Messaging-Lösungen.

Dieser Ansatz betrachtet die Größe des Microservice nicht. Trotz des Namens »Microservice« ist die Größe für eine grobe Definition nicht so entscheidend.

Monolithen

Microservices grenzen sich von Deployment-Monolithen ab. Ein Deployment-Monolith ist ein großes Software-System, das nur als Ganzes auf einmal deployt werden kann. Es muss als Ganzes durch alle Phasen der Continuous-Delivery-Pipeline wie Deployment, Test, Abnahme und Release laufen. Durch die Größe des Deployment-Monolithen dauert dieser Prozess länger als bei kleineren Systemen. Das reduziert die Flexibilität und erhöht die Kosten der Prozesse. Der Deployment-Monolith kann intern modular aufgebaut sein – nur müssen alle diese Module gemeinsam in Produktion gebracht werden.

1.2 Warum Microservices?

Microservices dienen dazu, Software in Module aufzuteilen und dadurch die Änderbarkeit der Software zu verbessern.

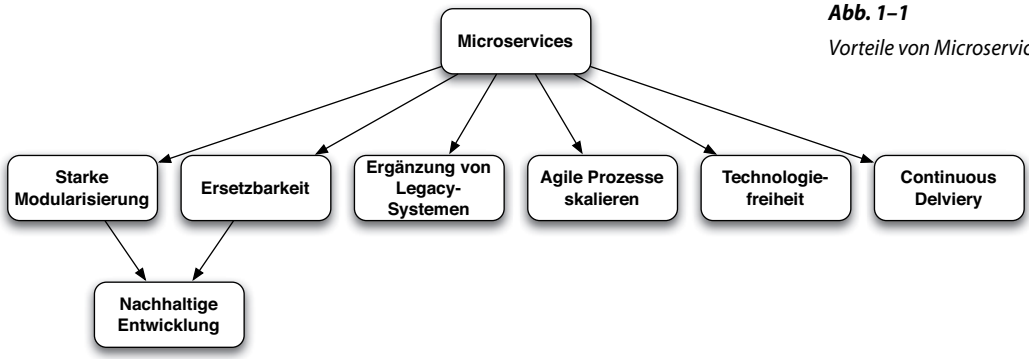

Abb. 1–1
Vorteile von Microservices

Microservices haben einige wesentliche Vorteile:

Starke Modularisierung

- Microservices sind ein *starkes Modularisierungskonzept*. Wird ein System aus Software-Komponenten wie Ruby GEMs, Java JARs, .NET Assemblies oder Node.js NPMs zusammengestellt, schleichen sich leicht ungewünschte Abhängigkeiten ein. Irgendwo referenziert jemand eine Klasse oder Funktion, wo sie eigentlich nicht genutzt werden soll. Und nur wenig später sind im System so viele Abhängigkeiten, dass eine Wartung oder Weiterentwicklung praktisch unmöglich ist. Microservices hingegen kommunizieren über explizite Schnittstellen, die mit Mechanismen wie Messages oder REST umgesetzt sind. Dadurch sind die technischen Hürden für die Nutzung eines Microservice höher. So schleichen sich ungewünschte Abhängigkeiten kaum ein. Es sollte zwar möglich sein, auch in Deployment-Monolithen eine gute Modularisierung zu erreichen. Die Praxis zeigt aber, dass die Architektur von Deployment-Monolithen meistens zunehmend schlechter wird.

Leichte Ersetzbarkeit

- Microservices können leichter ersetzt werden. Andere Komponenten nutzen einen Microservice über eine explizite Schnittstelle. Wenn ein Service dieselbe Schnittstelle anbietet, kann er den Microservice ersetzen. Der neue Microservice muss weder die Code-Basis noch die Technologien des alten Microservice übernehmen. An solchen Zwängen scheitert oft die Modernisierung von Legacy-Systemen. Kleine Microservices erleichtern die Ablösung weiter. Gerade die Ablösung wird bei der Entwicklung von Systemen oft vernachlässigt. Wer denkt schon gerne darüber nach, wie das gerade erst geschaffene wieder ersetzt werden kann? Die einfache Ersetzbarkeit von Microservices reduziert außerdem die Kosten von Fehlentscheidungen. Wenn die Entscheidung für eine Technologie oder einen Ansatz auf einen Microservice begrenzt ist, kann im Extremfall einfach der Microservice komplett ersetzt werden.

Nachhaltige Software-Entwicklung

- Die starke Modularisierung und die leichte Ersetzbarkeit erlauben eine nachhaltige Software-Entwicklung. Meistens ist die Arbeit an einem neuen Projekt recht einfach. Bei längerer Projektlaufzeit lässt die Produktivität nach. Ein Grund dafür ist die Erosion der Architektur. Das vermeiden Microservices durch die starke Modularisierung. Ein weiteres Problem sind die Bindung an alte Technologien und die Schwierigkeiten, alte Module aus dem System zu entfernen. Hier helfen Microservices durch die Technologiefreiheit und die Möglichkeit, Microservices einzeln zu ersetzen.

Legacy-Anwendung erweitern

- Der Einstieg in eine Microservices-Architektur ist einfach und bringt bei alten Systemen sogar sofort Vorteile: Statt die unübersichtliche alte Code-Basis zu ergänzen, kann das System mit einem Microservice ergänzt werden. Der kann bestimmte Anfragen bearbeiten und alle anderen dem Legacy-System überlassen. Er kann Anfragen vor der Bearbeitung durch das Legacy-System modifizieren. So muss nicht die gesamte Funktionalität des Legacy-Systems abgelöst werden. Der Microservice ist auch nicht an den Technologie-Stack des Legacy-Systems gebunden und kann mit modernen Ansätzen entwickelt werden.

Time-to-Market

- Microservices erlauben ein besseres Time-to-Market. Wie schon erwähnt, können Microservices einzeln in Produktion gebracht werden. Wenn in einem großen System jedes Team für einen oder mehrere Microservices zuständig ist und Features nur Änderungen an diesen Microservices benötigen, kann das Team ohne weitere Koordinierung mit anderen Teams entwickeln und Features in Produktion bringen. So können Teams ohne große Koordination an vielen Features parallel arbeiten, sodass mehr Features in derselben Zeit in Produktion gebracht werden können als bei einem Deploy-

ment-Monolithen. Microservices helfen dabei, agile Prozesse auf große Teams zu skalieren, indem das große Team in kleine Teams mit eigenen Microservices aufgeteilt wird.

Unabhängige Skalierung

- Jeder Microservice kann unabhängig von den anderen Services skaliert werden kann. Dadurch ist es nicht notwendig, das gesamte System zu skalieren, wenn nur wenige Funktionalitäten intensiv genutzt werden. Das kann oft eine entscheidende Vereinfachung sein.

Technologiefreiheit

- Bei der Umsetzung von Microservices herrscht Technologiefreiheit. Dadurch kann eine neue Technologie in einem Microservice erprobt werden, ohne dass andere Services betroffen sind. Das senkt das Risiko für die Einführung neuer Technologien und neuer Versionen vorhandener Technologien, da sie in einem kleinen Rahmen eingeführt und getestet werden können, in dem die Kosten kalkulierbar sind. Ebenso ist es möglich, spezielle Technologien für bestimmte Funktionalitäten zu nutzen – zum Beispiel eine spezielle Datenbank. Das Risiko ist gering, weil der Microservice jederzeit ersetzt oder entfernt werden kann. Die neue Technologie ist auf einen oder wenige Microservices beschränkt. Das reduziert das Risiko und ermöglicht vor allem unabhängige Technologie-Entscheidungen für unterschiedliche Microservices. Außerdem erleichtert es die Entscheidung für den Einsatz und die Evaluierung von neuen, hoch innovativen Technologien. Das kommt der Produktivität der Entwickler zugute und verhindert das Veralten der Technologie-Plattform. Aktuelle Technologien ziehen außerdem qualifiziertere Mitarbeiter an.

Continuous Delivery

- Für Continuous Delivery [1] sind Microservices vorteilhaft. Die Microservices sind klein und können unabhängig voneinander deployt werden. Die Umsetzung einer Continuous-Delivery-Pipeline ist wegen der Größe des Microservice einfach. Das Deployment eines einzelnen Microservice ist risikoärmer als das Deployment eines großen Monolithen. Es ist also einfacher, das Deployment eines Microservice abzusichern – beispielsweise durch den parallelen Betrieb verschiedener Versionen. Für viele Microservice-Nutzer ist Continuous Delivery der wesentliche Grund für die Einführung von Microservices.

Alle diese Gründe sprechen für die Einführung von Microservices. Welche Gründe am wichtigsten sind, hängt von dem Szenario ab. Die Skalierung agiler Prozesse und Continuous Delivery sind oft aus einer Geschäftssicht wichtig. Kapitel 5 widmet sich den Vorteilen von Microservices im Detail und geht auch auf die Priorisierung ein. Wo so viel Licht ist, ist auch Schatten. Daher wird Kapitel 6 noch detailliert darlegen, welche Herausforderungen bei der Umsetzung von Micro-

services existieren und wie man mit ihnen umgehen kann. Im Wesentlichen sind das die folgenden:

Beziehungen sind versteckt.

- Die Architektur des Systems besteht aus den Beziehungen der Services. Aber ohne Weiteres ist nicht klar, welcher Microservice welchen anderen aufruft. Dadurch wird Architekturarbeit zu einer Herausforderung.

Refactoring ist schwierig.

- Die starke Modularisierung hat auch Nachteile: Refactorings, bei denen Funktionalitäten zwischen Microservices verschoben werden, sind schwer umsetzbar. Die Aufteilung des Systems in Microservices ist nachträglich nur schwer zu ändern. Diese Probleme kann man durch geschicktes Vorgehen abmildern.

Fachliche Architektur ist wichtig.

- Die Aufteilung des Systems in fachliche Microservices ist wichtig, weil dadurch auch die Aufteilung in Teams festgelegt wird. Fehler bei der Aufteilung auf dieser Ebene beeinflussen auch die Organisation. Nur eine gute fachliche Aufteilung kann die unabhängige Entwicklung der Microservices gewährleisten. Da Änderungen an der Aufteilung schwierig sind, können Fehler gegebenenfalls nur schwer korrigiert werden.

Betrieb ist komplex.

- Ein System, das aus Microservices besteht, hat viele Bestandteile, die deployt, überwacht und betrieben werden müssen. Das erhöht die Komplexität im Betrieb und die Anforderungen an die Betriebsinfrastruktur. Microservices erzwingen eine Automatisierung der Betriebsprozesse, da sonst ein Betrieb der Plattform zu aufwendig ist.

Verteilte Systeme sind komplex.

- Die Komplexität für die Entwickler wächst: Ein Microservice-System ist ein verteiltes System. Aufrufe zwischen Microservices können wegen Netzwerkproblemen fehlschlagen. Aufrufe über das Netzwerk sind langsamer und haben eine geringere Bandbreite, als dies bei Aufrufen innerhalb eines Prozesses der Fall wäre.

[1] *http://jandiandme.blogspot.com/2006/10/jaoo-2006-werner-vogels-cto-amazon.html*

[2] Eberhard Wolff: Continuous Delivery: Der pragmatische Einstieg, dpunkt.verlag, 2014, ISBN 978-3864902086