Advertorial

Kollaborative API-Entwicklung als wesentlicher Erfolgsfaktor für moderne Applikationsarchitekturen

API first, Microservices und DevOps als Grundlage Cloud-nativer Anwendungsarchitekturen

von Sven Bernhardt

Traditionelle IT-Systemlandschaften bestehen oft aus monolithischen Applikationen, die langwierigen, formalisierten Release-Zyklen unterliegen. Dies steht konträr zu den Zielen vieler moderner IT-Organisationen, die sich als Innovationstreiber verstehen; Eigenschaften wie Agilität, Elastizität und Flexibilität der Anwendungsarchitektur sowie die Optimierung von Kostenstrukturen stehen dabei im Fokus.

Der vorliegende Artikel zeigt am Fallbeispiel eines Webshops, wie die Einführung von Microservices Agilität fördern kann. Dazu gehört auch die Etablierung eines „API first"-Entwicklungsansatzes, um die durch Microservices bereitgestellten Business-Funktionalitäten über intuitiv nutzbare Schnittstellen nach extern anzubieten. Die Etablierung einer Microservices-Architektur kann dabei nur erfolgreich gelingen, wenn die Bereitschaft zur Veränderung auf verschiedenen Ebenen einer Organisation vorhanden ist: DevOps muss aktiv gelebt werden! Weiterhin führt der Artikel in die Grundprinzipien der OMESA Referenzarchitektur ein, mit deren Hilfe moderne Architekturansätze in gewachsene Anwendungsarchitekturen integriert werden können. Anhand eines Fallbeispiels werden Prinzipien wie Microservices und DevOps sowie das „API first"-Vorgehen verdeutlicht und praktisch erläutert, welche Herausforderungen selbst in relativ überschaubaren Szenarien bestehen.

DevOps und Microservices

Moderne Ansätze wie Microservices-Architekturen [1] setzen auf die strikte Trennung unterschiedlicher Business Capabilities, die unabhängig voneinander implementiert, bereitgestellt und betrieben werden können. In der Folge erhöht sich dadurch die Elastizität der Architektur, da einzelne Services individuell und punktuell werden können, um z. B. kurzfristige Lastspitzen abzufangen. Werden Microservices-Architekturen auf einer Cloud-Plattform betrieben, ergeben sich einige Vorteile, da nur für die tatsächliche Nutzung von Ressourcen gezahlt werden muss („Pay-as-you-Go").

Die Kommunikation zwischen einzelnen Microservices erfolgt in der Regel eventbasiert über einen Event Hub. Die Services sind so vollständig voneinander entkoppelt. Änderungen, bedingt durch neue oder geänderte Anforderungen, können also ohne Beeinträchtigung bereits vorhandener Funktionalitäten erfolgen; so weit die Theorie.

In der Praxis müssen Betrieb und Entwicklung enger zusammenrücken. Die konsequente Umsetzung eines DevOps-Ansatzes ist

unabdingbar für den Erfolg von Microservices, um Vorteile wie die Steigerung von Agilität und Effizienz realisieren zu können. Die Einführung eines DevOps-Ansatzes bedingt ein Umdenken bei den bestehenden Denk- und Arbeitsweisen innerhalb der IT-Organisation. Dazu zählen unter anderem

das Aufbrechen getrennter Entwicklungs- und Betriebsbereiche,
die Formung neuer heterogener Teams
oder die Automatisierung großer Teile des Entwicklungsprozesses.

Das neue Mantra, das die neu geformten Teams in diesem Zusammenhang verinnerlichen müssen, lautet: „You build it, you run it!“ – ein Prozess, der nicht von heute auf morgen abgeschlossen ist.

Die erfolgreiche Einführung von Microservices ist also mit nicht zu unterschätzenden organisatorischen Herausforderungen verbunden, insbesondere für gewachsene IT-Organisationen. Aber auch technologisch bzw. architektonisch ergeben sich diverse Herausforderungen – denn in den seltensten Fällen startet ein Unternehmen auf der „grünen Wiese“.

OMESA-Referenzarchitektur für flexible Anwendungsarchitekturen

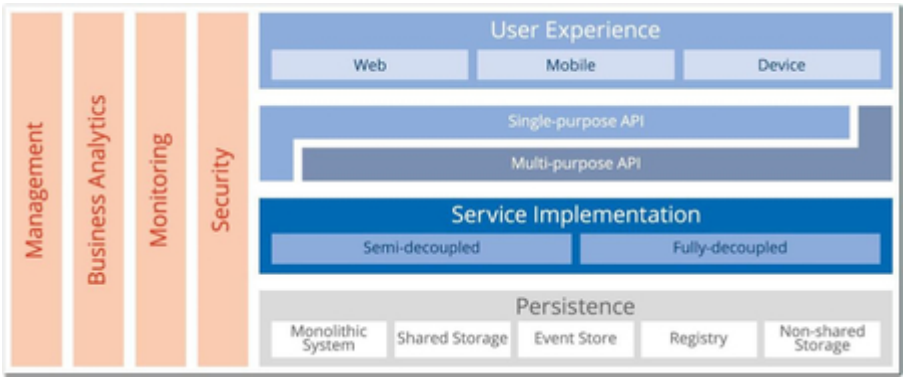

Abb. 1: OMESA-Referenzarchitektur

Das Projekt der Open Modern Enterprise Software Architecture (OMESA) [2] beschäftigt sich mit verschiedenen Fragestellungen; zum Beispiel, wie bewährte architektonische Grundprinzipien und Architekturmuster in modernen Softwarearchitekturen zu verankern sind. Das Ziel ist es, alte und neue Welt sinnvoll miteinander zu kombinieren. Sinnvolle Koexistenz anstelle kompletter Restrukturierung und Refaktorisierung heißt die Devise. Ein solches Vorgehen ist gerade in Bezug auf langjährig gewachsene IT-Systemlandschaften sinnvoll. Eine der zentralen Botschaften von OMESA lautet: „Microservices are no silver bullets!“ OMESA definiert zu diesem Zweck eine Referenzarchitektur sowie ein mehrstufiges Capability Model.

Die in **Abbildung 1** dargestellte Architektur zeigt die zentralen Ebenen, wobei Microservices im Bereich der „Service Implementation“ [3], als so genannte „Fully-decoupled Services“, einzuordnen sind.

APIs – Verbindung von Microservices und UIs

In der OMESA Referenzarchitektur ist die API-Ebene ein grundlegender Baustein moderner Softwarearchitekturen (vgl. **Abbildung 1**). Aber warum sind APIs eigentlich so essenziell?

Um möglichst unabhängig voneinander zu bleiben, interagieren Microservices untereinander hauptsächlich asynchron bzw. eventbasiert. Für die Kommunikation mit der Außenwelt, beispielsweise über Benutzeroberflächen, ist dieser Kommunikationsstil allerdings im Sinne einer guten User Experience (UX) nicht geeignet; synchrone Kommunikationsmechanismen sind zu bevorzugen.

Der in OMESA propagierte zweischichtige API-Ansatz, der Single- und Multi-Purpose APIs unterscheidet, macht die Gesamtarchitektur flexibler und agiler [4]. Multi-Purpose APIs sind allgemeiner, bieten einen breiteren Funktionsumfang und sind somit potenziell wiederverwendbar; Single-Purpose APIs hingegen können auf den Multi-Purpose APIs aufbauen und bilden dabei UI- oder device-spezifische Logik ab [5].

Die API-Ebene dient also vor allem dazu, die Service-Implementierung von der Benutzeroberfläche zu abstrahieren und die von den Services bereitgestellten Funktionalitäten sicher nach außen zu exponieren. „Sicher“ meint hierbei, dass die API-Ebene übergreifende Aspekte wie grundlegende Sicherheitsmechanismen (z. B. Authentifizierung und Autorisierung), Origin-Controls (Cross-Origin Sharing Resources, CORS [6]) oder Threat-Protection-Maßnahmen (z. B. Rate Limits) zentral und konsistent definiert, ohne diese explizit in jedem Service implementieren zu müssen. Das hat den Vorteil, dass sich Backend-Entwickler voll und ganz auf die Implementierung der Geschäftslogik konzentrieren können.

Zwischenfazit der wichtigsten Fakten

- Microservices-Architekturen bringen organisatorische, architektonische sowie technische Herausforderungen mit sich.
- DevOps ist essenziell für den Erfolg von Microservices.
- Intuitive APIs sind wichtig, um Business-Funktionalitäten nach extern anbieten zu können (Etablierung neuer digitaler Economys).

Im zweiten Teil dieses Artikels sollen die angesprochenen Aspekte kurz anhand eines Fallbeispiels näher beleuchtet werden [11]. Die Ideen der OMESA-Referenzarchitektur dienten in diesem Fallbeispiel als architektonische Grundlage für die Implementierung.

Beispielszenario: eine Webshop-Lösung

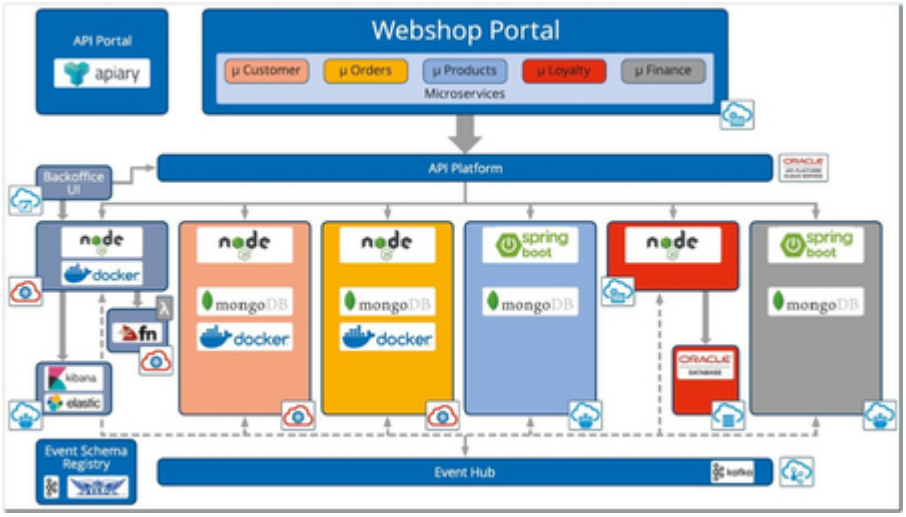

Abb. 2: Gesamtarchitektur der Webshop-Lösung

Das Beispielszenario ist ein fiktiver Webshop, der auf einer Microservices-Architektur basiert und von einem sechsköpfigen Team entwickelt wurde. Jedes Teammitglied war für die Umsetzung einer bestimmten Business Capability, also für jeweils einen Microservice, zuständig. **Abbildung 2** zeigt die Gesamtarchitektur sowie die verwendeten Technologien im Überblick.

Wie der Abbildung entnommen werden kann, sind bei der Umsetzung viele verschiedene Technologien zum Einsatz gekommen, um den Webshop mit seinen sechs Microservices „Logistics“, „Customers“, „Orders“, „Products“, „Loyalty“ und „Finance“ umzusetzen. Als Laufzeitumgebung für die Microservices dienen verschiedene Cloud Services wie beispielsweise ein Managed Kubernetes Cloud Service. Die Interaktion zwischen den Microservices erfolgt eventbasiert über einen Kafka-basierten Event-Hub [7].

Da die Microservices von UIs verwendet werden sollen, müssen REST-APIs exponiert werden, die über ein API Gateway sicher nach außen gestellt werden. Da in einem ersten Schritt nur eine Web UI bedient werden muss, wird auf API-Ebene nicht zwischen Multi- und Single-Purpose APIs unterschieden.

Das Webshop-Portal stellt nur den Rahmen zur Verfügung, der die übrigen microservice-spezifischen UIs einbindet. Dies bedeutet maximale Flexibilität bei Änderungen. Wenn beispielsweise aufgrund technischer Anpassungen ein Deployment des Microservices „Finance" notwendig wird, kann dieses jederzeit durchgeführt werden, ohne die übrigen Services zu beeinträchtigen. Benutzer

können also weiterhin den Produktkatalog durchsuchen oder Bestellungen durchführen; nur im Finance-Bereich käme es kurzfristig zu Einschränkungen.

Die Gesamtarchitektur ist also sehr flexibel aufgebaut. Sie besteht aus Einzelkomponenten, die auf horizontaler Ebene voneinander unabhängig sind. Auf diese Weise kann sehr agil auf sich ändernde Fachanforderungen reagiert werden.

„API first“-Entwicklung

Intuitive APIs sind kritische Erfolgsfaktoren moderner Softwarearchitekturen. Sie sollten einfach zu nutzen, schwer zu missbrauchen sowie wartungsfreundlich und konsistent definiert sein. Kurzum: Gutes API-Design ist wichtig für die Akzeptanz der API-Konsumenten und damit äußerst relevant für die Nutzung einer API. Wie bei der UX für UIs muss man sich also auch hier genauer ansehen, wie die API vom Konsumenten verwendet wird. Deshalb starten wir die Entwicklung nicht etwa mit der Implementierung der Backend-Logik, sondern mit dem Design der API.

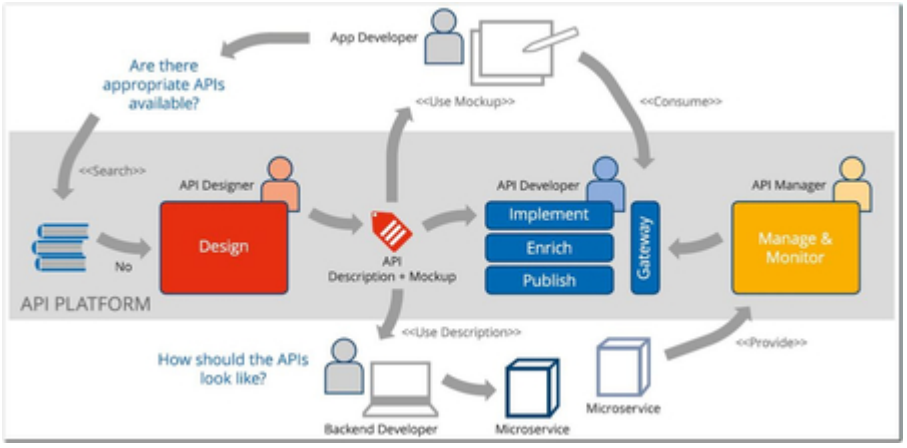

Abb. 3: „API first“-Design-Ansatz

Ein „API first“-Design-Ansatz ist unerlässlich für die Flexibilität und Agilität während des gesamten API-Lebenszyklus. Darüber hinaus ermöglicht API first die kollaborative Zusammenarbeit verschiedener Stakeholder an einer API-Definition und entkoppelt so API-Implementierung, UI- und Backend-Service-Entwicklung. Dieser Zusammenhang wird in **Abbildung 3** dargestellt.

Für die Microservices unseres Webshops wurde also zunächst die API beschrieben. Die Beschreibung einer API kann dabei über verschiedene Dialekte wie Swagger/Open API oder API Blueprint stattfinden. Moderne, cloud-basierte API-Design-Lösungen bieten dabei Funktionalitäten, mit denen APIs direkt nach Fertigstellung der Beschreibung in einer Mock-Variante verwendet werden können.

Wie in **Abbildung 3** dargestellt, erlaubt API first, dass App Developer, API Developer und Backend Developer direkt und völlig unabhängig voneinander mit der Implementierung der Web UI, der API und der Backend-Service-Implementierung starten können. Da die API-Beschreibung so allen Stakeholdern bereits zu einem sehr frühen Zeitpunkt zur Verfügung steht, sind Änderungen an der API einfach und ohne großen Aufwand möglich. Dank der kurzen Feedbackzyklen kommt man schnell zu einer guten API Usability.

Testautomatisierung

Das Thema Testautomatisierung spielt im Kontext von DevOps eine wichtige Rolle. Um Änderungen möglichst schnell produktiv zur Verfügung stellen zu können, ist eine hohe Testabdeckung notwendig. Bei der Entwicklung der Backend-Logik der Microservices werden Unittests geschrieben, die dann zum Build-Zeitpunkt mit Hilfe entsprechender Build Management Tools, etwa Apache Maven, automatisiert ausgeführt werden. In diesem Zusammenhang stellt sich allerdings die Frage, ob und wie es möglich ist, API-Definitionen automatisiert gegen das spezifizierte API-Design zu validieren. Schließlich muss sichergestellt werden, dass sich ein Backend Service konform zur API-Definition verhält.

Solche Tests können beispielsweise mit Hilfe von Dredd [8] automatisiert werden. Dredd ist ein HTTP API Testing Framework, das Tests gegen eine API ausführen kann.

Build- und Delivery-Automatisierung

Der Automatisierungsaspekt wird für das Webshop-Projekt mit einer Build- und Delivery-Plattform wie zum Beispiel Jenkins oder Wercker [9] abgebildet. Hierbei werden in entsprechenden Build Pipelines die Schritte beschrieben, die notwendig sind, um einen Microservice ausgehend von einer Quellcodeänderung möglichst automatisiert auf die Zielumgebung auszurollen. Auf dem Weg zum Rollout werden u. a. die Applikation gebaut, die automatisierten Tests (bspw. Unit- und Dredd-Tests) ausgeführt und die containerisierte Applikation in die Kubernetes-Umgebung deployt.

Fazit

Wie der Artikel zeigt, sind die Herausforderungen bei der Entwicklung von cloud-nativen Anwendungen mannigfaltig. Eine zentrale Rolle spielen der gewählte Architekturstil, APIs, welche die externe Kommunikation abbilden können, und ein konsistenter DevOps-Ansatz.

Anhand des Webshop-Fallbeispiels wurde gezeigt, worauf es bei der Umsetzung moderner, cloud-nativer Applikationen basierend auf einer Microservices-Architektur ankommt. Da die Teammitglieder über fünf verschiedene Länder verteilt waren, konnte man zudem einen ersten Eindruck von den organisatorischen Herausforderungen bekommen, die mit einer Microservices-Implementierung einhergehen.

Key Take-Aways

Referenzmodelle wie die OMESA-Referenzarchitektur sind eine entscheidende Hilfe für die Definition einer konsistenten Gesamtarchitektur.
Die Auswahl des passenden Architekturstils ist entscheidend für den Projekterfolg: „Microservices are no silver bullet!“
Ein Microservices-Ansatz bedingt organisatorische Veränderungen, auf die man sich einlassen muss: „Embrace change!“
Mit Hilfe von Microservices kann die Elastizität einer Anwendungsarchitektur entscheidend gesteigert werden.
Ein konsistenter DevOps-Ansatz ist ein entscheidender Faktor für die Umsetzung einer Microservices-Architektur.
Der „API first“-Ansatz ermöglicht ein effizientes, kollaboratives Entwicklungsvorgehen für die Erstellung intuitiv benutzbarer APIs.

Quellen

[1] James Lewis, Martin Fowler: “Microservices – A Definition of this New Architectural Term”, 2014, https://www.martinfowler.com/articles/microservices.html

[2] OMESA Group: “Open Modern Software Architecture Project”, OMESA Website, 2017, http://omesa.io

[3] OMESA Group: “Capabilities Service Implementation”, OMESA Website, 2017, http://omesa.io/serviceimplementation

[4] OMESA Group: “Capabilities API”, OMESA Website, 2017, http://omesa.io/apilayer/

[5] Sam Newman: “Pattern: Backends For Frontends”, Blog des Autors, 11/2015, https://samnewman.io/patterns/architectural/bff/

[6] Anne van Kesteren: “Cross-Origin Resource Sharing”, W3C Recommendation, 1/2014, https://www.w3.org/TR/cors/

[7] Apache Software Foundation: “Welcome to Apache Avro!”, Projektdokumentation, 2012, https://avro.apache.org

[8] „Dredd – HTTP API Testing Framework“, Projektdokumentation 5/18, http://dredd.readthedocs.io/en/latest/

[9] Oracle + Wercker: „Increase developer velocity …“, Oracle 2018, http://www.wercker.com

[10] Luis Weir: „Is BPM Dead, Long Live Microservices?“, Blog des Autors, 2/2018, http://www.soa4u.co.uk/2018/02/is-bpm-dead-long-live-microservices.html

[11] Lonneke Dikmans, Lucas Jellema, Luis Weir, Guido Schmutz, José Rodrigues und Sven Bernhardt, “Fallbeispiel Webshop”, Github, 4/2018, https://github.com/lucasjellema/soaring-through-the-cloud-native-sequel

---

Sven Bernhardt

ist als führender Integrationsarchitekt bei der OPITZ CONSULTING Deutschland GmbH tätig und verfolgt in dieser Rolle seine Leidenschaft für die Gestaltung und den Aufbau zukunftsorientierter, robuster Unternehmensanwendungen auf der Basis von wegweisenden Technologien. Sven Bernhardt arbeitet in diversen Integrationsprojekten, die sich mit Herausforderungen im Bereich der digitalen Transformation beschäftigen, und ist bei OPITZ CONSULTING insbesondere für das Thema API-Management und moderne Softwarearchitekturen zuständig.

E-Mail: sven.bernhardt(at)opitz-consulting.com

Bildnachweise:

Sven Bernhardt, Github