MASTERARBEIT

ENTWICKLUNG EINER MIGRATIONSSTRATEGIE FÜR LEGACY WEBANWENDUNGEN AUF EINE MODERNE CLOUD-PLATTFORM

ausgeführt an der

am Studiengang

Software Engineering Leadership

Von: Yves Boley

Personenkennzeichen: 1540030003

Graz, am 15.04.2018 .....

Unterschrift

EHRENWÖRTLICHE ERKLÄRUNG

Ich erkläre ehrenwörtlich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen nicht benützt und die benutzten Quellen wörtlich zitiert sowie inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

.....

Unterschrift

KURZFASSUNG

Cloud-Computing ist das Schlagwort der letzten Jahre in der Informationstechnologie schlechthin. Anbieter von Cloud-Lösungen versprechen Einsparungen bei Infrastrukturkosten, eine schnellere Serviceverfügbarkeit, eine bessere Performanz und kürzere Entwicklungszyklen.

Viele Unternehmen reizt deshalb der Umstieg auf eine Cloud-Infrastruktur. Doch mit einem einfachen Umzug der Anwendungen ist es selten getan. Diese Arbeit zeigt auf, wie Softwareanwendungen aufgebaut sein müssen, damit sie aus einer Cloud-Infrastruktur einen optimalen Nutzen ziehen können. Zudem sind auch organisatorische Änderungen nötig, um moderne Cloud-Anwendungen zu entwickeln. Auch diese Änderungen werden besprochen. Häufig stehen Unternehmen vor dem zusätzlichen Problem, dass Anwendungen seit mehreren Jahren in Betrieb, aber technisch veraltet sind. Diese Legacy-Anwendungen sind geprägt durch fehlendes Entwicklungs-Know-how und eine lange Einsatzphase ohne Modernisierung und Restrukturierung. Die Plattformen, die sie nutzen, sind oft veraltet und der technologische Sprung auf eine Cloud-Umgebung deshalb sehr groß. Diese Arbeit erklärt, was Legacy-Software ist, wie sie entsteht, und wie mit ihr verfahren werden kann. Zudem wird das Thema Softwaremigrationen erklärt. Verschiedene Migrationsarten werden vorgestellt, und der exemplarische Ablauf einer Softwaremigration aufgezeigt.

Das Ergebnis der Arbeit ist ein Konzept für Migrationsstrategien von Legacy-Anwendungen. Es wird für ein großes deutsches Versicherungsunternehmen entwickelt, das die Einführung einer Cloud-Infrastruktur plant. Der Umgang mit Altanwendungen, die bereits nicht mehr in die geplante Laufzeitumgebung passen, die jedoch weiterhin benötigt werden, ist derzeit ungeklärt. Er soll mit Hilfe dieser Arbeit festgelegt werden.

ABSTRACT

Cloud-computing is the buzz word of last years in information technology par excellence. Providers of cloud solutions promise savings in infrastructure costs, faster service availability, better performance and shorter development cycles.

Many companies are therefore moving towards a cloud infrastructure. However, with a simple relocation of applications, it is rarely done. This work demonstrates how software applications must be built to gain the most benefit from a cloud infrastructure. In addition, organizational changes are needed to develop modern cloud applications. These changes are also discussed. Often companies face the additional problem that applications have been in operation for several years and that are considered out-of-date. These legacy applications are characterized by a lack of development know-how and a long operational phase without modernization and restructuring. The platforms they use are often outdated and the technological gap to a cloud environment is therefore huge. This thesis explains what legacy software is, how it can be dealt with and how software turns into legacy software. In addition, software migrations are explained in detail. Various types of migrations are presented, and the exemplary sequence of a software migration is shown.

The result of the work is a migration concept for legacy applications, which proposes suitable migration strategies. It is being developed for a large German insurance company planning to roll out a cloud infrastructure. The handling of legacy applications that are no longer compatible with the planned runtime environment is currently still unclear and will be determined with the help of this work.

GLEICHHEITSGRUNDSATZ

Aus Gründen der Lesbarkeit wurde in dieser Arbeit darauf verzichtet, geschlechtsspezifische Formulierungen zu verwenden. Jedoch möchte ich ausdrücklich festhalten, dass die bei Personen verwendeten maskulinen Formen für beide Geschlechter zu verstehen sind.

INHALTSVERZEICHNIS

1 EINLEITUNG

"Software is eating the world"

Im Jahr 2011 schrieb Investor Marc Andreessen (2011) diesen Satz in einem Essay für das Wall Street Journal. In seinem Artikel folgerte Andreessen, dass sich Unternehmen egal welcher Branche gezwungen sehen, zu einem Softwareunternehmen werden zu müssen. Nur so können sie ihre Marktanteile verteidigen und neue Kunden gewinnen. Andreessen nennt hierfür einige Beispiele, bei denen Softwareunternehmen innerhalb der letzten Jahre zu Marktführern in einer Branche wurden. So besitzt Google die größte Plattform für Direktmarketing, und Amazon ist dank des Onlineverkaufs der größte Buchhändler weltweit. Auch Unternehmen, die Software- und Internettechnologie nicht direkt nutzen, um Produktion oder Dienstleistungen zu verkaufen, setzen, wie beispielsweise Wal-Mart, erfolgreich auf eine verstärkte Softwareunterstützung in Logistik und Beschaffung. Auch wenn einige andere Beispiele in Andreessens Artikel heute nicht mehr aktuell sind, zeigt er, dass aufstrebende Internet-Start-ups in sehr kurzer Zeit große Marktanteile erobern können. Sie stellen für etablierte Unternehmen in allen Geschäftsfeldern eine potenzielle Konkurrenz dar. Im Bereich der Versicherungswirtschaft werden diese Unternehmen InsurTechs[1] oder FinTechs genannt. Ein bekanntes Beispiel ist das Unternehmen *friendsurance*[2]. Ihren Erfolg erzielen diese Unternehmen durch den Einsatz von neuester Softwaretechnologie und der Möglichkeit, schnell auf Marktveränderungen und Kundenwünsche reagieren zu können (Startplatz, kein Datum). Besonders die Nutzung von Cloud-Computing ist bei Start-ups sehr hoch, um Abhängigkeiten zur Infrastruktur zu reduzieren und flexibel auf ein sich änderndes Lastverhalten reagieren zu können.

Ein Unternehmen, das Cloud-Computing sehr erfolgreich einsetzt, ist der Video-on-Demand Anbieter *Netflix*. Seinen Ursprung hat das Unternehmen im lokalen Filmverleih per Post. Mit der stetigen Verbreitung des Internets erkannte *Netflix*, dass Filme nicht mehr dem Kunden zugesandt werden müssen. Kunden können viel einfacher und flexibler über das Internet auf die Filme zugreifen. Für die Bereitstellung dieser Onlinedienste stützt sich Netflix ebenfalls auf Cloud-Computing (Izrailevsky, 2016). Mit Hilfe von Data Analytics war *Netflix* in der Lage herauszufinden, welche Filme die Kunden bevorzugen. Basierend auf den gesammelten Daten begann *Netflix* eigenständig Filme und Serien zu produzieren und sich so von Konkurrenten abheben. Die Mentalität von *Netflix* lässt sich am besten mit den Worten von Jeetu Patel (2016) beschreiben, der in

[1] Der Begriff InsurTech beschreibt den Einsatz moderner Technologie im Umfeld von Versicherungsunternehmen. Häufig wird der Begriff zur Beschreibung von Start-up Unternehmen genutzt, die durch den Technologieeinsatz Marktanteile auf dem Versicherungsmarkt erobern wollen.

[2] https://www.friendsurance.de/

einem Artikel Andreessens Aussage reflektiert und die Denkweise von Unternehmen wie Netflix zusammenfasst mit: „*It's not about what we can do now, but better; it's about what can we do now that was simply not possible until today*".

Etablierte Versicherungsunternehmen haben aus diesen Beispielen gelernt und die Gefahr durch InsurTechs erkannt. Eine große deutsche Versicherung stellt sich dieser Herausforderung mit einer Digitalisierungsstrategie und dem massiven Einsatz finanzieller Mittel. Die zwei Hauptaspekte der Strategie sind:

1. Die Digitalisierung aller Prozesse, bei denen dies sinnvoll ist und die digital einen Kundennutzen stiften, um die Kundenzufriedenheit zu erhöhen und Durchlaufzeiten und Kosten zu senken.
2. Die Modernisierung der Laufzeitumgebung und der Arbeitsweise in der Softwareentwicklung in erster Linie für Webanwendungen, die von Endkunden, Vertretern und Maklern genutzt werden.

Im Rahmen des zweiten genannten Aspektes soll die eigene IT-Infrastruktur für Webanwendungen abgelöst werden. Die bestehenden Anwendungen sollen im Rechenzentrum eines großen Cloud-Computing Anbieters eine neue Heimat finden. Offen ist hierbei der Umgang mit Altanwendungen, die noch in Nutzung sind, aber nicht mehr dem Stand der Technik und den internen Programmierstandards entsprechen. Sie können deshalb nicht ohne Anpassung in der neuen Umgebung betrieben werden. Das Ziel dieser Arbeit ist es, zu untersuchen, mit welchen Maßnahmen diese Anwendungen auf eine moderne Cloud-Umgebung migriert werden können, um sie dort dauerhaft zu betreiben.

Der erste Teil dieser Arbeit legt die technischen Grundlagen für das Verständnis dieses komplexen Umfeldes. Nach einigen grundlegenden Definitionen in Kapitel 2 gibt das darauffolgende Kapitel 3 einen Überblick über das Thema *Cloud-Computing*. Der Fokus liegt hierbei auf den Eigenschaften, die eine Cloud-Umgebung von herkömmlichen Laufzeitumgebungen unterscheidet. Zudem werden weiterführende Themen angesprochen, die oft mit der Einführung einer Cloud-Infrastruktur einhergehen. Dies sind unter anderem der Trend zu einer Microservices-Architektur (Abschnitt 3.6) und die Einführung eines DevOps-Modells (Abschnitt 3.7). Das 4. Kapitel thematisiert *Legacy-Anwendungen*, also Software, die als veraltet und damit oft als problembehaftet gilt. Neben einer Definition von Legacy-Anwendungen wird auf die Entstehung und die Probleme von Legacy-Anwendungen eingegangen. Zum Abschluss gibt das Kapitel einen Überblick, wie mit Legacy-Software umgegangen werden kann. Das folgende Kapitel 5 behandelt das Thema *Softwaremigrationen*. Es gibt einen Überblick über die Gründe, die zu einer Softwaremigration führen, und über die verschiedenen Arten, wie Software in eine neue Umgebung überführt werden kann. Es schließt mit einem exemplarischen Ablauf einer Softwaremigration. Das 6. Kapitel stellt die Online IT-Systeme der *Allianz Deutschland AG* vor. Sie sind geprägt durch eine sehr heterogene Erscheinung mit Anwendungen, die teils seit mehr 15 Jahren im Einsatz sind. Die Historie und die Umstellungspläne des Unternehmens auf eine Cloud-Infrastruktur werden ebenfalls im Kapitel behandelt. Das 7. Kapitel beschreibt die Entwicklung eines Migrationskonzepts, das passende Migrationsverfahren für die Altanwendungen vorschlägt. Das Praxisbeispiel ist eine Plattform mit

Legacy Online Anwendungen der *Allianz Deutschland AG*. Ziel ist es, die Altanwendungen - nach Möglichkeit - auf einer modernen Cloud-Umgebung zu betreiben. Die Arbeit schließt mit einem Fazit und einem Ausblick.

2 GRUNDLEGENDE DEFINITIONEN

Dieses Kapitel dient dazu, einige Themen abzugrenzen, die innerhalb dieser Arbeit aufgegriffen werden oder als Voraussetzung für spätere Kapitel dienen.

2.1 Webanwendungen

Diese Arbeit bezieht sich in der Hauptsache auf Webanwendungen. Kappel, Pröll, Reich und Retschitzegger (2003) definieren eine Webanwendung folgendermaßen:

> *„Eine Web-Anwendung ist ein Softwaresystem, das auf Spezifikationen des World Wide Web Consortium (W3C) beruht und Web-spezifische Ressourcen wie Inhalte und Dienste bereitstellt, die über eine Benutzerschnittstelle, den Web-Browser, verwendet werden“*

Die wichtigsten Eigenschaften von Webanwendungen sind nach Ndegwa (2016) und Plate (2017):

1. Sie sind nach dem Client-Server-Modell aufgebaut. Ein Server stellt Funktionen in einem Netzwerk bereit, die von verschiedenen Clients angefragt werden können. Wegen der Trennung von Client und Server durch das Netzwerk können Webanwendungen plattformübergreifend arbeiten.

2. Webanwendungen benutzen das HTTP-Protokoll zur Datenübertragung und HTML zur Darstellung von Informationen. Ergänzend können JavaScript und CSS (Cascading Style Sheets) verwendet werden.

3. Webanwendungen sind grundsätzlich zustandslos. Das bedeutet, dass die Verbindung zwischen Client und Server nicht dauerhaft besteht.

4. Clientseitig kommt ein Browser zur Kommunikation mit dem Server zum Einsatz.

5. Der oder die Server generieren die Webseiten dynamisch je nach Anfrage und senden sie an den Client.

2.2 Webservices und Serviceorientierte Architektur

Im Gegensatz zu Webanwendungen, bei denen Menschen über einen Client Anwendungen nutzen, dienen Webservices der Maschine-zu-Maschine Kommunikation. Nach Takai (2017) kommunizieren die Partner auf Grundlage eines Vertrags, der Regeln für die Ein- und Ausgabe festlegt. Ein typisches Beispiel hierfür ist die WSDL-Datei (Web Services Description Language), die

einen SOAP-Service beschreibt. Ein wichtiger Vorteil von Services ist die Kommunikation über ein Netzwerk. Durch diese Trennung können die Partner in unterschiedlichen Programmiersprachen implementiert sein. Es ist lediglich von Belang, dass sie die gleiche Sprache zur Kommunikation verstehen. Üblich sind in der Praxis insbesondere zwei Technologien: Representational State Transfer (REST) und Simple Object Access Protocol (SOAP).

Webservices werden oft mit dem Architekturmodell der Serviceorientierten Architektur (SOA) verbunden. Es kam Anfang der 2000er Jahre auf und verbreitet sich seitdem (Takai, 2017). Thomas Erl (2008) hat acht Entwurfsprinzipien zusammengestellt, die für eine Serviceorientierte Architektur gelten:

1. Es werden Serviceverträge genutzt, die Aufrufe, Eingabe und Rückgabewerte sowie Fehlerzustände beschreiben. Auf diese Festlegungen können sich die Vertragspartner verlassen.
2. Lose Kopplung: Services sollen so gut es geht voneinander unabhängig sein. Jeder Kommunikationspartner muss seine Abhängigkeiten zu anderen Partnern kennen. Optimalerweise werden diese zentral innerhalb eines Service verwaltet.
3. Die Webservices werden weitgehend abstrahiert. Aufrufende kennen nur die Schnittstelle und nicht die interne Funktionsweise eines Service.
4. Webservices können wiederverwendet werden und unterstützen mehrere aufrufende Partner.
5. Webservices sind autonom in Entwicklung und Betrieb. Sie sollen nicht voneinander abhängig sein.
6. Webservices sind zustandslos (stateless). Bei einer Anfrage muss nichts über vorherige Anfragen bekannt sein.
7. Webservices können automatisch entdeckt werden. Dies kann mit Hilfe eines zentralen Servicekatalogs geschehen und trägt zur Unabhängigkeit zwischen den Services bei.
8. Services lassen sich wie einzelne Komponenten nutzen und austauschen. Dadurch können komplexe Anwendungen aufgebaut werden.

Eine Serviceorientierte Architektur ist nach Takai (2017) nicht unumstritten. Herausforderungen sind unter anderem das Management aller aufrufenden Partner sowie ein gemeinsames Datenmodell, dass viele Unternehmen mit dem Umstieg auf ein Serviceorientierte Architektur einführen wollten. Nach Takai ist dieses Vorhaben oft an der Komplexität und Heterogenität großer Unternehmen gescheitert.

Webservices können auch in einem anderen Architekturmodell eingesetzt werden: als Microservice. Dieses Modell ist in Abschnitt 3.6 beschrieben.

2.3 Laufzeitumgebung

Eine Laufzeitumgebung stellt für Softwareanwendungen den Rahmen der möglichen Sprachmittel dar. Die Laufzeitumgebung muss bei der Ausführung einer Softwareanwendung vorhanden sein. Die Softwareanwendung kann dann auf die Funktionen der Laufzeitumgebung zugreifen. Das sind die elementaren Bestandteile einer Programmiersprache, wie Datentypen, Basisfunktionen und deren Verhalten.

Eine Laufzeitumgebung wird oft auch Ablaufumgebung oder auf Englisch Runtime genannt. Bekannte Beispiele dafür sind das *Java Runtime Environment* (JRE) oder *.NET* von Microsoft. Bei Online Anwendungen wird die Laufzeitumgebung durch einen Application Server bereitgestellt, auf dem die Softwareanwendung ausgeführt wird.

2.4 Softwarequalität

Der Begriff *Softwarequalität* steht für alle Eigenschaften eines Softwareproduktes in Bezug auf dessen Eignung, um den vorgesehenen Zweck zu erfüllen. Die ISO-Norm 25010:2011 (früher 9126-1) beschreibt Qualitätskriterien für Softwaresysteme. Diese sollen im Folgenden kurz vorgestellt werden. Nach Takai (2017) unterteilt die ISO-Norm die Qualitätskriterien in zwei Bereiche: Die Betriebsqualität, bezogen auf die gewünschte Qualität bei der Nutzung des Systems durch Anwender, und die Produktqualität, die gewünschte Eigenschaften einer Anwendung beschreibt. Diese Qualitätsmerkmale der zwei Themenfelder sind:

Betriebsqualität *Gewünschte Qualität bei der Interaktion zwischen Anwendern und dem System*	**Produktqualität** *Statische und dynamische Eigenschaften der Software*
Effektivität Effizienz Zufriedenheit Risikoverringerung Anwendungsbereich	Funktionalität Effizienz Kompatibilität Bedienbarkeit Zuverlässigkeit Sicherheit Wartbarkeit Portierbarkeit

Tabelle 1: Qualitätsmerkmale nach ISO-Norm 25010:2011

Im Laufe dieser Arbeit wird insbesondere auf die Produktqualität von Softwaresystemen Bezug genommen. Das wichtigste Qualitätsmerkmal für diese Arbeit ist die *Wartbarkeit*, als das Merkmal, das insbesondere für Anwendungen interessant ist, die eine lange Lebensdauer haben und an wechselnde Rahmenbedingungen angepasst werden müssen. Die anderen Qualitätsmerkmale sind ebenfalls sehr wichtig für Anwendungen, spielen aber bei Softwaremigrationen keine so dominante Rolle wie die Wartbarkeit. Der Begriff *Wartbarkeit* beschreibt, wie einfach ein System an geänderte Anforderungen angepasst werden kann. Er wird von Takai (2017) unterteilt in:

1. Konzeptionelle Integrität: Die Eignung der Technik für die zu erledigende Arbeit
2. Konsistenz: Die Durchgängigkeit von Entwurfsentscheidungen in Architektur, Design und Technologie
3. Testbarkeit: Der Aufwand, mit dem Fehler in einem System gefunden werden
4. Analysierbarkeit: Die Vorhersagbarkeit für Änderungen an einem System
5. Änderbarkeit: Die Geschwindigkeit, mit der sich Änderungen an einem System vornehmen lassen

2.5 Software-Tests

Innerhalb dieser Arbeit wird auf verschiedene Arten von Software-Tests eingegangen. Diese werden in den folgenden Abschnitten kurz definiert und erklärt.

System- und Abnahmetests sind nach (Takai, 2017) die wichtigsten Tests, die eine Anwendung durchläuft. Bei ihnen wird geprüft, ob ein Softwareprodukt die fachlichen Anforderungen erfüllt, die an es gestellt werden. Hierfür werden die Anforderungen auf Anwendungsfälle heruntergebrochen, einzeln getestet und protokolliert. Abnahmetests, die am Ende eines Projektes stehen, sind in vielen Entwicklungsprodukten ein wichtiger Meilenstein.

Unit Tests dienen der Prüfung einzelner Software-Komponenten. Nach Michael Feathers (2011) ist dies bei objektorientiertem Code eine Klasse oder bei prozeduralem Code eine Funktion. Die Prüfung dieser Komponenten erfolgt isoliert. Die Vorteile dieser Testmethoden sind eine einfachere Fehlerlokalisierung und eine geringere Ausführungsdauer. Durch den begrenzten Kontext, den ein Testfall abdeckt, werden Fehler nach Feathers schnell lokalisiert. Durch die kurze Ausführungsdauer können die Testfälle häufig durchlaufen werden, zum Beispiel bei jedem Build. Damit erhalten Entwickler direkt eine Rückmeldung, ob eine Änderung Nebeneffekte hat und können die Fehler direkt eingrenzen und beheben. Ein letzter Vorteil, den Feathers (2011) nennt, ist die Möglichkeit, mit Hilfe einer Abdeckungsanalyse zu erkennen, welche Teile des Quellcodes durch einen Unit-Test abgedeckt werden und bei welchen Teilen Nachholbedarf besteht.

Regressionstests dienen nach Feathers (2011) dazu, die korrekte Arbeitsweise einer Anwendung nach einer durchgeführten Änderung sicherzustellen. Sie werden üblicherweise durch ein

eigenes Team oder automatisiert durchgeführt. Die Grundlage von Regressionstests sind vorangegangene Durchläufe, die ein erwartbares Verhalten der Anwendung zeigen. Dadurch können Regressionstests sicherstellen, dass eine Änderung keine Seiteneffekte hat. Diese Tests dienen jedoch nicht dazu, die korrekte Funktion der Änderung zu prüfen. In der Praxis werden Regressionstests oft durch Oberflächen-Test-Frameworks wie Selenium durchgeführt (Schilli, 2006). Sie prüfen damit die Funktionen, die dem Endanwender zur Verfügung stehen.

Neben den genannten Testformen gibt es weitere, die speziellen Zwecken dienen. Dies sind beispielsweise Performanz- und Sicherheitstests. Diese werden in Abschnitt 5.4 - Ablauf einer Softwaremigration in kurzer Form beschrieben.

3 CLOUD-COMPUTING

Die grundlegenden Bestandteile und Eigenschaften von Cloud-Computing werden in diesem Kapitel beschrieben, um die wichtigsten Unterschiede zu herkömmlichen Laufzeitumgebungen aufzeigen zu können.

3.1 Definition und Historie

In der Literatur finden sich viele verschiedene Definitionen für den Begriff *Cloud-Computing*. Diese enthalten viele Gemeinsamkeiten. Eine genaue Definition für den Begriff scheint es jedoch nicht zu geben. Grance und Mell (2011) vom US-amerikanischen *National Institute of Standards and Technology* (NIST) definieren *Cloud-Computing* als die Verwendung von verschiedenen IT-Dienstleistungen über ein Netzwerk, wenn sie die folgenden Eigenschaften erfüllt:

1. Nutzung gemeinsamer physischer Ressourcen
2. Unverzügliche Anpassungsfähigkeit durch dynamische Skalierung
3. Selbstbedienung nach Bedarf
4. Umfassender Netzwerkzugriff
5. Service Monitoring - Überwachung der Ressourcenverwendung

Lewis Cunningham (2008) definiert *Cloud-Computing* weniger technisch mit dem Satz:

> *Cloud computing is using the internet to access someone else's software running on someone else's hardware in someone else's data center while paying only for what you use.*

Cloud-Computing umfasst also die Nutzung verbundener Rechner durch mehrere Anwendungen, auf die Anwender über das Internet zugreifen. Die Anwendungen können ihren Ressourcenverbrauch dynamisch ausdehnen, wenn die Zugriffslast dies erfordert. Das Rechenzentrum wird meist nicht von dem Unternehmen betrieben, das die Anwendungen bereitstellt. Die Abrechnung dieser Dienstleistung erfolgt in der Regel über das *pay-per-use* Prinzip, dass Cunningham in seiner Definition nennt. Cloud-Lösungen können unter anderem durch ihren Auslagerungs- und Abstrahierungsgrad unterschiedliche Ausprägungen annehmen. Diese werden in den folgenden Abschnitten 3.2 und 3.3 beschrieben.

Die Fähigkeit, schnell auf eine steigende Nutzerzahl zu reagieren und mehr Ressourcen zur Verfügung zu stellen, um Nutzeranfragen abzuarbeiten, wird mit dem Begriff *Skalierbarkeit* bezeichnet. Es wird zwischen vertikaler und horizontaler Skalierbarkeit unterschieden. Horizontale Skalierung bedeutet nach Haselmann, Hoeren und Vossen (2012) eine Verwendung von mehreren Knoten. Die eingehenden Anfragen werden beispielsweise auf mehrere Rechner verteilt. Bei der vertikalen Skalierbarkeit werden die Ressourcen pro Knoten erhöht, zum Beispiel, indem Server

durch leistungsstärkere Modelle ersetzt werden. Eine hohe Skalierbarkeit bedeutet in der Regel die Bereitstellung einer gut ausgebauten Rechnerinfrastruktur. Dies verursacht jedoch auch hohe Kosten, insbesondere durch Kapazitäten, die für Hochlast-Situationen zur Verfügung gestellt werden, die möglicherweise jedoch nie oder nur selten genutzt werden (Takai, 2017).

Cloud-Lösungen versuchen, eine Infrastruktur bestmöglich auszulasten. Durch Anwendungen von verschiedenen Unternehmen, möglicherweise aus verschiedenen Zeitzonen und mit unterschiedlichen Geschäftszwecken, entsteht ein sehr heterogenes Bild des Ressourcenbedarfs. Deshalb erfolgt die Bereitstellung der Cloud-Umgebung meist durch einen großen Dienstleister. So ist es möglich, die vorhandenen Ressourcen über viele Anwendungen zu verteilen und dynamisch verfügbar zu machen. Wichtige technische Eigenschaften, die Anwendungen erfüllen müssen, damit sie in einer Cloud-Umgebung betrieben werden können, werden in Abschnitt 3.5 behandelt.

Viele Eigenschaften, die das *Cloud-Computing* definieren, sind keine neuen Erfindungen, sondern bereits seit vielen Jahren bekannt. Verteilte Systeme, die sich Aufgaben teilen und parallel bearbeiten, damit die Verarbeitungsgeschwindigkeit steigt, sind seit den 1960er Jahren im Einsatz. Haselmann et al. (2012) unterscheiden in *Cluster* und *Grid-Computing*, zwei Systemmodelle, die sich in ihrer Zusammensetzung unterscheiden. Laut den Autoren bestehen Cluster-Systeme aus vielen gleichartigen Rechnern und sind homogen aufgebaut. Grid-Systeme dagegen sind heterogen aufgebaut und enthalten verschiedene Rechnersysteme, die spezielle Aufgaben übernehmen können. Grid-Systeme weisen die Besonderheit auf, dass sich einzelne Rechner aus- und einklinken können. Moderne Cloud-Systeme teilen gleich mehrere Eigenschaften dieser Modelle. Um den Betrieb einfach zu gestalten, setzen die Anbieter auf eine weitgehend homogene Hardware. Innerhalb der Cloud können sich jedoch einzelne Instanzen zu- und abschalten, ohne die Funktionsweise des Gesamtsystems zu gefährden.

Ein weiterer Kernaspekt des Cloud-Computings, das Outsourcing, also die Abgabe von Aufgaben an einen externen Dienstleister, ist nach Haselmann et al. (2012) seit den 1980er Jahren in der Informationstechnologie verbreitet. Die Auslagerung von IT-Funktionen wurde durch das Aufkommen von Hochgeschwindigkeitsnetzwerken ermöglicht. Dies führte zu einem neuen Geschäftsmodell: dem Application-Service-Provider. Diese Unternehmen stellten Anwendungen für Kunden zur Verfügung. Für jeden Kunden wurde dabei eine dedizierte Hardware beim Application-Service-Provider aufgebaut, um die Kunden untereinander zu trennen. Der Kunde hatte Zugriff auf die Anwendung über Netzwerke oder das Internet. Er bezahlte die Lizenzgebühren und Wartungskosten an den Application-Service-Provider und musste sich dadurch nicht mehr selbst um Hard- und Software kümmern. Das Application-Service-Provider-Modell scheiterte unter anderem daran, dass zu viel Hardware vorgehalten werden musste und so die Provider die geplanten Skaleneffekte nicht erreichen konnten. Eine gemeinsame Nutzung der Ressourcen durch mehrere Kunden, inklusive der dafür notwendigen Abgrenzung, wie sie das *Cloud-Computing* vorsieht, war damals nicht realisierbar. Dennoch sehen Haselmann et al. (2012) einen Einfluss der Application-Service-Provider auf das *Cloud-Computing*.

Einen weiteren Einfluss auf das moderne *Cloud-Computing* sehen Haselmann et al. (2012) im *Utility-Computing*, dass John McCarthy 1961 vorstellte. In seiner Vision sprach er von Rechenleistung, die wie das Strom- oder Telefonnetz verteilt wird und von überall je nach Bedarf genutzt werden kann. Cloud-Computing, wie es heute existiert, kommt dieser Vision sehr nahe. Einfluss hatte McCarthys Vision auch auf das heute Abrechnungsverfahren, bei dem meist nur die tatsächlich genutzten Ressourcen bezahlt werden müssen. Die Messung erfolgt über die benötigte Speicher- oder Rechnerkapazität oder über die Aufrufzahlen. Hierfür ist ein umfangreiches Service Monitoring notwendig, wie es auch in der NIST-Definition von Grance und Mell (2011) genannt wird.

3.2 Cloud-Modelle

Cloud-Lösungen können nach Grance und Mell (2011) durch den Auslagerungsgrad unterschieden werden. Es gibt hierbei folgende Kategorien:

1. Private Cloud
2. Öffentliche Cloud
3. Hybride Cloud
4. Community Cloud

Bei einer *privaten Cloud* wird die Rechnerinfrastruktur vom nutzenden Unternehmen beschafft und betrieben. Sie dient in diesem Fall nur einem Unternehmen. Die Aufteilung der Ressourcen wird nur innerhalb der Unternehmens-IT durchgeführt. Der Betrieb der Cloud ist jedoch nicht der eigentliche Unternehmenszweck des Unternehmens. Private Cloud-Umgebungen werden häufig mit Angeboten wie *VMware vSphere* oder *OpenStack* realisiert. Gegenüber altbekannten Rechenzentren zeichnet sich die Private Cloud insbesondere durch eine gemeinsame und flexible Ressourcennutzung aller Anwendungen untereinander aus, die sich an den aktuellen Bedarf der einzelnen Anwendungen anpasst.

Das Gegenteil hierzu ist eine *öffentlichen Cloud-Lösung*. Betreiber der Cloud-Infrastruktur ist ein spezialisiertes Unternehmen, dass diese Leistung anderen Unternehmen zur Verfügung stellt. Die vorhandenen Ressourcen werden dabei über alle Kunden verteilt, um Hochlastsituationen bei einzelnen Kunden dynamisch bedienen zu können. Große Anbieter für solche Kundenlösungen sind *Microsoft*, *Amazon Webservices* und *IBM* (Vaske, 2016). Die Abrechnung erfolgt zumeist nur nach dem tatsächlich genutzten Bedarf des Kunden. Der Kunde trägt dadurch kein Kostenrisiko mehr bei der Bereitstellung von Rechnerkapazität.

Eine Mischform aus beiden genannten Cloud-Formen ist die *hybride Cloud*. Hierbei nutzen Unternehmen öffentliche Cloud-Angebote, oft weil diese billiger sind. Für kritische Anwendungen, wie beispielsweise die Datenspeicherung, betreiben sie eine kleine, private Rechnerinfrastruktur.

Ein Grund für diese Entscheidung können zum Beispiel nationale Datenschutzgesetze sein, die eine Datenspeicherung innerhalb eines oder mehrerer Länder vorschreiben.

Die vierte Form der Cloud ist die sogenannte *Community Cloud*. Sie kommt in Unternehmen sehr selten zum Einsatz, wird dagegen häufig von Open Source Projekten genutzt. Der Betreiber ist in diesem Fall meist eine Open Source Stiftung, wie die Apache Software Foundation, die die Rechenleistung offenen, gemeinnützigen Projekten kostenfrei zur Verfügung stellt.

3.3 Abstrahierungsgrad

Neben dem Auslagerungsgrad unterscheidet Hoffmann (2016) Cloud-Lösungen durch den Bereitstellungsgrad, den ein Dienstleister übernehmen kann. *Infrastructure-as-a-service* ist der geringste Bereitstellungsgrad, den Cloud-Betreiber anbieten. Hierbei besteht ihr Angebot aus der Hardwareinfrastruktur und meist aus einem installierten Betriebssystem. Der Kunde muss sich um alle weiteren Funktionen selbst kümmern, ist aber frei in der Wahl der Software.

Die nächste höhere Stufe wird *Platform-as-a-service* genannt. Neben Hardware und Betriebssystem erhält der Kunde auch eine Laufzeitumgebung. Ein Beispiel hierfür wäre ein Server mit einem installierten Unix Betriebssystem und einem J2EE (*Java Platform Enterprise Edition*) Application Server. Zusätzlich kann der Cloud-Anbieter neben der Bereitstellung auch festgelegte Support- und Wartungsaufgaben erbringen. Der Kunde steuert in diesem Fall nur noch die eigene Anwendung bei.

Wenn der Cloud-Anbieter neben Hardware, Betriebssystem und Laufzeitumgebung dem Kunden auch eine Anwendung zur Verfügung stellt, wird dieses Angebot *Service-as-a-service* genannt. Der Kunde kann dieses Angebot quasi ad hoc nutzen. Betrieb und Wartung für alle Komponenten werden durch den Service-Anbieter erbracht. Haselmann et al. (2012) haben die verschiedenen Auslagerungsgrade in der nachfolgenden Abbildung 1 veranschaulicht.

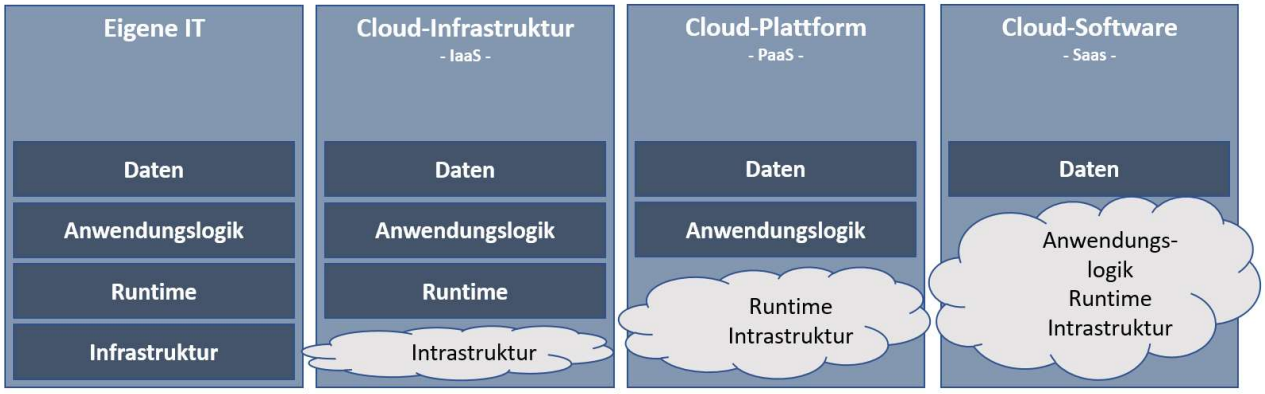

Abbildung 1: Entwicklungsoptionen der Unternehmens-IT mit Cloud-Diensten nach Haselmann et al. (2012)

3.4 Virtualisierung und Container

Nach Haselmann et al. (2012) beschreibt der Begriff *Virtualisierung* die Abbildung logischer Ressourcen auf physische Ressourcen (Hardware). Der Nutzer eines Systems erhält Zugriff auf logische Ressourcen, die sich von den physischen Ressourcen unterscheiden können. Virtualisierung kann als eine Schicht zwischen der eigentlichen Hardware und den Anwendungen und Betriebssystemen verstanden werden. Haselmann et al. (2012) visualisieren das, wie in Abbildung 2 gezeigt.

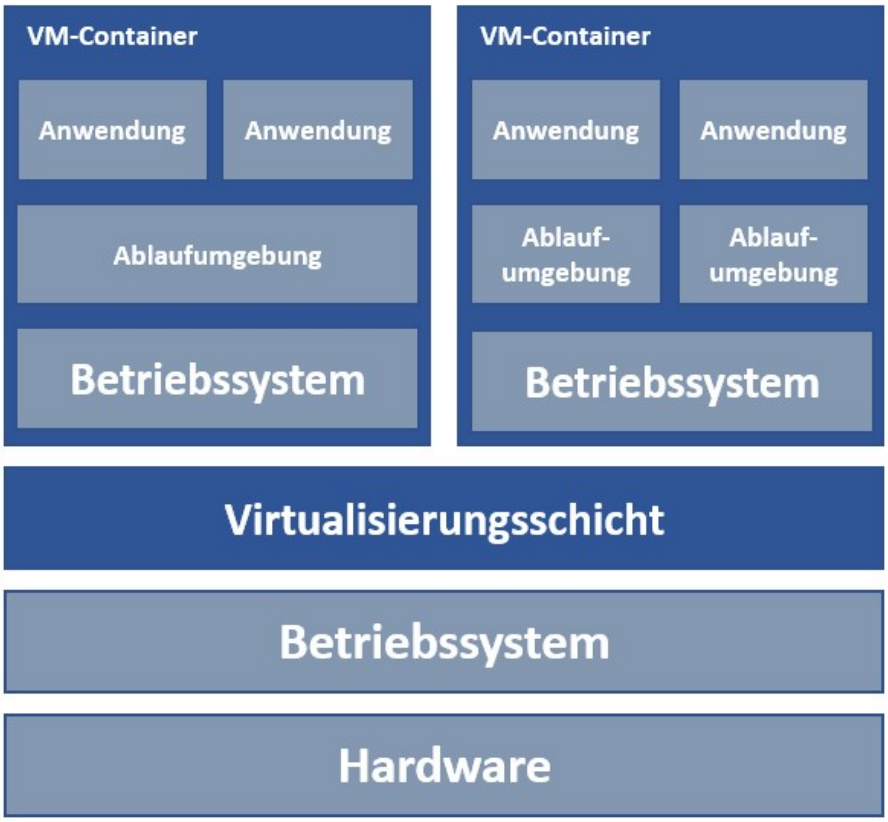

Abbildung 2: Möglicher Aufbau einer virtualisierten Infrastruktur nach Haselmann et al. (2012, S. 18)

Virtualisierung kann mehrere (heterogene) Systeme zu einem logischen System zusammenzuschalten. Zum Einsatz kommt diese Technik oft bei Speichermedien, die Nutzern als eine singuläre Einheit dargestellt werden. Virtualisierung wird jedoch auch genutzt, um Ressourcen für mehrere Anwender aufzuteilen und die getrennten Bereiche voneinander abzuschotten (wie in Abbildung 2 dargestellt). Die einzelnen Anwender agieren in einem eigenen Bereich, mit einem eigenen Betriebssystem. Diese Bereiche werden virtuelle *Container* genannt. Durch die Abschottung zwischen den Containern kann zudem vermieden werden, dass Probleme innerhalb eines Containers Auswirkungen auf andere Container haben. Der Zugriff auf die Hardware erfolgt über eine

Virtualisierungsschicht, die den Zugriff reglementiert und steuert. Sie abstrahiert die vorhandene Hardware, sodass aus einem physikalischen Server mehrere virtuelle werden.

Container, wie sie in Abbildung 2 dargestellt sind, werden seit mehreren Jahren in Rechenzentren eingesetzt. Sie enthalten ein Betriebssystem, mehrere Laufzeitumgebungen sowie mehrere Anwendungen. Ein Beispiel für eine Virtualisierungslösung für Rechenzentren ist das Produkt *vSphere* des Herstellers *VMware Incorporated*.

Mit Cloud-Computing wird auch eine neue Art der Containerisierung verbunden. Die Container enthalten eine Laufzeitumgebung, die Applikation, alle dazugehörigen Abhängigkeiten und Konfigurationen (Haselmann, Hoeren, & Vossen, 2012). Die Virtualisierung erfolgt hierbei auf einer anderen Ebene. Jede Anwendung erhält einen eigenen Container mit all ihren Abhängigkeiten. Das Betriebssystem liegt außerhalb der Container und ist pro Hardware nur einmal vorhanden. Dies ist in Abbildung 3 schematisch dargestellt:

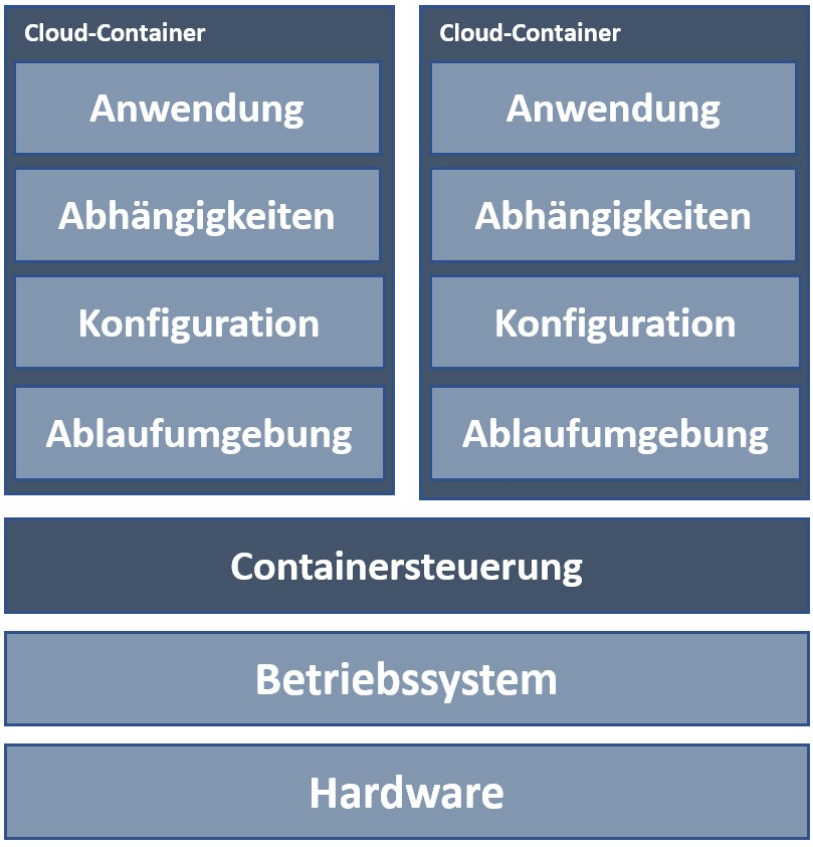

Abbildung 3: Schematischer Aufbau einer virtualisierten Cloud-Infrastruktur

In ihrem Lebenszyklus wird die Anwendung immer in diesem Container betrieben. Durch dieses Vorgehen ergeben sich gleich mehrere Vorteile. Probleme durch unterschiedlich bestückte Ab-

laufumgebungen werden umgangen. Sie können entstehen, wenn auf einer Test- und Produktionsumgebung unterschiedliche Softwareversionen oder Hardwarekomponenten eingesetzt werden, beispielsweise Server mit unterschiedlichen Versionsständen.

Zudem können auch innerhalb eines physikalischen Servers mehrere verschiedene Laufzeitumgebungen genutzt werden, ohne mehrere Betriebssysteme zu betreuen. Als Beispiel können Container mit einem *IBM WebSphere Application Server* neben Container mit einem *Apache Tomcat Server* genutzt werden, ohne Nebeneffekte zu erwarten. Ein weiterer Vorteil des Einsatzes von Containern auf Anwendungsebene ist die gesteigerte Fehlertoleranz. Probleme innerhalb eines Containers beeinträchtigen andere Container nicht. Beim Containermodell, das in Rechenzentren bislang verwendet wird (siehe Abbildung 2 auf Seite 13), können sich Probleme in einer Anwendung schnell auf die Laufzeitumgebung, sprich den gemeinsam genutzten Application Server auswirken.

Eine wichtige Eigenschaft von Cloud-Systemen ist die hohe Skalierbarkeit (beschrieben im Abschnitt 3.1 auf Seite 9). Cloudanwendungen werden in der Regel horizontal skaliert. Dies geschieht, indem je nach Last Anwendungscontainer hoch- oder heruntergefahren werden. Die Anzahl der Anwendungsinstanzen wird also dynamisch reguliert. Dies ist eine Hauptaufgabe der Containersteuerung. Sie steuert den Lebenszyklus der Container und verteilt die eingehenden Anfragen gleichmäßig auf die Container.

Der Einsatz beider vorgestellter Virtualisierungsmodelle schließt sich nicht gegenseitig aus. Nach Maier und Rubens (2015) werden beide Verfahren gleichzeitig genutzt, um durch die zusätzliche Isolationsschicht eine weitere Trennung der Komponenten zu erreichen. Die folgende Abbildung 4 zeigt, wie diese mehrfache Virtualisierung aufgebaut sein kann:

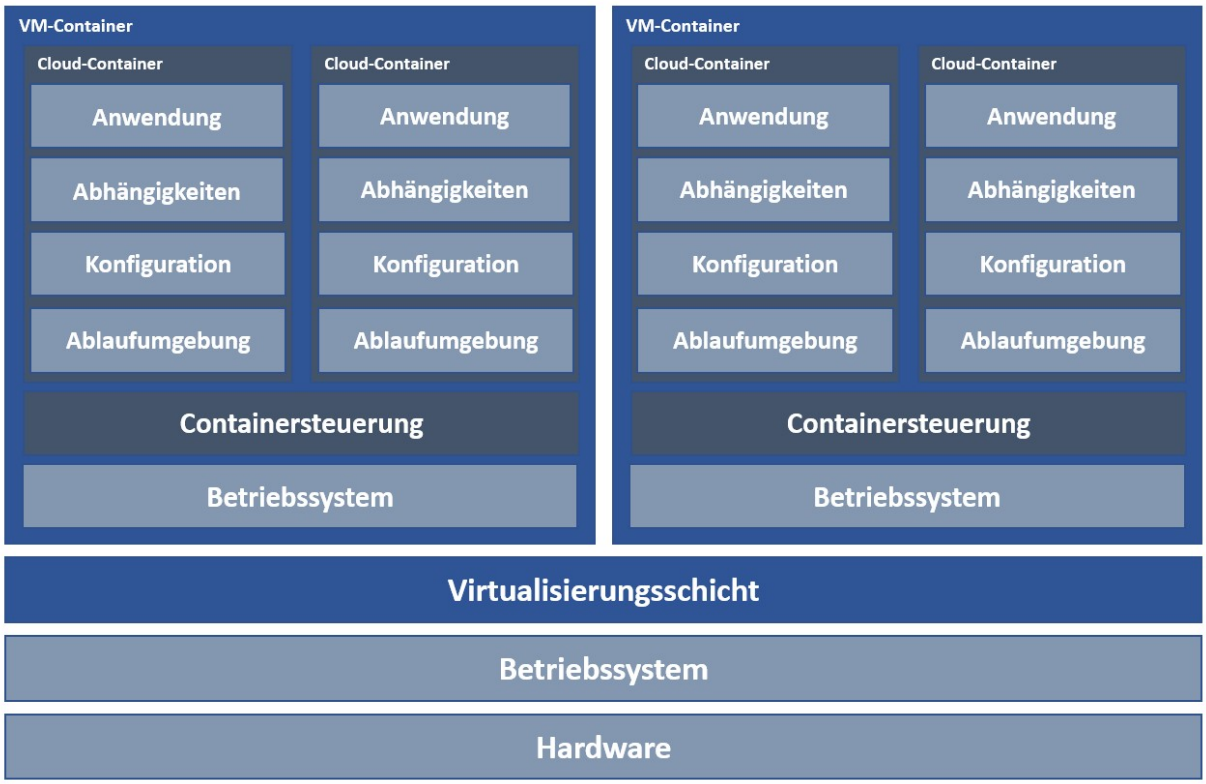

Abbildung 4: Schematischer Aufbau einer Cloud-Infrastruktur mit mehrfacher Virtualisierung

Ein physikalischer Server wird mittels Container einer virtuellen Maschine auf mehrere Instanzen aufgeteilt. In ihnen werden mittels einer Containersteuerung mehrere Anwendungscontainer betrieben. Hardware und Anwendungen werden entkoppelt und die Anwendungen erhalten nur so viele Ressourcen, wie sie für den Betrieb benötigen. Insgesamt steigert Virtualisierung nach Stine (2015) die Effizienz, da Ressourcen genauer zugewiesen werden können, sowie die Geschwindigkeit und Übertragbarkeit von Anwendungen auf andere Umgebungen. Allerdings erhöht sich durch die Vielzahl der Schichten und Systeme die Komplexität, was Einfluss auf den Betrieb haben kann.

In der Praxis zeigt sich, dass die Trennung zwischen Containern nur bis zu einem bestimmten Grad funktionieren kann, da die physikalischen Ressourcen dennoch begrenzt sind. Takai (2017) nennt dieses Phänomen in seinem Buch das *Noisy-Neighbor-Problem* und beschreibt als Lösung die Anwendung des sogenannten *Cullings*. Es dient dazu, zu erkennen, welche Instanzen eines virtuellen Servers oder eines virtuellen Containers mit einem Ressourcenengpass konfrontiert sind (auch wenn dieser Engpass nur in kleinem Ausmaß zum Tragen kommt), und diese Instanzen durch neue zu ersetzen. Dabei werden bestimmte Parameter, wie Antwortzeit oder die Geschwindigkeit von Lese-/Schreiboperationen ständig gemessen und mit den Ergebnissen anderer Instanzen verglichen. Die schlechtesten Instanzen oder Instanzen, die unter bestimmte Grenzwerte fallen, werden heruntergefahren und zerstört. Als Ersatz werden neue Instanzen erzeugt, mit der Hoffnung, dass diese auf einem Server betrieben werden, der nicht so sehr unter Last

steht. *Culling* führt nach Takai (2017) generell zu einer stark erhöhten Komplexität und lässt sich nur in Systemen anwenden, die darauf ausgelegt sind, dass sich die Last oft und schlagartig ändert, da laufend virtuelle Instanzen erzeugt und zerstört werden.

3.5 Cloud-Native Anwendungen

Auf einer Cloud-Plattform können prinzipiell fast alle Anwendungen betrieben werden. Wichtig ist jedoch, dass die Anwendungen ohne Abhängigkeiten zur Hardware entwickelt werden. Diese Anwendungen werden als cloud-native bezeichnet. Außerdem muss es möglich sein, dass mehrere Anwendungsinstanzen parallel und unabhängig voneinander im Betrieb sein können.

Adam Wiggins hat Modelle und Strukturen zusammengefasst, die eine cloud-native Anwendung ausmachen und als *Twelve-Factor-Application* veröffentlicht (Wiggins, Die zwölf Faktoren, 2017). Matt Stine (2015) bezeichnet Sicherheit, Performance, Skalierbarkeit und die Unabhängigkeit zur Laufzeitumgebung als Kernpunkte der zwölf Faktoren. Ziel ist es, eine sichere und performante Anwendung bereitzustellen, die automatisch in mehreren Instanzen betrieben werden kann und die auf andere Umgebungen einfach portierbar ist. Wiggins *Twelve-Factor-Application* erfüllt folgende Punkte:

1. Es soll ein *Sourcecode-Management-System* eingesetzt werden, in dem die Codebasis verwaltet wird. Jede Anwendung hat genau eine Codebasis. Ziel ist es, mit Hilfe eines Sourcecode-Management-Systems viele Deployments zu ermöglichen. Gleichzeitig soll gemeinsam von mehreren Anwendungen genutzter Code erkannt und ausgelagert werden. (Wiggins, Codebase, 2017)

2. *Abhängigkeiten* einer Anwendung sollen deklariert und isoliert werden. Um dies zu erreichen, empfiehlt Wiggins den Einsatz eines Tools wie *Maven*. Das Ziel dieses Faktors ist es, dass eine Anwendung unabhängig von ihrer Umgebung in ihrem Container möglichst überall lauffähig ist. Dafür müssen alle Abhängigkeiten an einer Stelle in der Anwendung bekannt sein. (Wiggins, Abhängigkeiten, 2017)

3. Der dritte Faktor bezieht sich auf die *Konfiguration* einer Anwendung. Sie enthält Werte, die sich normalerweise je nach Abnahmestufe unterscheiden. Typischerweise sind das Zugriffspfade auf Backendsysteme wie Datenbanken. Nach Wiggins dürfen diese Werte nicht im Sourcecode enthalten sein, sondern müssen ausgelagert werden. Er empfiehlt, die Konfiguration als Umgebungsvariablen anzulegen, damit sie möglichst unabhängig von Programmiersprache und Betriebssystem ist. (Wiggins, Konfiguration, 2017)

4. Datenbanken und Backendservices sollen als *angehängte Ressourcen* behandelt werden. Dies bedeutet, dass genutzte Systeme, wie Datenbanken, entkoppelt werden und somit leicht ersetzbar sind. Dafür erfolgt der Zugriff über eine möglichst standardisierte Schnittstelle. (Wiggins, Unterstützende Dienste, 2017)

5. Wiggins nennt drei Phasen, die eine Anwendung durchlaufen muss, um auf einem Server betrieben zu werden. Die erste Phase ist die *Build-Phase*, in der ein Codestand aus dem Sourcecode-Management-Systems transformiert wird, sodass er ausführbar wird. Dabei werden beispielsweise die deklarierten Abhängigkeiten gesammelt und mit verpackt. Die zweite Phase ist die *Release-Phase*. In ihr wird der ausführbare Code mit einer Konfiguration für eine Abnahmestufe kombiniert. Die Anwendung ist in dieser Form lauffähig. Die dritte Phase nennt Wiggins *Run-Phase*. In ihr wird die Anwendung betrieben und kann genutzt werden. Wichtig ist die strikte Trennung zwischen den Phasen. Codeänderungen zur Laufzeit, also in der *Run-Phase*, sind nicht möglich, ohne die *Build-* und *Release-Phase* zu durchlaufen. (Wiggins, Build, Release, Run, 2017)

6. Eine cloud-native Anwendung soll nach Wiggins *stateless*, also zustandslos sein. Zustände werden, wenn nötig, kurzzeitig in Caches oder langfristig in Datenbanken abgelegt. Der Grund für dieses Paradigma ist die Skalierbarkeit in einer Cloud-Umgebung. Instanzen der Anwendung werden nach Bedarf erzeugt und zerstört. Hierbei darf kein Datenverlust auftreten. (Wiggins, Prozesse, 2017)

7. In der Anwendung soll *Port Binding* eingesetzt werden, um harte Abhängigkeiten zu anderen Services, wie Webservern, zu verhindern (Wiggins, Bindung an Ports, 2017). Port Binding regelt, wie Webanwendungen aufrufbar sind. Hierbei soll die cloud-native Anwendung keine Abhängigkeiten zu einem externen System aufweisen. Sie selbst bestimmt das Port Binding, damit sie auch ohne einen Umgebungskontext nutzbar ist. Eine Routing-Schicht setzt gegebenenfalls öffentliche Hostnamen so um, dass die Anwendung angesprochen wird.

8. Nach Wiggins soll in der Anwendung ein *Prozess-Modell* genutzt werden, um Nebenläufigkeit zu realisieren und so eine horizontale Skalierung zu schaffen. Hierbei werden mehrere Prozesstypen eingeführt, die dedizierte Aufgaben erfüllen. Zum Beispiel könnte ein Prozesstyp auf kurzlaufende Operationen, wie eingehende http-Requests, reagieren. Langlaufende Operation könnten in einem anderen Worker-Prozess ausgeführt werden. Durch die unterschiedlichen Prozesstypen kann der Ablauf innerhalb der Anwendung optimiert werden, wenn beispielsweise für langlaufende Operationen mehrere Prozessinstanzen zur Verfügung gestellt werden. (Wiggins, Nebenläufigkeit, 2017)

9. *Prozesse* sollen sich schnell starten und beenden lassen, und zudem nur einmal genutzt werden. Dieses Paradigma unterstützt ebenfalls die Skalierbarkeit der Anwendung, die dadurch insgesamt schneller hoch- oder heruntergefahren werden kann. (Wiggins, Einweggebrauch, 2017)

10. Der zehnte Faktor besagt, dass möglichst keine *Lücken zwischen Entwicklungs- und Produktionsumgebung* vorhanden sind. Wiggins denkt hierbei in drei Dimensionen:

    a. Einer zeitlichen Dimension, in der eine möglichst geringe Zeit zwischen Entwicklungsende und Produktionseinführung liegen soll.

b. Einer personellen Dimension, in der sich Entwicklungs- und Betriebsteam nicht unterscheiden (siehe auch Abschnitt 3.7 - Personalrollen).

c. Einer technischen Dimension, in der keine Unterschiede zwischen den verschiedenen Abnahmestufen existieren. Unter Abnahmestufen werden üblicherweise eine Entwicklungs-, eine Integrations-, eine Test- und eine Produktivumgebung verstanden. Das 10. Paradigma dient dazu, Probleme frühzeitig zu erkennen und Durchlaufzeiten zu senken. Dies geschieht durch den Einsatz von *Continuous Deployment* (näher beschrieben in Abschnitt 3.8). (Wiggins, Dev-Prod-Vergleichbarkeit, 2017)

11. Das *Routing von Logs* übernimmt bei einer cloud-nativen Anwendung die Laufzeitumgebung. Das Ziel ist hierbei nicht mehr eine klassische Textdatei, sondern ein Datenbanksystem, das die Logmeldungen speichert und zugänglich macht. Hintergrund ist hierbei ebenfalls die Skalierbarkeit der Anwendung. Beim Herunterfahren eines Containers gingen klassische Logfiles in einer Textdatei verloren, da der Server mit seinem Filesystem zerstört wird. (Wiggins, Logs, 2017)

12. Administrative Aufgaben, die die Anwendung übernimmt, werden als einmalige Prozesse behandelt. Als Beispiel führt Wiggins hier Datenbankmigrationen an, die durch die Anwendung ausgeführt werden, jedoch von einem Entwickler oder einem Betriebsverantwortlichen ausgelöst werden. Diese administrativen Aufgaben fügen sich in das Prozessmodell ein (siehe Punkt 8). (Wiggins, Admin-Prozesse, 2017)

Die nachfolgende Abbildung 5 zeigt die Aufteilung der zwölf Faktoren auf die vier Handlungsfelder nach Stine (2015): Sicherheit, Performance, Skalierbarkeit und die Unabhängigkeit zur Laufzeitumgebung.

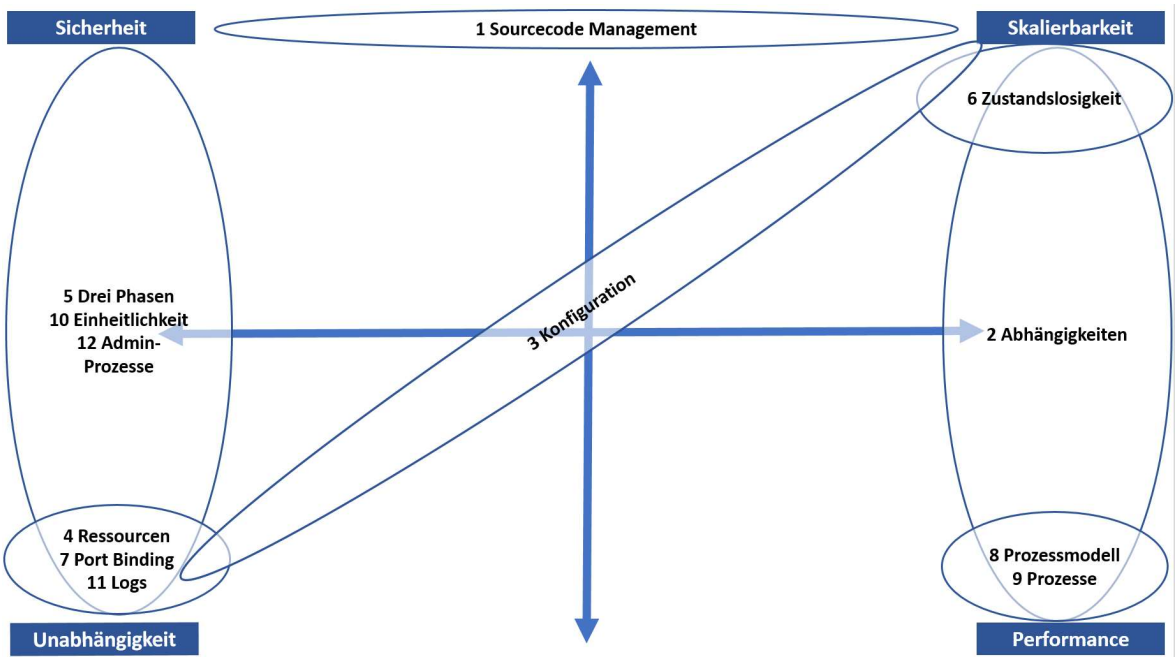

Abbildung 5: Aufteilung der zwölf Faktoren auf die Handlungsfelder nach Stine

Auffällig ist eine Konzentration vieler Faktoren auf die Handlungsfelder *Unabhängigkeit zur Laufzeitumgebung* und *Skalierbarkeit*. Es sind die Haupteigenschaften, mit denen sich cloud-native Anwendungen von herkömmlicher Software abgrenzen. Sie ermöglichen die gewünschte Flexibilität beim Einsatz von Cloud-Anwendungen. Die Themen *Sicherheit* und *Performance* sind üblicherweise bei allen Anwendungen von Belang, unabhängig von einem Einsatz in einer Cloud.

3.6 Microservices

Nach Wolff (2017) dienen Microservices der Modularisierung von Software. Gegenüber anderen Modularisierungsansätzen, wie Klassen und Packages, unterscheiden sie sich jedoch durch einen elementaren Punkt. Sie können unabhängig voneinander installiert werden. Gemeinsam bilden viele kleine, selbständige Services eine Anwendung (Stine 2015). Sie kommunizieren über Schnittstellen untereinander. Takai (2017) definiert eine Anwendung als eine Komposition von Microservices, die eine Benutzerschnittstelle enthält.

Abbildung 6: Ein Geschäftssystem als Komposition von Anwendung und Microservices nach Takai (2017)

Der Einsatz von Microservices bringt gleich mehrere Vorteile mit sich. Durch die Unabhängigkeit beim Deployment können Abhängigkeiten zwischen Anwendungsbestandteilen reduziert werden. Das kann die Zeit senken, die benötigt wird, um neue Funktionen in Produktion einzuführen, da Anwendungstests weniger umfangreich sind und schneller durchgeführt werden können. Zudem verringert die geringe Größe der Microservices das Risiko von Fehlfunktionen, da die Codeblöcke, die in Produktion gebracht werden, kleiner und überschaubarer sind als herkömmliche Anwendungen. Ein weiterer Vorteil zeigt sich im Betrieb von Microservices auf einer Cloud-Infrastruktur. Microservices werden getrennt voneinander in eigenen Containern betrieben. Damit können sie auf einer Cloud-Infrastruktur auch getrennt voneinander skaliert werden. So stehen von rechenintensiven Services mehr Instanzen zur Verfügung. Funktionen, die selten genutzt werden, werden nur bei Bedarf hochgefahren. Die Trennung der Services durch Container steigert zudem die Robustheit, da Probleme einzelner Services nicht direkt andere Services beeinflussen können. Ein letzter Vorteil von Microservices ist nach Wolff (2017) die Nutzung von unterschiedlichen Programmiersprachen. Sie wird ermöglicht, wenn die Schnittstellen, die die Services untereinander nutzen, von der Programmiersprache unabhängig sind. Eine gängige Kommunikationsmethode ist der *Representational State Transfer*, kurz REST. Hierbei erfolgt die Kommunikation auf Basis des Hypertext Transfer Protocol (HTTP), einem weltweiten Standard.

Durch die genannten Vorteile wird ein unabhängiges Arbeiten mehrerer Teams ermöglicht. Sie sind sowohl zeitlich als auch technologisch voneinander entkoppelt und können flexibler Aufträge umsetzen. Wolff (2017) hat dies in zwei Abbildungen illustriert. Abbildung 7 zeigt eine klassische Anwendungsorganisation, die von einer zentralen Stelle ihre Anforderungen erhält. Viele Entwickler arbeiten gemeinsam an einer Deployment-Einheit. Als Herausforderung zeigt die Abbildung die Koordination über ein großes Entwicklungsteam.

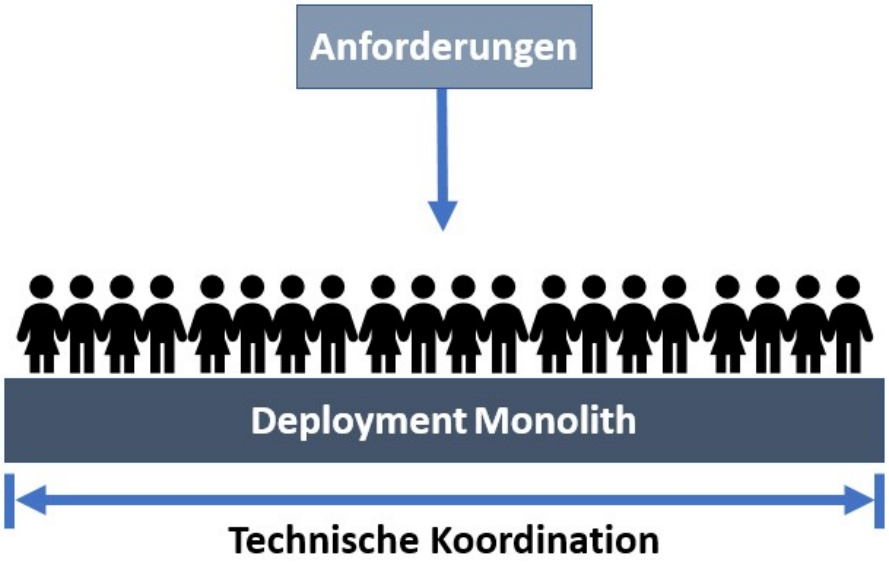

Abbildung 7: Deployment-Monolithen erzwingen technische Entscheidungen und Releases über das gesamte System (nach Wolff, 2017)

Abbildung 8 zeigt dagegen eine Organisationsform, die auf die Erstellung von Microservices ausgerichtet ist. Kleine Teams arbeiten getrennt voneinander an ihren eigenen Services. Die technische Koordination erfolgt im Team. Die Teams müssen lediglich Schnittstellen (technischer und organisatorischer Natur) abstimmen.

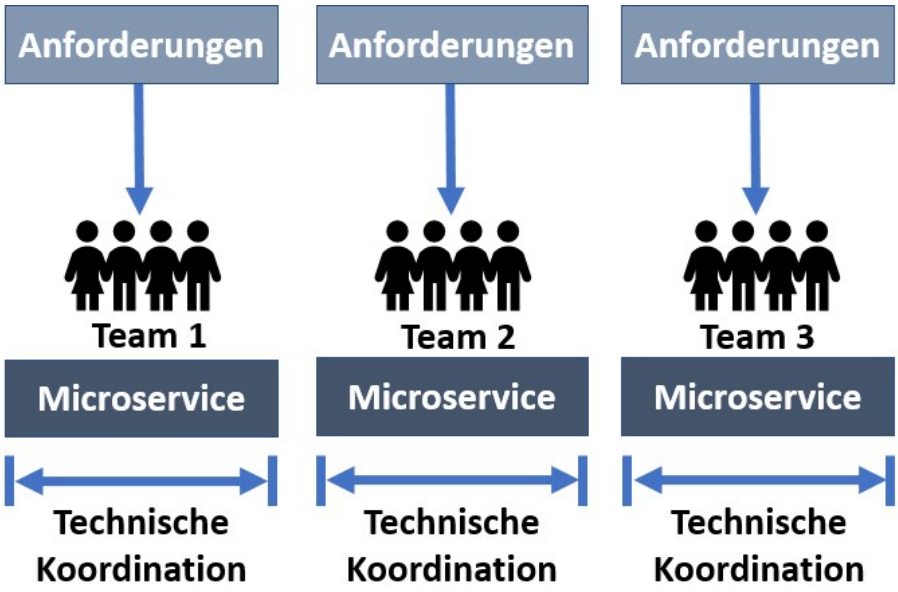

Abbildung 8: Microservices isolieren technische Entscheidungen (nach Wolff, 2017)

Ein klarer Nachteil einer Microservice-Architektur ist nach Wolff (2017) die gesteigerte Komplexität. Durch die Trennung einer Anwendung in mehrere Services steigt die Anzahl der Schnittstellen, der Deployment-Einheiten und der Zuständigkeiten. Organisationen müssen sich dieses Trade Offs bewusst sein, wenn sie eine Microservice-Architektur einführen möchten.

Eine weitere Herausforderung sieht Wolff (2017) in der notwendigen Umstrukturierung von Unternehmen, die Microservices entwickeln möchten. Mit der Unabhängigkeit einzelner Teams voneinander werden weniger übergreifende Abstimmungsgremien benötigt. Softwarearchitekten und Manager werden in Teilen nicht mehr benötigt. Des Weiteren muss ein gewisser Automatisierungsgrad innerhalb der Organisation vorliegen. Tests und Deployments sollten automatisch ausgeführt werden, damit einerseits Änderungen ohne Aufwand getestet und so Seiteneffekte erkannt werden können, und andererseits das erhöhte Deployment-Aufkommen gehandhabt werden kann.

Gegenüber einer Serviceorientierten Architektur (siehe Abschnitt 2.2) unterscheiden sich Microservices nach Takai (2017) insbesondere dadurch, dass ein Microservice nur eine Aufgabe erfüllt. Er ist dadurch kleiner und somit leichter test- und analysierbar. Zudem sind die Skalierbarkeit und die Wartbarkeit höher. Ein Nachteil entsteht durch die gesteigerte Kommunikation über ein Netzwerk, die die Latenzzeiten erhöht. Auch steigt die Komplexität durch die Vielzahl an Deployment-Einheiten.

3.7 Personalrollen: Das DevOps-Modell

Mit der Nutzung einer Cloud-Infrastruktur und der Einführung einer Microservices Architektur geht nach Peschlow (2016) oft auch eine Änderung am klassischen Rollenmodell einher. Ein Rollenmodell, das oft mit diesen Trends verbunden wird, ist das *DevOps*-Modell. Der Name setzt sich zusammen aus den englischen Worten *Development* (Entwicklung) und *Operations* (Betrieb). In dieser Organisationsstruktur sind Programmierer nicht nur für die Entwicklung einer Anwendung zuständig, sondern übernehmen auch Betriebsaufgaben. Dazu zählt die Installation der Anwendung und die Betreuung im laufenden Betrieb. Eine Trennung zwischen Entwicklungs- und Betriebseinheiten, wie sie in der Vergangenheit weit verbreitet war, findet nicht mehr statt. *DevOps*-Teams bestehen aber nicht zwingend nur aus Softwareentwicklern und Systemadministratoren. Nach Augsten (2017) können auch Manager und Tester einbezogen werden, um den gesamten Entwicklungsprozess personell zu erfassen.

DevOps-Teams erhalten größere Befugnisse, jedoch auch eine größere Verantwortung. Ziel ist es, die Produktivität zu erhöhen, indem Kommunikationswege verkürzt oder vermieden werden und Hierarchien verflacht werden (Augsten, 2017). Erkenntnisse, die beim Betrieb der Anwendung entstehen, können direkt in die Weiterentwicklung einfließen. Das kann eine korrigierte Fehlerbehandlung sein, optimierte Logausgaben oder die Behebung von Instabilitäten. Damit ein *DevOps*-Team effektiv arbeiten kann, ist auch eine Automatisierung von wiederkehrenden Arbei-

ten, wie Tests und Deployments, essentiell (Peschlow, 2016). Dadurch können sich die Teammitglieder den Kernaufgaben widmen. Zudem muss es innerhalb dieser Teams klar definierte Prozesse geben, denen die Teammitglieder folgen.

Eine Herausforderung bei der Einführung von *DevOps*-Teams stellt sich beispielsweise bei der Definition der Reaktionszeiten der Teams. In klassischen Organisationsformen ist der IT-Betrieb oft als *Follow-the-sun*-Einheit aufgestellt, die rund um die Uhr handlungsfähig ist. Wenn nun Entwickler diese Aufgabe erfüllen sollen, ist es meist eine Herausforderung, diese Einheiten ebenso rund um die Uhr verfügbar zu machen. Eine oft gewählte Lösung besteht aus einem Mittelweg zwischen dem klassischen Modell und einem *DevOps*-Team. Durch übergreifende Teammitglieder rücken Entwicklung und Betrieb näher zusammen. Beide Einheiten haben aber grundsätzlich noch ihre eigene Domäne. In der Praxis kann das bedeuten, dass Betriebs- und Entwicklungseinheiten gemeinsame Meetings haben, damit ein Wissensaustausch stattfindet. Die Betriebseinheiten sind so frühzeitig und detailliert über Änderungen und Neuerungen informiert und Probleme aus dem Betrieb werden schnell zurück an die Entwickler gespielt. In manchen Fällen machen die Mitglieder der Teams Hospitationen in der anderen Domäne, um sie besser kennenzulernen und den Kontakt zwischen beiden Gruppen zu vereinfachen.

3.8 Entwicklungsverfahren: Continuous Integration and Delivery

Die Begriffe *Continuous Integration*, *Continuous Delivery und Continuous Deployment* beschreiben Methoden und Techniken, die dazu dienen, Software möglichst zügig und automatisiert in Produktion zu bringen. Damit ein Programm produktionsreif ist, muss es in der Regel die Schritte *Build*, *Integration*, *Test*, *Release* und *Deployment* durchlaufen. Die folgende Abbildung 9 zeigt grafisch die Reichweite der verschiedenen Entwicklungstechniken auf, in Bezug auf die verschiedenen Phasen des Entwicklungsprozesses.

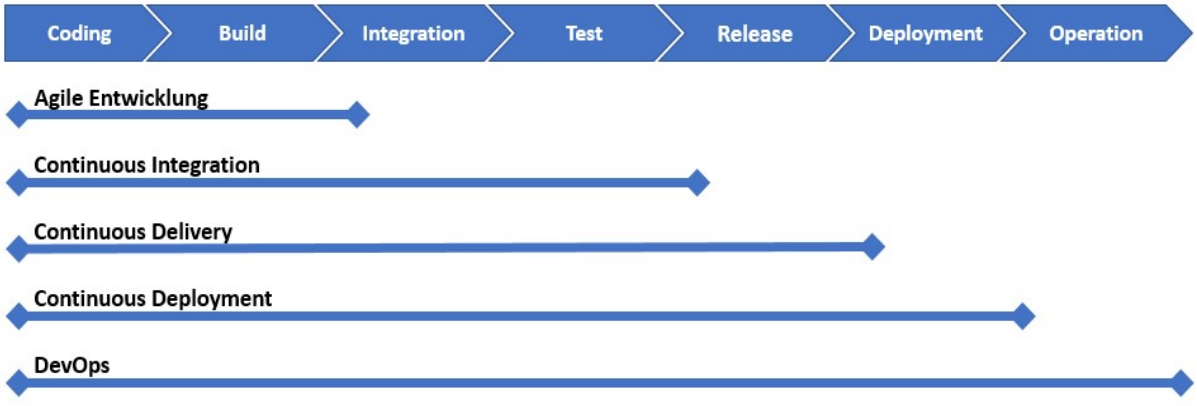

Abbildung 9: Vergleich verschiedener Entwicklungstechniken in Bezug auf den Entwicklungsprozess (nach Soparkar, 2017)

Continuous Integration setzt bei den Schritten *Integration*, *Test* und *Release* an und ergänzt damit die typische, agile Entwicklung. Das Verfahren wird von Duncan Winn (2016) als eine Entwicklungsvorgehensweise definiert, die das Ziel hat, Softwarekomponenten stetig zu integrieren. Diese bedeutet, dass bei einer Änderung des Softwarestandes in einem zentralen Sourcecode-Management-System ein Build erzeugt wird. Dies geschieht automatisch durch einen Buildserver. Während des Builds werden alle vorhandenen Tests ausgeführt, damit Fehler und Seiteneffekte der neuesten Änderungen direkt auffallen. *Continuous Integration* ist jedoch nicht nur die Nutzung von automatisierten Prozessschritten, sondern auch eine kulturelle Umstellung im Entwicklungsteam. Damit die Mittel der Automatisierung sinnvoll und effektiv genutzt werden können, müssen Softwareentwickler konsequent Testfälle für neue Funktionen anlegen. Nur so ist ein automatisierter Build-Prozess effektiv, da Fehler frühzeitig aufgedeckt werden. Damit dieser Build-Prozess sein Ziel erreichen kann, müssen Änderungen regelmäßig und in kurzen Abständen in das zentrale Sourcecode-Management-System übertragen werden. Ansonsten fallen bei der Integration von Änderungen möglicherweise große Konsolidierungsaufwände an, und Fehler werden erst nach diesem Schritt gefunden. Das Ziel von *Continuous Integration* ist es, eine höhere Softwarequalität zu erhalten und gleichzeitig eine Konzentration der Softwareentwickler auf die Kernarbeit zu ermöglichen, indem Prozesse automatisiert stattfinden. Zudem soll möglichst zu jedem Zeitpunkt ein nutzbarer Stand der Anwendung vorliegen. Als Folge kann schneller auf Kundenanforderung reagiert werden, da Rüstzeiten entfallen und Routinearbeiten aufwandsarm stattfinden. Die automatisierten Tests können einfache Testfälle ersetzen und geben so personelle Kapazitäten frei, die früher durch diese Aufgabe gebunden waren. Einen händischen Abnahmetest können diese Testfälle teilweise ersetzen.

Continuous Delivery schließt auf der Prozessebene an *Continuous Integration* an. Das Ziel ist es, den Auslieferungsprozess von Software zu verkürzen und Software somit schneller in Produktion zu bringen. Zentraler Ansatzpunkt ist auch hierbei eine weitreichende Automatisierung. Es entsteht eine Deployment-Pipeline, die alle Schritte nach einer Änderung im Sourcecode-Management-System bis zur Produktionseinführung beinhaltet (Bernstein, 2015). In ihr können auch manuelle Tests enthalten sein. Das Ziel ist es jedoch möglichst viele Regressionstestfälle automatisiert ablaufen zu lassen. Auftraggebern steht durch die Nutzung von *Continuous Delivery* mit einem geringen zeitlichen Versatz der aktuelle Entwicklungsstand für Tests zur Verfügung. Dies ermöglicht kürzere Feedbackschleifen und eine zielführende Softwareentwicklung

Continuous Deployment reicht über die beiden vorgestellten Techniken hinaus. Bei *Continuous Delivery* ist die Produktionseinführung eines Releases noch eine bewusste Entscheidung, die eine manuelle Freigabe erfordert. Bei *Continuous Deployment* ist dieser Schritt ebenfalls automatisiert (Takai, 2017). Wenn nach dem Bauen des Releases keine Testfälle fehlschlagen oder der Prozess nicht durch einen Entwickler unterbrochen wird, wird ein Release automatisiert in Produktion installiert. Continuous Deployment eignet sich insbesondere für Microservices, wenn die geänderten Codestellen überschaubar sind und es möglich ist, schnell eine alte Version der Anwendung in Produktion zu bringen, falls Fehler auftreten. In der Praxis ist es auch verbreitet, neue Funktionen mit einem Schalter zu versehen, sodass der Livegang bewusst durchgeführt werden kann. Zudem wird *Continuous Deployment* genutzt, um neue Funktionen vorab einem

kleinen Nutzerkreis zur Verfügung zu stellen und so ohne großen Zeitversatz Rückmeldung einzuholen. Takai (2017) weist darauf hin, dass in der Praxis oft Compliance oder Revisionsgründe gegen ein automatisch ausgeführtes Produktionsdeployment sprechen. In (meist großen) Unternehmen oder bei kritischen Anwendungen muss die Freigabe durch einen Entwickler oder Produktverantwortlichen erfolgen.

4 LEGACY-ANWENDUNGEN

4.1 Begriffsdefinition

Eine einheitliche Definition für Legacy Software ist in der Literatur nicht zu finden. Das englische Wort *Legacy* bedeutet übersetzt Altsystem, Altlast oder Erbe. Nach Michael Feathers (2011) ist Legacy Code damit wörtlich gemeint eine Anwendung, die von jemand anderem übernommen oder geerbt wurde. Lewis, Plakosh und Seacord (2003) nennen das Klischee, dass Legacy Code bereits der Code ist, der gestern geschrieben wurde. Und laut den drei Autoren ist dieses Klischee zunehmend wahr. Im Umkehrschluss bedeutet dies überspitzt, dass jede Art von Software per se Legacy-Software ist. Der Grund für die Annahme der drei Autoren ist die Beobachtung, dass die typischen Wartungs- und Verbesserungsmaßnahmen Software in vielen Fällen nicht auf dem aktuellen Stand der technologischen Entwicklung halten können. Es muss explizit eine Modernisierung durchgeführt werden. Eine deutlich einfachere Definition veröffentlichte Michael Feathers (2011), da ihm die wörtliche Übersetzung nicht weit genug geht: Für ihn gilt eine Anwendung als Legacy Software, wenn sie nicht mit automatisierten Tests versehen ist. Das Alter der Anwendung spielt für Feathers keine Rolle.

Nach Lewis et al. (2003) zeichnet sich Legacy-Software durch folgende Eigenschaften aus:

1. Sie ist in der Regel historisch gewachsen und im Laufe der Jahre um neue Funktionalitäten erweitert worden.

2. Oft sind diese Erweiterungen minimalinvasiv, also ohne Modernisierung der gesamten Anwendung vorgenommen worden.

3. Die Anwendung gilt allgemeinhin als sehr komplex.

4. Aufgrund der teils langen Entwicklungs- und Betriebszeit sind die Entwicklungs- und Laufzeitumgebungen oft veraltet und nicht mehr Stand der Technik.

5. Die vorhandene Dokumentation ist unzureichend oder nicht vollständig. Meist beschränkt sie sich auf die angebotenen Schnittstellen, behandelt jedoch nicht die innere Funktionsweise einer Anwendung.

6. Entwicklerkapazitäten mit Kenntnis der Anwendung für Wartung und Erweiterungen ist rar oder nicht vorhanden.

Legacy-Software ist per Definition in Betrieb und erfüllt die ihr zugedachten Aufgaben. Die genannten Eigenschaften führen in der Regel erst zu Problemen, wenn die Software angepasst werden soll oder muss.

4.2 Entstehung von Legacy-Software

Legacy-Software entsteht nach Lewis et al. (2003) aus verschiedenen Gründen. Ein Phänomen, das in vielen Unternehmen auftritt, ist, dass Softwareprojekte nur finanziert werden, wenn sie einen unmittelbaren Nutzen hervorbringen. Bei Modernisierungsvorhaben ist der Gewinn oder Nutzen allerdings oft selten messbar. Er könnte sich beispielsweise in einer besseren Wartbarkeit der Software zeigen. Die Zeit, die Entwickler hierdurch jedoch in Zukunft einsparen, weil sie Fehler leichter beheben können, lässt sich nicht beziffern.

Auch der Einsatz von eingekaufter Software schützt nicht vor der Entstehung von Legacy-Software. In der Praxis zeigt sich, dass die Softwarehersteller zwar regelmäßig moderne Versionen ihrer Produkte in Umlauf bringen, Kunden diese jedoch nicht einsetzen. Die Gründe dafür sind unter anderem, dass die Kunden in einem Releasewechsel ein hohes Risiko für Instabilitäten sehen. Aber auch die hohen Kosten für einen Releasewechsel werden gescheut, wenn das neue Release nicht ein unmittelbares Problem behebt. (Lewis, Plakosh, & Seacord, 2003)

Zusammengefasst lässt sich sagen, dass der Hauptgrund für die Entstehung von Legacy-Anwendungen die fehlende Investition in funktionierende Software ist. Releasewechsel und Wartungsarbeiten, wie ein Refactoring einer Anwendung, ergeben oft keinen direkt erkennbaren Kundennutzen, der die Investitionen rechtfertigen würde.

4.3 Probleme von Legacy-Software

Gemeinhin ist der Spruch „*Never change a running system“* bekannt. Die Probleme von Legacy-Systemen treten allerdings zu tage, wenn ein System zu lange nicht verändert wurde. Feathers (2011) nennt vier Hauptgründe, warum Software verändert wird:

1. Eine Funktion wird hinzugefügt.
2. Ein Fehler wird beseitigt.
3. Das Design wird verbessert.
4. Die Ressourcennutzung wird optimiert.

Takai (2017) ist in seiner Aufzählung noch detaillierter. Er nennt als Treiber für Änderungen gesetzliche Vorgaben und Regularien sowie Geschäftsanforderungen. Auch aus Sicherheitsgründen müssen oftmals Änderungen an Legacy-Systemen erfolgen. Zusätzlich nennt der Autor Änderungen am Kontext einer Anwendung, die Anpassungen erforderlich machen oder Sonderfälle, wie einen Wechsel in der Zuständigkeit. Wenn ein Team eine bestehende Anwendung übernimmt, wird diese laut dem Autor oftmals den neuen Rahmenbedingungen (andere Technologieauswahl) angepasst.

Das Hauptproblem veralteter Software ist, dass im Laufe der Zeit wertvolles Wissen verloren geht. Das Wissen, das bei der Implementierung einer Anwendung im Unternehmen vorhanden ist, veraltet mit der Zeit und wird vergessen. Es zeigt sich zum Beispiel, dass bei Legacy-Software der genaue Zweck und die Arbeitsweise oft unklar sind. Eine Dokumentation beschreibt häufig nur, wie ein Programm genutzt wird, nicht wie es arbeitet. Anpassungen an einer Anwendung, wie die von Michael Feathers genannten, oder die Ablösung durch ein anderes Programm werden so stark erschwert und sind risikoreich, da zunächst eine ausführliche Analyse stattfinden muss (Lewis, Plakosh, & Seacord, 2003).

Das Problem des fehlenden Wissens zeigt sich aber auch noch auf einer anderen Ebene: Für Unternehmen ist es sehr schwer, Experten für alte Technologien zu finden. Ein Beispiel sind die Großrechnersprachen COBOL und PL/1, die seit den 1950er und 1960er Jahren im Einsatz sind. Für Unternehmen ist es zu teuer, Mitarbeiter mit passendem Know-how in diesen Sprachen vorzuhalten, wenn sie nicht dauerhaft mit Aufgaben ausgelastet sind. Deshalb sind Unternehmen nach Lewis et al. (2003) häufig auf externe Spezialisten angewiesen, beziehungsweise von diesen abhängig. Selbst wenn eine Anwendung über Jahre hinweg stabil und ohne Betreuung funktioniert, kann sie ausfallen. Auch kann sich plötzlich eine Anforderung für eine Änderung ergeben, beispielsweise wenn eine Sicherheitslücke entdeckt wird. In solchen Fällen sind Unternehmen, wie angesprochen, auf externe Hilfe angewiesen. Doch es kann zu Situationen kommen, in denen der Markt den Bedarf nicht decken kann. Ein Ereignis, bei dem sich in kurzer Zeit ein massiver Bedarf an COBOL-Entwicklern zeigte, war das *Jahr-2000-Problem*, bei dem Ende der 1990-Jahre festgestellt wurde, dass viele Programme möglicherweise aufgrund der verkürzten Jahreszahl *00* nicht mehr korrekt arbeiten würden: Es folgte weltweit eine Überprüfung von Programmen, die Jahresangaben verarbeiten. Daran schloss sich eine Welle von Fehlerbeseitigungen und Datenbereinigungen an. Die BBC (2000) schätzte die Kosten für diese Maßnahmen weltweit auf bis zu 600 Milliarden Dollar.

Neben den genannten finanziellen Folgen kann veraltete Software für Unternehmen auch zu einem Imageproblem werden. Über den schlechten Zustand der Computersysteme der *Deutschen Bank* wurde mehrfach in der deutschen Presse berichtet (Pletter & Storn, 2016). Der Vorstandsvorsitzende John Cryan bestätigte dies nach seinem Amtsantritt und kündigte umfangreiche Konsolidierungs- und Migrationsprojekte an (Meck, 2016).

Lewis et al. (2003) haben in ihrem Buch analysiert, wie gut eine automatische Umwandlung von einer alten Anwendung in eine moderne Form funktioniert. Sie führten diesen Test mit einem COBOL-Anwendung aus, die in C++ übersetzt wurde. Es zeigte sich, dass der Code zwar ausführbar ist, jedoch nicht den üblichen Programmierstandards entspricht und nicht wartbar ist. Die größten Probleme entstehen durch die Übersetzung einer prozedural aufgebauten Anwendung in eine objektorientierte Form. Technische Übersetzer können den Wechsel dieses Programmierparadigmas nicht leisten. Nach Lewis et al. (2003) waren die generierten Klassen sehr weitläufig und ohne erkennbare Hierarchie. Die Methoden und Variablen waren alle öffentlich und Daten wurden häufig über globale Variablen manipuliert. Zudem wurden erklärende Kommentare aus dem alten Sourcecode nicht übernommen. Das Ergebnis entsprach damit nicht den üblichen Softwarekonventionen, konnte weder gewartet noch weiterentwickelt werden und war damit nicht

sinnvoll einsetzbar. Die Autoren schlossen daraus, dass die Arbeit an Altanwendungen weiterhin von qualifiziertem Personal vorgenommen werden muss.

Norbert Eder (2017) erläutert in seinem Artikel „*Wie gehe ich mit Legacy Code um?*“ noch ein weiteres Problem, das im Umgang mit Legacy-Software auftritt. Er beschreibt seine Beobachtung, dass Anpassungen oft unerwartete Nebeneffekte an anderen Stellen einer Anwendung hervorrufen. Die korrekte Reaktion wäre eine saubere, wenn auch zeitintensive Behebung der Probleme. In angespannten Projektsituationen wird nach Eder jedoch häufig der schnellste Lösungsweg gewählt. Dieser hat oft einen nachteiligen Effekt auf die Architektur und Verständlichkeit. Im Endeffekt verschlimmert sich so die Wartbarkeit einer Altanwendung weiter. Eder spricht mit Blick auf dieses Phänomen von einer Abwärtsspirale, da eine suboptimale Problemlösung langfristig weitere Probleme hervorruft.

4.4 Umgang mit Legacy-Software

Nach Lewis et al. (2003) entwickeln sich die Anforderungen an ein Softwaresystem im Laufe der Zeit immer weiter. Eine einmalige Entwicklung, um die Kundenwünsche zu befriedigen, ist demnach nicht ausreichend. Kleinere Verbesserungen und Erweiterungen können im Rahmen von Wartungsarbeiten abgehandelt werden. Im Laufe der Zeit wird jedoch die Differenz zwischen Kundenwünschen und Implementierungsstand so groß, dass eine aufwendigere Modernisierung erfolgen muss. Wenn auch Modernisierungsvorhaben nicht mehr ausreichend sind, folgt im Lebenszyklus der Anwendung die Ablösung. Dies zeigt die folgende Abbildung 10.

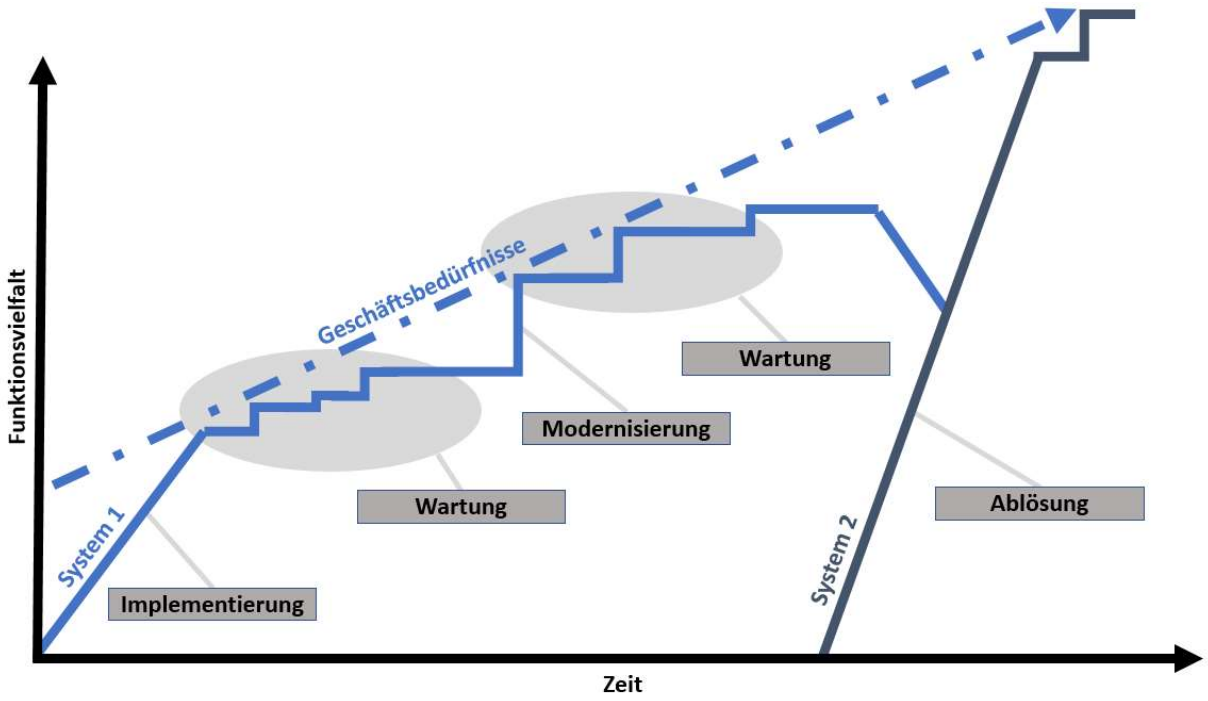

Abbildung 10: Lebenszyklus einer IT-Anwendung nach Lewis et al. (2003, S. 8)

Wartungsarbeiten werden von Lewis et al. (2003) als iterativen und inkrementellen Prozess beschrieben, der kleine Änderungen an einer Anwendung umfasst. Größere, strukturelle Änderungen fallen unter den Begriff *Modernisierung*. Hierbei werden tiefgreifende Änderungen durchgeführt, die der Stabilität, Weiterentwicklung oder Architekturkonformität dienten (Lewis et al., 2003). Es kann unterschieden werden zwischen White-Box und Black-Box Modernisierungsvorhaben. Bei White-Box-Modernisierungen ist ein tiefes Verständnis der Anwendung und der verwendeten Technologien nötig, da die Änderungen in der Anwendung durchgeführt werden. Bei einer Black-Box Modernisierung werden nur die Ein- und Ausgaben des Systems analysiert und verändert. Die Anpassungen erfolgen in einer Zwischenschicht, einem Wrapper, der zwischen der Anwendung und ihren Nutzern oder den Backendsystemen liegt. Das bedeutet, dass die Anwendung in ihrer bestehenden Form auf einer anderen Technologie betrieben werden kann oder ein neues Datenmodell zur Kommunikation verwendet, ohne dass Anpassung innerhalb der Anwendung vorgenommen werden müssen.

Eine wichtige vorbeugende Maßnahme ist nach Feathers (2011) und Eder (2017) das Schreiben von automatisierbaren Testfällen. Dies muss nicht zwingend für die gesamte Anwendung nachgeholt werden, jedoch sollten neue Funktionen und Erweiterung mit automatisierbaren Tests geprüft werden. So wird der Grundstein für eine nachhaltigere Entwicklung gelegt. Wichtig ist nach Eder (2017), dass der Aufwand für das Schreiben der Testfälle von Anfang an in den Projektkosten berücksichtigt ist. Dadurch können sich die Entwickler bewusst Zeit für die Erstellung nehmen. Das Einführen von Testfällen hat nach Eder (2017) noch einen Nebeneffekt. Wenn bei der ur-

sprünglichen Implementierung kein Wert auf eine gute Testbarkeit gelegt wurde, ist ein Refactoring (siehe auch Absatz 5.3) des Legacy-Codes unumgänglich, um Testfälle einzuführen. Die Anwendung wird dadurch in kleinen Schritten modernisiert und aufgeräumt. Zusätzlich steigt bei den Entwicklern die Kenntnis über die Abläufe in der Anwendung.

Feathers (2011) unterscheidet generell zwei Vorgehensweisen zur Durchführung von Änderungen an Legacy-Software. Eine Änderung, die durch neue, automatisierte Tests abgesichert wird, nennt er *Cover and Modify.* Kernbestandteil dieser Technik sind die genannten Tests, die Feathers als *Sicherheitsnetz* bezeichnet. Das Gegenteil dazu bezeichnet der Autor als *Edit and Pray*, eine Vorgehensweise, die seiner Meinung nach sehr häufig gewählt wird. Die Herangehensweise an Änderungen beginnt mit einer gründlichen Analyse der zu ändernden Anwendung. Daraufhin werden die Änderungen geplant und durchgeführt. Es erfolgt ein manueller Test, den Feathers mit dem Wort *herumprobieren* beschreibt. Er dient dazu, die geänderte Funktionalität zu prüfen und mögliche Seiteneffekte aufzudecken. Allerdings bietet dieses Vorgehen nicht die Gewissheit, dass alle Fehler gefunden werden. Deshalb nennt Feathers den zweiten Teil des Vorgehens *Pray*.

5 SOFTWAREMIGRATIONEN

Dieses Kapitel behandelt das Thema *Softwaremigrationen*. Es gibt einen Überblick über die Gründe, die zu einer Softwaremigration führen, und über die verschiedenen Arten, auf die Software in eine neue Umgebung überführt werden kann. Es schließt mit einem exemplarischen Ablauf einer Softwaremigration.

5.1 Begriffsdefinition

Der Begriff *Softwaremigration* beschreibt nach Heilmann, Sneed und Wolf (2010) „die Überführung eines Softwaresystems in eine andere Zielumgebung oder in eine sonstige Form, wobei die fachliche Funktionalität unverändert bleibt“. Eine besondere Wichtigkeit enthält der letzte Halbsatz dieser Definition: fachliche Änderungen an der betroffenen Anwendung sollten nicht mit der Migration, sondern vor oder nach ihr umgesetzt werden. Das Ziel einer Migration ist es, die Funktionalität einer bestehenden Anwendung in einem anderen Kontext bereitzustellen, sodass diese für Nutzer (möglichst) unverändert nutzbar ist. Hierauf verweisen auch Gimnich und Winter (2005) und Rosenberger (2013) in ihrer Definition des Begriffs Softwaremigration. Anwender haben durch die Softwaremigration nicht zwangsläufig einen Mehrwert. Nach Rosenberger (2013) besteht der Mehrwert einer Migration darin, zukünftige Anpassungspotentiale zu schaffen, die bei einer späteren Weiterentwicklung genutzt werden können. Als Beispiel führt er die Verbesserung von Qualitätseigenschaften an, wie die Erhöhung der Wartbarkeit oder der Performanz.

5.2 Gründe für Softwaremigrationen

Die Treiber für Softwaremigrationen sind hauptsächlich die fortschreitenden Entwicklungen in der Informationstechnologie. Nach dem *Mooresches Gesetz* (Moore, 1965) verdoppelt sich die Speicherkapazität von Rechnern alle zwei Jahre. Nach Heilmann et al. (2010) verändert sich die Softwaretechnologie alle fünf Jahre grundlegend. Die Autoren beschreiben *Technologiewellen*, die ungefähr in diesem Abstand auftreten. Das sind beispielsweise die Client-Server-Technologie, die Objektorientierung, die Webtechnologie und die Serviceorientierte Architektur (SOA). Die Autoren sprechen des Weiteren von einem Modezwang, neue Technologie ausprobieren zu müssen. Dadurch wird bestehende Software für die Autoren mit dem Beginn jeder neuen Technologiewelle zur Legacy-Software mit einem Migrationsbedarf. Der Grund für diese vergleichsweise hohe Änderungsdauer von fünf Jahren liegt an den Anwendern, die erst den Umgang mit neuen Technologien erlernen müssen.

Der typische Lebenszyklus für Software enthält in der Regel drei Phasen:

1. Die Entwicklung der Software bis hin zur Produktionseinführung
2. Der Betrieb einer Anwendung, mit Weiterentwicklungen oder Fehlerkorrekturen
3. Die Abschaltung einer Anwendung oder Ablösung durch ein anderes Programm

Softwaremigrationen dienen dazu, die Betriebsphase zu verlängern und damit länger von den Anfangsinvestitionen zu profitieren, bevor eine Anwendung endgültig abgelöst wird.

Die Migration von Anwendungen auf neue Technologien wird durch zwei Strömungen ausgelöst. Die erste ist der beschriebene technische Wandel. Mit dem Fortschreiten der technologischen Entwicklung ist die Rückwärtskompatibilität über mehrere Innovationszyklen nicht mehr gegeben. Um alte und neue Anwendungen miteinander zu betreiben, kommt im Laufe der Zeit der Punkt, an dem Migrationen unumgänglich sind. Zudem geben die Anbieter von Soft- oder Hardware nur für einen begrenzten Zeitraum Support für ihre Produkte. Danach können Anwender keine Funktionsupdates oder Sicherheitspatches mehr erwarten.

Welche Folgen veraltete Software haben kann, zeigte sich bei der Angriffswelle des Erpressungstrojaners *Petya* im Jahr 2017. Allein die dänische Großrederei *Maersk* hatte aufgrund des Angriffes finanzielle Einbußen von geschätzt 200 bis 300 Millionen Euro (Gierow, 2017). Die Sicherheitslücke, die der Trojaner im Betriebssystem *Windows* nutzte, war bereits länger bekannt und wurde vom Hersteller *Microsoft* vor dem Angriff geschlossen (Bundesamt für Sicherheit in der Informationstechnik, 2017). Die Reederei *Maersk* hatte es jedoch versäumt, die entsprechenden Updates zu installieren.

Der zweite Grund, der Softwaremigrationen notwendig machen kann, ist das technische Wissen der beteiligten Mitarbeiter. Viele Softwareentwickler sind nach dem Entwicklungsende einer Anwendung nicht weiter mit ihr betraut. Aufgebautes Wissen geht mit der Zeit verloren und wird selten gut genug dokumentiert, um es an Kollegen weiterzureichen. Zudem haben junge Kollegen im Team im Laufe der Zeit auch kein Wissen mehr über alte Technologien. Damit fehlt sowohl Wissen über die Anwendung an sich, als auch über genutzten Technologie. Diese Wissenslücke kann fatal sein, wenn plötzlich äußere Umstände Änderungen an diesen Altanwendungen notwendig machen. Ein Beispiel hierfür ist die 2014 bekannt gewordene POODLE Lücke (Padding Oracle On Downgraded Legacy Encryption) im SSL Protokoll, die im Webumfeld Anpassungen an vielen Anwendungen nötig macht (Schmidt, Poodle: So funktioniert der Angriff auf die Verschlüsselung, 2014 und Schmidt, So wehren Sie Poodle-Angriffe ab, 2014).

Generell gibt es nach Heilmann et al. (2010) drei Möglichkeiten, um alte Software auf einen neuen Stand zu heben. Die Autoren unterscheiden folgende Varianten:

1. Eine Migration hin zu einem Zielsystem
2. Eine Neuentwicklung, die den Zielvorstellungen entspricht
3. Eine Ablösung durch Funktionsübernahme einer anderen Anwendung

Eine vierte Alternative ist die bewusste Beibehaltung des Status quo. Diese Variante wird meist aus Kosten- oder Risikogründen gewählt. Sie wird im weiteren Verlauf dieser Arbeit nicht betrachtet. Insbesondere auf die Optionen *Migration*, *Neuentwicklung* und *Ablösung* soll in der Folge eingegangen werden.

5.3 Arten von Softwaremigration

Softwaremigrationen können in zwei Dimensionen unterschieden werden. Die erste Dimension bezieht sich auf die Plattform, die verändert wird. Gimnich und Winter (2005) unterschieden drei Typen:

1. Migrationen der Laufzeitumgebung
2. Migration der Architektur
3. Migration der Entwicklungsumgebung

Die drei Typen unterscheiden sich insbesondere dadurch, in welchem Maße die Anwendung angepasst wird. Bei der Migration der Laufzeitumgebung erfolgt in der Regel eine Anpassung an die neue Laufzeitumgebung, sofern dies nötig ist. Ebenso werden die Anpassungen bei der Migration der Entwicklungsumgebungen möglichst gering gehalten. Dem gegenüber können Architekturmigrationen in Bezug auf die Anwendung sehr aufwandsintensiv sein.

Die Migration der Laufzeitumgebung ist dabei der Anwendungsfall, der in der Praxis am häufigsten auftritt. Deshalb soll im Laufe dieser Arbeit hauptsächlich auf diese Migrationsart eingegangen werden.

Die zweite Dimension beschreibt die Art der Änderungen an der zu migrierenden Anwendung. Um ein bestehendes Softwaresystem in einen neuen Kontext zu überführen, gibt es insgesamt fünf verschiedene Möglichkeiten. Diese werden in der folgenden Abbildung 11 dargestellt:

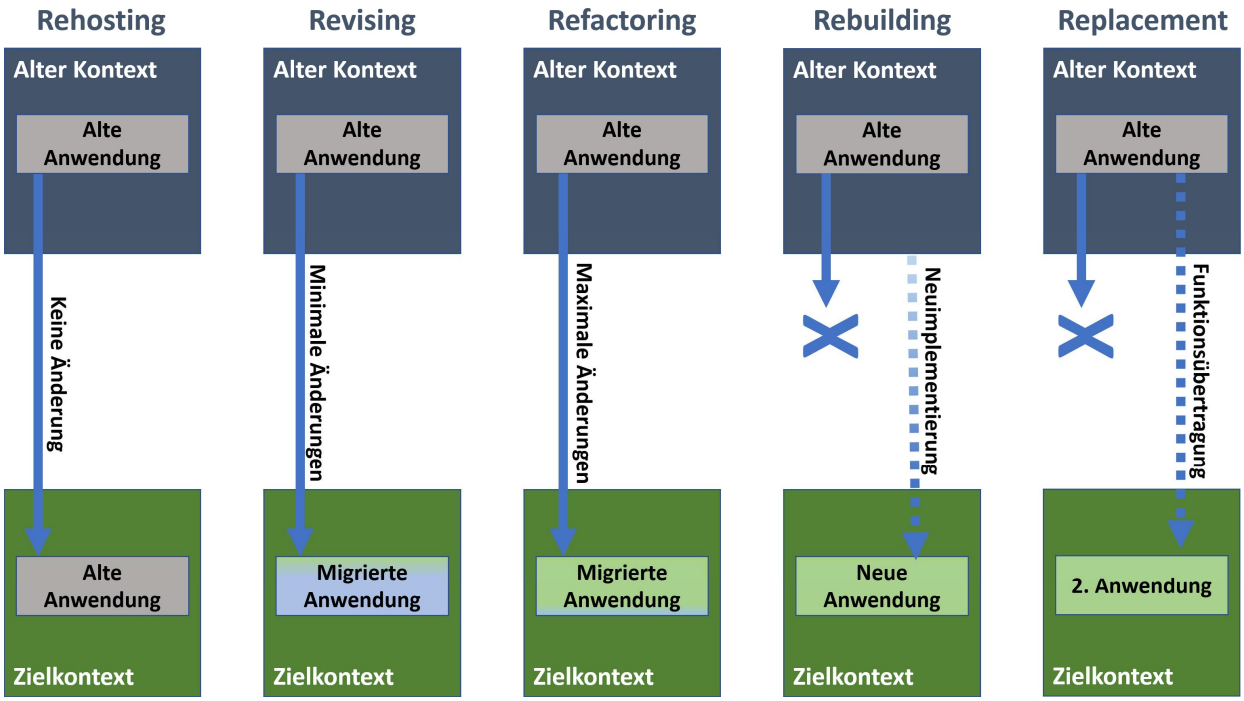

Abbildung 11: Grafische Gegenüberstellung verschiedener Migrationsarten

Die fünf gezeigten Migrationsverfahren wurden von Pettey und van der Meulen (2011) gesammelt und beschrieben. Sie unterschieden sich im Anpassungslevel an die neue Laufzeitumgebung und werden in den folgenden Abschnitten genauer erklärt.

Rehosting

Die Migrationsart mit dem geringsten Anpassungsaufwand wird *Rehosting* genannt. Bei ihr wird eine Anwendung ohne Änderungen an der Technologie oder am Aufbau auf der neuen Laufzeitumgebung betrieben (Haselmann, Hoeren, & Vossen, 2012). Wichtige Voraussetzung hierfür ist, dass der alte und der neue Systemkontext in wesentlichen Punkten identisch oder abwärtskompatibel sind.

Revising

Bei einem *Revising* wird die Anwendung im benötigten Maße umgebaut, damit sie auch in der neuen Laufzeitumgebung funktionieren kann. Funktionelle Änderungen werden bei dieser Migrationsart nicht umgesetzt. Nach Feathers (2011) können die Änderungen größere Code-Fragmente betreffen und sind damit nicht zwingend risikoarm. Ein Beispiel ist die Migration auf eine höhere Version einer Laufzeitumgebung, wie ein Upgrade auf eine höhere *Java Runtime Version*. Es erfolgen nur die notwendigen Änderungen, damit die Anwendung kompatibel zum Zielkontext ist. Allerdings sollte die gewählte Lösung den geltenden Architekturanforderungen entsprechen.

Refactoring

Das *Refactoring* geht über ein *Revising* hinaus. Die Anwendung wird nicht nur minimal an die Zielplattform angepasst, sondern so umgebaut, dass sie die Vorteile der neuen Laufzeitumgebung bestmöglich ausnutzt (Bernstein, 2015). Das bedeutet beispielsweise bei der Migration auf eine Cloud-Laufzeitumgebung, dass großer Wert auf die Erfüllung der 12-Factor-Application nach Wiggins (siehe 3.5 Cloud-Native Anwendungen) gelegt wird.

Rebuilding

Die Migrationsart *Rebuilding* ist meistens die aufwandsintensivste Migrationsart. Anwendungen werden neu aufgebaut oder im großen Stil umgebaut. Bei Um- oder Neubau wird großes Augenmerk auf die Umsetzung als Cloud native Anwendung oder Microservice gesetzt, um zum Beispiel die Vorteile einer Cloud-Umgebung nutzen zu können und neue Architekturvorgaben zu erfüllen. Optimalerweise ändert sich beim *Rebuilding* nur die interne Implementierung, nicht die Schnittstellen zu anderen Anwendungen.

Replacement

Die fünfte Migrationsart ist das vollständige Ersetzen der Anwendung. Es empfiehlt sich für Anwendungen, die perspektivisch nicht mehr genutzt werden, deren Funktionen von anderen Anwendungen übernommen werden können oder bei denen eine Migration unwirtschaftlich wäre (Heilmann, Sneed, & Wolf, 2010).

Die Möglichkeit, die Software zu ersetzen, bietet sich insbesondere bei (eigenentwickelter) Querschnittssoftware an, die durch den Einsatz von Standardsoftware abgelöst werden kann. Gut geeignet für diese Lösungsoption sind Programme, die Standardanwendungsfälle abdecken, wie Buchhaltungssysteme und Out- und Inputmanagementsysteme (Heilmann, Sneed, & Wolf, 2010). Zu beachten ist neben den Lizenzkosten der Anpassungsaufwand auf Seiten der Anwendung und der heimischen IT-Systeme, um eine Standardsoftware erfolgreich einzuführen. Dieser kann dazu führen, dass eine andere Lösungsvariante günstiger ist.

5.4 Ablauf einer Softwaremigration

Der folgende Abschnitt soll einen exemplarischen Überblick über den Ablauf einer Softwaremigration geben. In der deutschen und englischsprachigen Literatur gibt es keine einheitliche Unterteilung in verschiedene Phasen. Manche der aufgeführten Phasen sind deshalb als optional zu

betrachten. Ein Grund dafür sind beispielsweise unterschiedliche Schwerpunkte, die sich aufgrund der Kontextänderung ergeben. Das vorgestellte Vorgehen orientiert sich am *Chicken Little* Ansatz nach Brodie und Stonebraker aus dem Jahr 1993. Diese Migrationsstrategie besteht aus einzelnen, kleinen Schritten, die aufeinander aufbauen. Ein zentrales Ziel der Aufteilung in verschiedene Schritte ist es, das Risiko des Gesamtvorhabens auf einzelne Phasen zu verteilen und handhabbarer zu machen (Brodie & Stonebraker, 1993). Die Beschreibung von Brodie und Stonebraker impliziert jedoch eine vollständige Neuentwicklung der Anwendung. Zudem unterteilen die Autoren die Migration in einzelne Phasen für die Schnittstellen der Anwendung, die Anwendung an sich und die Datenbanken. Diese Trennung ist in der folgenden Beschreibung nicht vorhanden. Auch soll es das Ziel sein, möglichst weite Teile der Anwendung unverändert auf das Zielsystem zu überführen.

Entwicklung des Zielsystems

Zu Beginn eines Migrationsvorhabens wird das Zielsystem definiert. Basis hierfür ist die Zielsetzung der Migration beziehungsweise die Frage: Was soll mit der Migration erreicht werden? Diese Anforderungen werden in der Regel von Software-Architekten in ein detailliertes Zielbild überführt. Das Ergebnis kann beispielsweise eine neue Infrastrukturumgebung sein, auf der Anwendungen betrieben werden sollen, oder neue Programmiervorgaben, wie ein Service-orientierter Ansatz, dem die Anwendungen folgen sollen.

Analyse der betroffenen Anwendung

Um eine Migration zu planen und durchführen zu können, muss nach Lewis et al. (2003) ein tiefgreifendes Verständnis der betroffenen Anwendung gegeben sein.

Das Ergebnis der Analysephase sind folgende Punkte:

- Ein fachliches Verständnis der Anwendung
- Ein technisches Verständnis
- Ein Projektstrukturplan mit dem Überblick über die einzelnen Arbeitspakete
- Risiken des Vorhabens
- Eine Schätzung des Migrationsaufwandes

Der erste Ansatzpunkt ist hier in der Regel die Dokumentation, sofern sie vorhanden (und vollständig) ist. Sie sollte die Fragen beantworten, welche Aufgaben die Anwendung erfüllt (fachlich) und wie sie dabei vorgeht (technisch). Wenn die Dokumentation diesem Anspruch nicht genügt, muss für die Klärung der Fragen auf andere Methoden zurückgegriffen werden. Zur Klärung der fachlichen Funktionen bietet sich die Rücksprache mit den Nutzern der Anwendung an. Für das Verständnis der technischen Umsetzung hilft ein Reverse Engineering (Bisbal et al., 1997 und Rosenberger, 2013).

Das fachliche Verständnis wird für einen späteren Test des migrierten Programms benötigt. Das technische Verständnis muss vorliegen, um den Migrationsaufwand zu bestimmen und die Entwicklung zu planen. Nach der technischen Analyse sollen sich Arbeitspakete bilden lassen, die alle benötigten Schritte zur Umsetzung auf das Zielsystem enthalten. Diese lassen sich anschließend schätzen. Dadurch ist es nach der Analysephase möglich, dass auf der Managementebene Kosten und Nutzen des Vorhabens gegenübergestellt und beurteilt werden.

Durchführung der Migration

In diesem Schritt werden die festgelegten Arbeitspakete ausgeführt. Ziel ist es, die betroffene Anwendung im neuen Systemkontext mit unverändertem Funktionsumfang zuverlässig bereitzustellen.

Zu beachten ist, dass durch das neue Zielbild (entstanden im ersten Migrationsschritt, der Entwicklung des Zielsystems) ein Schulungsbedarf bei Entwicklern und Betriebsverantwortlichen entstehen kann. Dies ist beispielsweise beim Einsatz neuer Technologien oder neuer Infrastrukturkomponenten der Fall. Dieser Schulungsbedarf sollte vor oder zu Beginn der Migrationsdurchführung abgeschlossen werden und muss in den Projektkosten berücksichtigt werden.

Test

Nach Abschluss der Migrationsarbeiten folgt eine Testphase. Sie soll sicherstellen, dass die migrierte Anwendung die fachlichen und nicht funktionalen Anwendungen erfüllt. Während der Durchführung der Migration sollten jedoch bereits Tests stattfinden. Rosenberger (2013) führt dennoch eine dedizierte Testphase auf, um die Wichtigkeit und die verschiedenen Testdimensionen zu unterstreichen. Neben Einführungs- und Akzeptanztests werden folgende Tests typischerweise im Rahmen einer Softwaremigration durchgeführt:

- **Fachlicher Abnahmetest:** Er beantwortet die Frage, ob die migrierte Anwendung weiterhin die gleichen Funktionen erfüllt und sich identisch zur alten Implementierung verhält. Dies bezieht sich insbesondere auch auf Fehlerfälle. Als Basis kann ein Abnahmetest dienen, der mit der ursprünglichen Version der Anwendung durchgeführt wurde.

- **Performanztest:** Die migrierte Anwendung wird im neuen Systemkontext einem Lasttest unterzogen, um sicherzustellen, dass sie die Menge der eingehenden Anfragen zuverlässig abarbeiten kann (Takai, 2017). Dieser Test offenbart nicht zwangsläufig Fehler in der Anwendung, sondern dient auch dazu, die Infrastruktur zu testen, falls sie im Rahmen der Migration verändert wurde. Ein wichtiger Punkt sind insbesondere Einstellungen in der Laufzeitumgebung, die einen direkten Einfluss auf die Performance der Anwendung haben können. Ein Beispiel sind die Einstellungen für Threads und Worker, die festlegen, ob und wie die Anwendung Aufgaben nebenläufig abarbeitet (Takai, 2017).

- **Sicherheitstest:** Auf Basis einer Sicherheitsabschätzung, zum Beispiel durch einen Information Security Officer, der mit dem Projekt vertraut ist, sollte die migrierte Anwendung bei Bedarf einem Sicherheitstest unterzogen werden. Dadurch wird sichergestellt, dass durch die Migration und den geänderten Kontext keine Sicherheitslücken entstanden sind.

Produktionseinführung und Hypercare

Nach erfolgreichem Bestehen der aufgeführten Tests und der Freigabe der Anwendungsverantwortlichen kann die migrierte Anwendung in Produktion eingeführt werden.

Denkbar sind hierfür zwei Ansätze. Eine „Big-Bang“ Einführung, bei der die migrierte Anwendung an einem festgelegten Zeitpunkt alle Funktionen ihres Vorgängers übernimmt und dieser abgeschaltet wird (Bisbal et al., 1997 und Rosenberger, 2013). Als Alternative dazu bietet sich eine inkrementelle Einführung an, um das Risiko für Produktionsausfälle zu verteilen und somit die Auswirkungen zu verringern. Denkbar ist es, die Migration in mehrere inkrementelle Schritte zu unterteilen oder einzelne Anwendungsteile nur zu migrieren, wenn hierzu die Notwendigkeit besteht (Rosenberger, 2013). Ein wichtiger Aspekt, der über die Art der Einführung entscheiden kann, ist die Migration der Daten. Wenn für die Datenspeicherung eine neue oder andere Technologie eingesetzt werden soll, ist eine Big-Bang Einführung oft unausweichlich, um den Datenbestand konsistent zu halten.

An die Produktionseinführung schließt sich eine Phase der intensiven Systemüberwachung an. Sie wird *Hypercare* genannt. Ziel ist es, unvorhergesehenes Verhalten schnell zu erkennen und zügig zu beheben. Typische Probleme, die in dieser Phase erkannt werden, sind eine ungewöhnlich hohe Ressourcennutzung oder das Auftreten von Fehlern, die in der Entwicklung nicht erkannt wurden. Die Phase wird in der Regel durch Entwicklungs- und Betriebseinheiten gemeinsam durchgeführt.

Übergang in den Betrieb und Nacharbeiten

Wenn sich die migrierte Anwendung über einen gewissen Zeitraum stabil zeigt, kann sie in die regulären Betriebsprozesse eingegliedert werden. Es folgen je nach Bedarf Nacharbeiten, wie das Abschalten der ursprünglichen Anwendung oder der Abbau alter, nicht mehr benötigter Infrastruktur. Wichtig ist, dass die Aufwände für diese Arbeiten bereits bei der Analyse und Schätzung beachtet werden.

6 DIE IT-SYSTEME DER ALLIANZ DEUTSCHLAND AG

Die *Allianz Deutschland AG* ist ein Finanzdienstleister, der Privat- und Unternehmenskunden seit 1890 Produkte anbietet, um Risiken abzusichern. Die Muttergesellschaft ist die *Allianz SE*, die national aufgestellte Beteiligungsgesellschaften verwaltet und steuert. Unter ihrem Dach sind Allianz-Gesellschaften in 70 Ländern aktiv und erwirtschaften 126,1 Milliarden Euro Umsatz. (Allianz SE, 2017). Nach Umsatzzahlen ist die Allianz SE das zweitgrößte Versicherungsunternehmen der Welt (Statista GmbH, 2017).

Die *Allianz Deutschland AG*, der deutsche Ableger der Allianz Gruppe, besteht ebenfalls aus Tochterformen für die Sach-, Kranken- und Lebensversicherung sowie den Vertrieb. Zusammen haben die deutschen Allianz-Gesellschaften einen Umsatz von 32 Milliarden Euro im Jahr 2016 erwirtschaftet und beschäftigen gut 16.000 Mitarbeiter (Allianz Deutschland AG, 2016).

Die Allianz Gesellschaften, die in Deutschland operieren, haben ihre IT-Themen einheitlich in der *Allianz Deutschland AG* gebündelt. Die IT-Systeme lassen sich in folgende Kategorien einteilen:

1. Bestands- und Verwaltungssysteme
2. Online-Systeme
3. Business Intelligence
4. Organisation und Führung

Die Bestands- und Verwaltungssysteme und die Online Systeme werden in der Folge vorgestellt, um einen Überblick über die wichtigen Kernsysteme zu geben. Die Systeme für Business Intelligence, Organisation und Führung stellen unterstützenden Funktionen, zum Beispiel für die Personalverwaltung und für das Controlling, bereit. Diese sind jedoch vergleichsweise unabhängig vom Geschäftszweck des Unternehmens und werden deshalb nicht näher erläutert.

Die Online-Systeme der Allianz sollen als Praxisbeispiel für ein Migrationsszenario stehen. Sie werden im Abschnitt 6.2 im Detail beschrieben und in der Folge als Beispiel verwendet.

6.1 Die Bestands- und Verwaltungssysteme der Allianz Deutschland AG

Der Kern jeder Versicherungs-IT sind die Systeme, die die Verträge und Kundendaten verwalten. Kurz gesagt speichern sie die Informationen, mit denen das Unternehmen Geld verdient. In ihnen sind die Verträge mit den Risiken abgelegt, die abgesichert sind, und den möglichen Ansprüchen, die ein Kunde erheben kann. Sie verwalten zudem die Zahlungen, die Kunden leisten, und die gemeldeten Schadensfälle.

Aufgrund der langen Laufzeit von Versicherungsverträgen, zum Beispiel über 60 Jahre bei Lebens- und Rentenversicherungen, fällt eine große Menge an Daten an, die verwaltet und interpretiert werden müssen. Deshalb nutzt der Großteil der Versicherer selbst implementierte Softwarelösungen, die ihre Ursprünge oft in den 1950er Jahren haben. Dadurch ergibt sich auch die weite Verbreitung von Großrechnersystemen (Bock, 2015). Zudem sind in vielen Versicherungshäusern Entwicklungseinheiten vorhanden, die die IT-Systeme warten und erweitern.

Dies ist ebenfalls bei der *Allianz Deutschland AG* der Fall. Im Einsatz sind je ein Verwaltungssystem für die Branchen Sach-, Kranken- und Lebensversicherung, sowie eine Neuentwicklung, die die drei anderen Systeme zukünftig ablösen soll. Die Systeme bestehen alle aus folgenden Komponenten:

1. Eine Komponente zur Speicherung von Kunden- und Vertragsdaten
2. Einer Zugriffsmöglichkeit für Sachbearbeiter zur Pflege von Daten
3. Einer Prozesssteuerung, um Vorgänge zu steuern (beispielsweise Vertragsanpassungen, Beitragszahlungen und das Management von Schadensfällen)

Im Jahre 2005 begann die *Allianz Deutschland AG* damit, ein neues, spartenübergreifendes Bestandsverwaltungssystem einzuführen, mit dem Ziel die vorhandenen Systeme schrittweise zu ersetzen und die alte Großrechnertechnologie abzulösen. Das *Allianz Business System* (ABS) ist modular aufgebaut und hat seinen Ursprung in Österreich, wo die dort ansässige Tochtergesellschaft ein ähnliches Migrationsvorhaben begann. Die *Allianz Österreich* stellt für alle weiteren Allianz-Gesellschaften den ABS Kern bereit, auf dem die Landesgesellschaften eigene Module entwickeln und betreiben, die landesspezifische Inhalte haben.

Die Ziele dieses Migrationsvorhabens sind auf der technischen Seite, die eingesetzte Technologie zu modernisieren und zu konsolidieren und so die Kosten für Wartung und Entwicklung zu senken. Auf der organisatorischen Seite soll es ermöglicht werden, IT-gestützt eine ganzheitliche Kundenbetreuung über Spartengrenzen hinweg anzubieten.

Anfang des Jahres 2018 veröffentlichte die Allianz Pläne, Teile des ABS Kerns als Open Source Software in eine Stiftung auszulagern und zu veröffentlichen. Die Basis des ABS-Systems soll damit auch anderen Versicherern zur Verfügung stehen. Durch den Einsatz in mehreren Häusern erhofft sich der CIO Dr. Andreas Nolte sinkende Instandhaltungskosten, sowie eine stetige Weiterentwicklung durch die verschiedenen Nutzer. (Allianz SE, 2018)

6.2 Die Onlinesysteme der Allianz Deutschland AG

Die *Allianz Deutschland AG* betreibt mehrere hundert Online-Anwendungen, die verschiedenen Zwecken dienen (Allianz Deutschland AG, 2018). Die Anwendungen sind standardmäßig in der Programmiersprache Java implementiert. Grob lassen sie sich auf folgende Domänen unterteilen:

1. Services für Kunden und Interessenten, wie die Webseite *allianz.de*, die Information zu den Allianz- Produkten und Rechner zur Preisberechnung bereitstellt. Hinzu kommt das Kundenportal *Meine Allianz* mit einer Vertragsübersicht, einem Postfach und mehreren Services zur Änderung von Vertrags- und Personendaten.

2. Services für Makler und Vertreter in einem eigenen Onlineportal, wie beispielsweise die Berechnung und Verwaltung von Verträgen und Kundendaten, sowie Informationen über aktuelle Themen und Regularien.

3. Online-Anwendungen für interne Mitarbeiter zur Betreuung von Kundenanliegen sowie zur Unterstützung verschiedenster Arbeitsgebiete.

Serverinfrastruktur

Die genannten Online-Anwendungen teilen sich eine gemeinsame Serverinfrastruktur. Die folgende Abbildung 12 zeigt eine Übersicht. Diese Landschaft wird intern als *ePortale2* bezeichnet. Die Trennung in die verschiedenen Domänen Kunden, Vertreter & Makler und interne Mitarbeiter ist in der Serverlandschaft ebenfalls abgebildet.

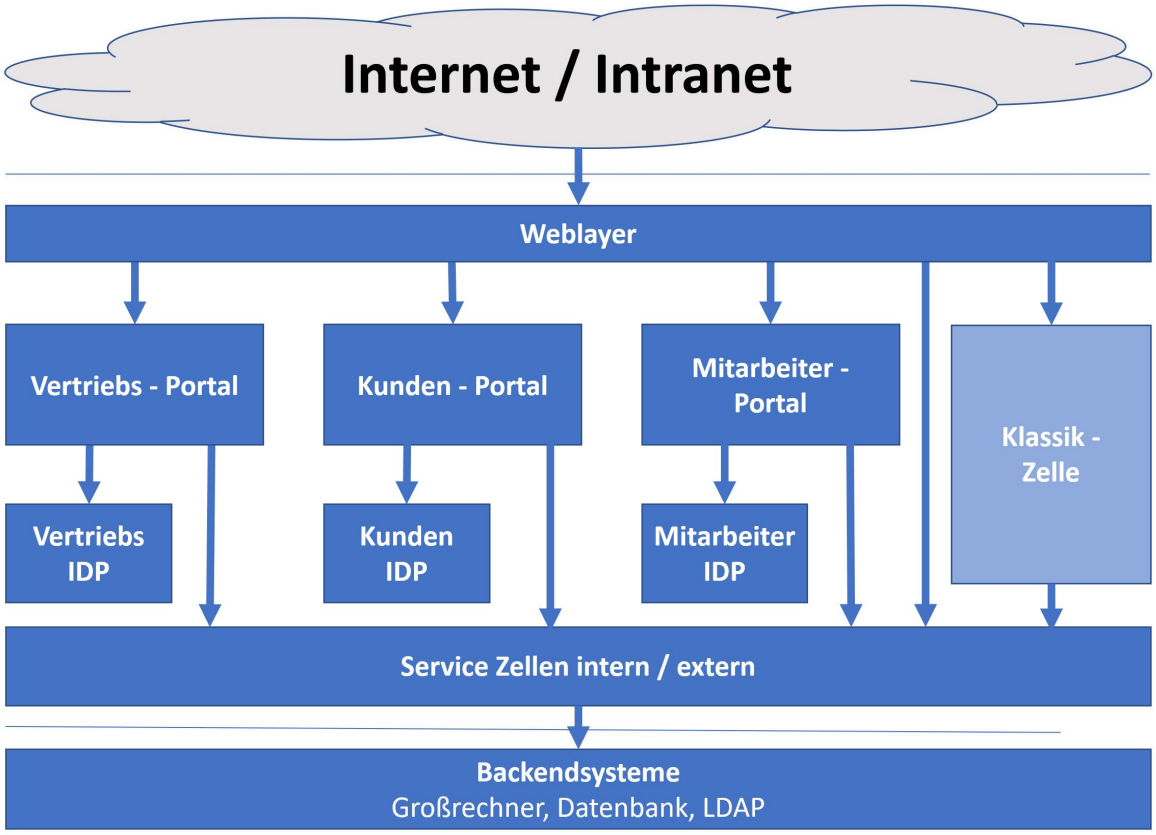

Abbildung 12: Schematische Darstellung der Serverinfrastruktur ePortale2 der Allianz Deutschland AG

Der Kern der Serverlandschaft sind die drei Portale für Vertrieb-, Kunden- und Mitarbeiter-Anwendungen. Hierfür werden 106 Instanzen von *IBM WebSphere Portal Servern* jeweils in Version 8 eingesetzt (Allianz Deutschland AG, 2018). Auf ihnen ist ein Content Management System installiert, das statische Inhalte und Anwendungen in der Form von Portlets online zugänglich macht.

Für Webservices, die keine grafische Oberfläche besitzen, sind zwei Serverzellen vorhanden. Sie unterscheiden sich in der Erreichbarkeit: eine Zelle ist aus dem Internet erreichbar, die andere Zelle beherbergt interne Services. Neben ihnen sind für jedes der drei Portale noch eigene Server für Identity Provider (IDP) vorhanden. Die Service- und IDP-Zellen bestehen aus *IBM WebSphere Application* Servern. Alle Serverzellen sind im produktiven Betrieb in zwei unabhängige Stränge unterteilt, um eine Hochverfügbarkeit und unterbrechungsfreie Deployments zu ermöglichen.

Als Weblayer kommen 58 Instanzen von *Apache HTTP Servern* zum Einsatz (Allianz Deutschland AG, 2018). Sie exponieren die DNS[3]-Endpunkte der Anwendungen und ermöglichen damit die Erreichbarkeit aus dem Internet beziehungsweise Intranet.

Eine Besonderheit ist die sogenannte Klassik-Zelle. Sie ist in Abbildung 12 auf der rechten Seite dargestellt. Sie ist ein Überbleibsel der Vorgängerplattform *ePortale1* und besteht aus *IBM WebSphere Application Server* in Version 6.1. Die *ePortale2* Infrastruktur wurde ab dem Jahr 2009 (Siemens, 2009) aufgebaut. Hierbei wurden nicht alle bestehenden Anwendungen auf die neue Umgebung migriert. Anwendungen, für die eine Migration aus Kostengründen oder wegen einer kurzen Lebenserwartung nicht in Frage kam, wurden in der Klassik-Zelle gesammelt. Sie werden bis heute auf dieser Infrastruktur betrieben und genutzt.

Die verwendete Infrastruktur wurde von einer Tochterfirma, der *Allianz Technology SE,* betrieben. Ende des Jahres 2013 kündigten die *Allianz Managed Operations and Services SE*, der Vorgänger der *Allianz Technology SE*, zusammen mit der *IBM Corporation* an, dass der Betrieb der Serverinfrastruktur innerhalb der nächsten Jahre an die *IBM* übergehen soll (Wilkens, 2013). Diese Transformation soll Ende 2018 abgeschlossen werden.

6.3 Die Klassik-Zelle im Detail

Der Vorgänger der *ePortale2* Infrastruktur (*ePortale1*), wurde zu Beginn des Webzeitalters Anfang der 2000er Jahre bei der *Allianz Deutschland AG* aufgebaut (Siemens, 2009). Sie bestand hauptsächlich aus mehreren Dutzend *WebSphere Application Server* in Version 6.1 oder in Vorgängerversionen. Neben ihnen kamen aber auch Server der Firma *Sun Microsystems* zum Einsatz, die jedoch im Laufe der Jahre abgelöst wurden. Als Basis für alle Webanwendungen der Allianz wurde ein eigenes Framework entwickelt, das *I*NET* genannt wurde. Es beinhaltet Oberflächen-Bausteine auf Basis von *JavaServer Faces* (JSF)[4] und übergreifende Services in Java, beispielsweise für die Identifikation von Nutzern und die Session-Steuerung. Im Jahr 2009 (Siemens, 2009) wurde begonnen, alte *I*NET*-Anwendungen auf die *ePortale2* Landschaft zu migrieren. Hierfür wurden die *I*NET*-Komponenten ersetzt und die Java Version entsprechend der Zielplattform erhöht. Zudem wurden die Benutzeroberflächen ersetzt und an den neuen Styleguide der *ePortale2*-Landschaft angepasst. Die folgende Abbildung 13 zeigt die eingesetzte Softwarebasis der Klassik-Anwendungen:

[3] DNS steht für Domain Name System und bedeutet die Auflösung von Domains, also Webadressen, in die zugehörigen IP-Adressen der Server.

[4] *JavaServer Faces* (JSF) ist ein Framework, das Komponenten bereitstellt, um grafische Oberflächen zu erzeugen. JSF ist Teil der *Java Platform Enterprise Edition* (JEE). Es verwendet JavaServer Pages und Servlets und beruht auf HTML. (Horn, 2014)

Abbildung 13: Software-Stack der Anwendungen der Klassik-Zelle

Der *IBM Application Server* stellt für die Anwendungen ein Java Development Kit (JDK) in Version 6.0 bereit, das von der IBM entwickelt wurde. Es entspricht weitestgehend der offiziellen Java Implementierung von *Oracle* (früher *Sun Microsystems*), enthält jedoch Erweiterungen, um auf Systemfunktionen des *WebSphere Application Server* zuzugreifen. Die Klassik-Anwendungen nutzen des Weiteren die *Java Platform Enterprise Edition*[5] in Version 1.4. Diese wurde im November 2003 veröffentlicht (Mahmoud, 2003).

Das *I*NET* Framework ist eine Allianz-eigene Implementierung, die Anfang der 2000er Jahre begonnen wurde. Das Framework besteht aus einer Vielzahl von Services und Oberflächen-Komponenten. Es diente als eine Art Baukasten, mit dem Webanwendungen implementiert wurden. Es enthält beispielsweise Services für:

- Die Konfiguration von Anwendungen
- Das Logging in der Anwendung
- Die Fehlerbehandlung
- Die Legitimation von Nutzern, sowie die Steuerung von Login und Logout

[5] Die *Java Platform Enterprise Edition* (kurz JEE oder J2EE) unterscheidet sich gegenüber der *Java Platform Standard Edition* durch eine zusätzliche Unterstützung für verteilte, hochskalierbare und sichere Netzwerkanwendungen, wie sie in Anwendungsfällen von Unternehmen benötigt werden. (Oracle Corporation, 2012)

- Den Versand von E-Mails
- Den Druck von Dokumenten als PDF oder über die unternehmenseigene Druckstraße
- Das Tracking von Nutzeraktivitäten
- Den Ablauf wiederkehrender oder regelmäßiger Funktionen (Scheduling)

120 Anwendungen der Klassik-Zelle wurde bis heute aus Kostengründen nicht auf eine moderne Plattform migriert (Allianz Deutschland AG, 2018). Sie wurden im Jahr 2009 letztmals aktualisiert und danach fachlich eingefroren (Siemens, 2009). Für sie steht bis heute eine Serverzelle bestehend aus 22 *IBM WebSphere Application Server 6.1* zur Verfügung (Allianz Deutschland AG, 2018). Änderungen wurden nur durchgeführt, sofern Probleme hinsichtlich der Sicherheit oder des Betriebs bestanden. (Siemens, 2009)

Problematisch an der Existenz der Klassik-Zelle sind insbesondere zwei Eigenschaften, die sich im Laufe der Zeit weiter verschärft haben. Einerseits werden viele Anwendungen nicht mehr weiterentwickelt. Damit schwindet das Know-how für Betrieb und mögliche Migrationen. Andererseits gewährleistet der Hersteller *IBM* nur eine begrenzte Zeit Unterstützung und Wartung für sein Produkt. Offiziell wird der *WebSphere Application Server* in Version 6.1 seit dem 30. September 2013 nicht mehr gewartet (International Business Machines Corporation, 2012). Auch Patches für Sicherheitslücken stehen seitdem offiziell nicht mehr zur Verfügung. Die *Allianz Deutschland AG* konnte mit dem Hersteller eine Wartungsverlängerung bis ins Jahr 2018 vereinbaren. Nach Ablauf dieser Frist muss die Plattform als toxisch[6] betrachtet werden. Deshalb muss ein Wechsel auf eine andere Technologie angestrebt werden.

6.4 Die Cloud-Umgebung der Allianz Deutschland AG

Die *Allianz Deutschland AG* beschloss zu Beginn des Jahres 2017, die *ePortale2* Landschaft auf eine Cloud-Infrastruktur zu migrieren. Die Basis bilden *Amazon EC2 Server*[7] mit einer Linux-Installation, die von *Amazon Webservices* bereitgestellt werden. Die *Allianz Deutschland AG* ist also Mieter einer *Infrastructure-as-a-Service* Lösung (siehe auch Abschnitt 3.3). Darauf sind zwei verschiedene Laufzeitumgebungen für Anwendungen installiert. Einerseits ist dies *Cloud Foundry*

[6] Toxisch bedeutet in diesem Fall, dass für ein genutztes Framework, eine Bibliothek oder die Infrastruktur kein Support mehr vom Hersteller oder der Community verfügbar ist. Zudem sind entweder schwerwiegende Sicherheitslücken bekannt oder es ist wahrscheinlich, dass sie in naher Zukunft gefunden werden.

[7] EC2 steht für Elastic Compute Cloud, skalierbare Server, die Amazon Web Service als Dienstleistung zur Verfügung stellt. Die Server sind mit den Betriebssystemen Linux oder Microsoft Windows Server verfügbar (Amazon Web Services Incorporation, Amazon Web Services - Amazon EC2, kein Datum)

vom Hersteller *Pivotal Incorporated*, andererseits *OpenShift* Instanzen der Firma *Red Hat Incorporated*, die *Docker Container* verwalten.

- *Cloud Foundry* ist ein *Platform-as-a-Service-Angebot* für Cloud-Umgebungen. *Cloud Foundry* ist als Open Source Software verfügbar. Die kommerziellen Angebote als *Platform-as-a-Service* werden von der Firma *Pivotal Software Incorporated* vertrieben. Die Firma verkauft zudem auch Beratungsleistungen und Implementierungsunterstützung.

  Cloud Foundry ist eine Cloud-Plattform, die das Hosting von Anwendungen übernimmt und sie skaliert. Intern kommen dabei Container zum Einsatz, in denen die Anwendungen mit einer Ablaufumgebung betrieben werden. Die zwei häufigsten Modelle sind der Betrieb einer Java Anwendung, basierend auf *Spring Boot*[8], oder einer JavaScript Anwendung auf Basis von *node.js*[9]. *Cloud Foundry* zeichnet sich durch eine einfache Benutzerschnittstelle und eine intuitive Bedienung aus. (Soroker, 2017)

  In der Cloud-Umgebung der *Allianz Deutschland AG* ist *Cloud Foundry* die Standard-Laufzeitumgebung. Im Regelfall soll ein *Apache Tomcat Server* innerhalb der Container verwendet werden, der eine Java Laufzeitumgebung zur Verfügung stellt.

- *OpenShift* ist ein Software-Projekt der *Red Hat Incorporated*, das ebenfalls quelloffen zur Verfügung gestellt wird. *OpenShift* ist wie *Cloud Foundry* eine Container-Anwendungsplattform. Sie besteht aus mehreren Subprojekten, die auch einzeln oder in andere Kontexten verwendet werden können.

  Intern werden *Docker Container* verwendet, die die Anwendungen enthalten. Sie haben ein schlankes Linux Betriebssystem als Basis. Es enthält nur die wichtigsten Funktionen und ist deshalb sehr ressourcenschonend. Weitere Funktionen können bei Bedarf dazu installiert werden. Das Linux Betriebssystem kann als Basis für unzählige Ablaufumgebungen fungieren. *Docker Container* sind damit deutlich flexibler als ihr Pendant in *Cloud Foundry*. (Weißer, 2018)

  Kubernetes ist ein ehemaliges Projekt von Google Incorporated, das inzwischen von der *Cloud Native Computing Foundation*, einer Stiftung, betreut und zur Verfügung gestellt wird (The Linux Foundation, 2018). *Kubernetes* übernimmt den Betrieb der Container in einer *OpenShift* Instanz. Das bedeutet, Kubernetes verantwortet die Steuerung, Orchestrierung und Skalierung der Container und damit die Kernpunkte einer Cloud-Umgebung.

[8] Spring Boot ist ein Teil des Spring-Framework für Java Anwendungen. Es unterstützt viele Software Pattern und nimmt Entwicklern damit (wiederkehrende) Arbeit ab. Spring Boot dient zu Implementierung eigenständiger Anwendungen und erweitert sie mit einem eingebauten Server, auf dem die Anwendungen laufen. Spring Boot wird ebenfalls von *Pivotal Software Incorporated* als Open Source Software verwaltet.

[9] Node.js ist ein offenes Framework für JavaScript Anwendungen und Webserver. Es verwendet den *V8* Compiler, der von Google stammt, und zeichnet sich dadurch aus, dass der JavaScript Code serverseitig kompiliert und ausgeführt wird.

OpenShift zeichnet sich gegenüber *Cloud Foundry* durch deutlich flexiblere Einsatz- und Konfigurationsmöglichkeiten. Das bedeutet allerdings auch ein erhöhter Aufwand auf Entwicklerseite, um eine funktionierende Umgebung zu erstellen (Weißer, 2018). Innerhalb der Cloud-Plattform der Allianz soll *OpenShift* nur für Anwendungsfälle genutzt werden, die mit *Cloud Foundry* nicht abgedeckt werden können. Für Neuentwicklungen gilt die Vorgabe, dass *Cloud Foundry* als Ablaufumgebung gesetzt ist.

Die folgende Abbildung 14 zeigt den schematischen Aufbau der gewählten Cloud-Infrastruktur der *Allianz Deutschland AG*. Er entspricht dem Konzept der *Mehrfachen Virtualisierung*, das im Abschnitt 3.4 auf Seite 13 beschrieben wird. Die Basis sind virtuelle EC2-Server, die von Amazon Web Services betrieben werden, und die darauf installierten *Cloud Foundry* und *OpenShift* Instanzen.

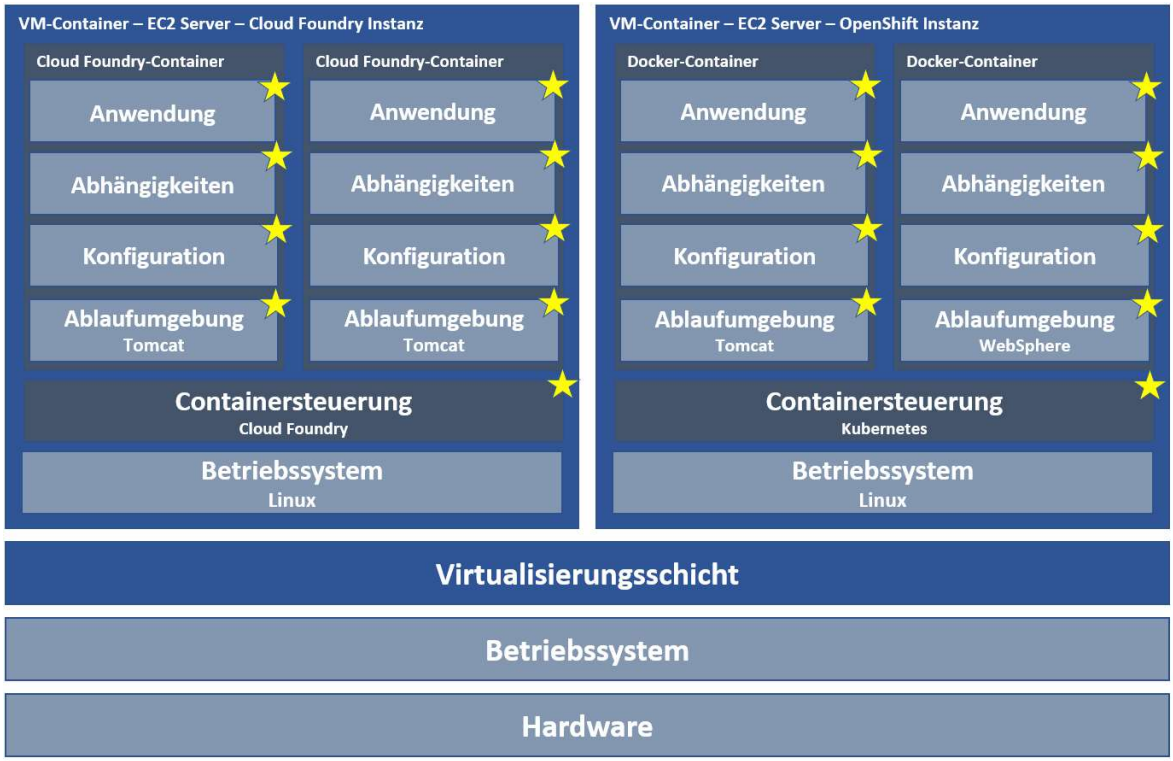

Abbildung 14: Schematischer Aufbau der Allianz - Amazon Cloud-Infrastruktur mit mehrfacher Virtualisierung

Die mit einem gelben Stern gekennzeichneten Komponenten werden in diesem Fall von der *Allianz Deutschland AG* geliefert und betrieben. Die Komponenten ohne Kennzeichen sind Produkte von *Amazon Web Services* und werden von dieser Firma betrieben und gewartet. Bis auf das Linux Betriebssystem, die Schnittstelle zum Kunden, sind für die Allianz die anderen Komponenten transparent. Die Server sind als *Virtual Private Cloud* gebündelt (siehe Abschnitt 3.2 - Cloud-Modelle). Diese Bündelung dient zur Organisation und zur Abschottung größerer Serververbünde

innerhalb der *Amazon Web Services* (Amazon Web Services Incorporation, Amazon Virtual Private Cloud - Was ist Amazon VPC?, kein Datum).

Der Aufbau innerhalb der *Virtual Private Cloud* wird in der folgenden Abbildung 15 beschrieben. Sie zeigt die Verteilung der *Cloud Foundry* und *OpenShift* Instanzen, für den sich die *Allianz Deutschland AG* entschieden hat.

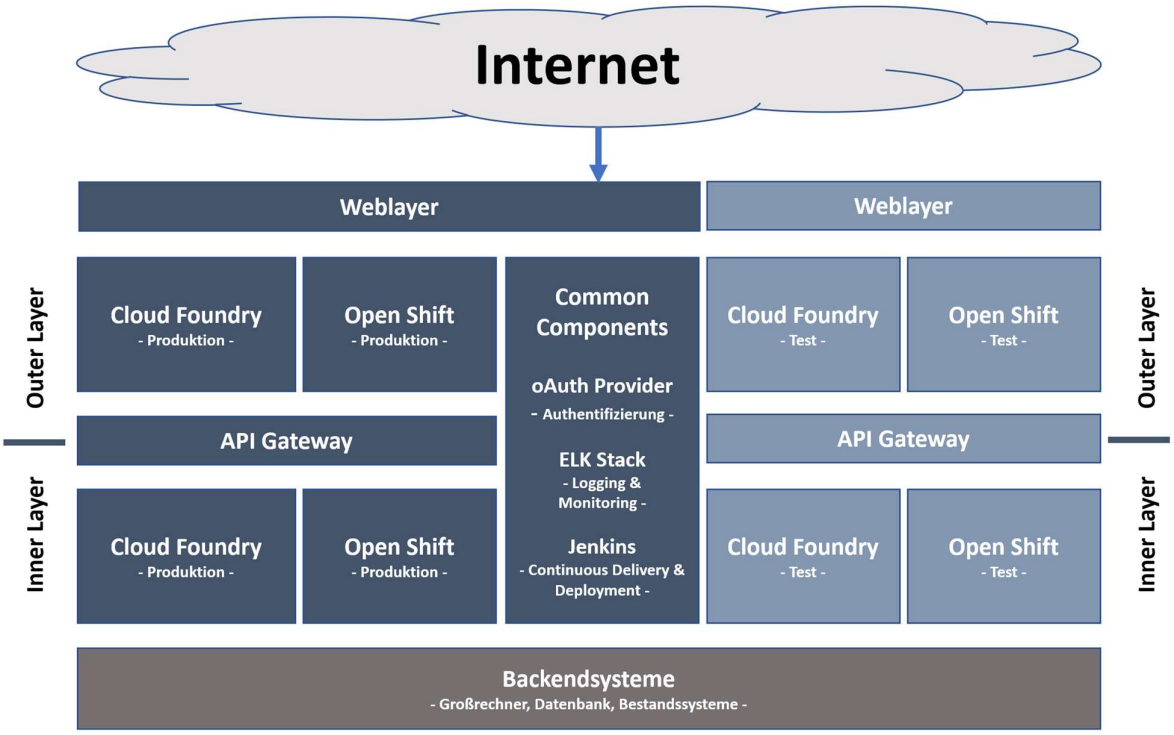

Abbildung 15: Aufbau der verschiedenen Laufzeitumgebungen in der Allianz Amazon Cloud

Kernbestandteile sind zwei Layer, die sich Outer Layer und Inner Layer nennen. In ihnen gibt es jeweils eine *Cloud Foundry*- und eine *OpenShift* Laufzeitumgebung, unterteilt in einen Testbetrieb rechter Hand und den Produktivbetrieb links. Eine Unterteilung nach verschiedenen Nutzergruppen ist nicht mehr vorgesehen. Die beiden Layer sind durch ein API-Gateway getrennt, das wichtige Sicherheitsfunktionen wahrnimmt. Anwendungen, die im Inner Layer betrieben werden, haben die Möglichkeit, Backends wie Datenbanken und versicherungstechnische Systeme zu rufen. Diese Datenbanken und Bestandsysteme werden weiterhin in einem Rechenzentrum der Allianz Gruppe betrieben. Durch diese Trennung findet die Speicherung von wichtigen Unternehmensdaten weiterhin nur im eigenen Haus und unter besonderem Schutz statt. Aus dem Internet erreichbar ist nur der Outer Layer. Vor ihm wird ein zentraler Weblayer betrieben.

Das API-Gateway schützt die Systeme des Inner Layers vor unberechtigten Zugriffen. Alle Zugriffswege sind hier im Vorhinein konfiguriert und freigeschalten. Zudem prüft das API-Gateway das Zertifikat und damit die Identität der rufenden Anwendung. Um sicherzustellen, dass Aufrufe nicht verändert werden, kommen signierte OAuth[10] Token zum Einsatz. Sie werden ebenfalls vom API-Gateway geprüft.

Für übergreifendende oder unterstützende Dienste gibt es eine separate Cloud-Zelle, die *Common Components* genannt wird. Sie ist auf Abbildung 15 mittig dargestellt und beherbergt beispielsweise den oAuth Provider, der zur Absicherung der Backend-Zugriffe dient. Für das Logging in Anwendungen wird in den Common Components ein ELK Stack betrieben. ELK steht für die Nutzung der Komponenten *Elasticsearch*, *Logstash* und *Kibana* zur Logverwaltung. Die Basis ist eine Datenbank, die Logmeldungen speichert. *Logstash* normalisiert mit Hilfe von Filtern die Logmeldungen, damit diese ein einheitliches Format haben. *Elasticsearch* dient als Suchmaschine, um die Logdateien auslesen zu können. *Kibana* kann auf Basis der *Elasticsearch* Funktionen Auswertungen erzeugen (Rohmann, 2016). Auf der Common Components Zelle hat auch der Build-Server *Jenkins* seine Heimat. Er ist der zentrale Service der neuen Toolchain, die mit der neuen Cloud-Umgebung bei der *Allianz Deutschland AG* eingeführt wird. Sie beinhaltet alle Systeme, die benötigt werden, um eine Änderung am Quellcode auf die Test- und Produktionsumgebung zu spielen. Die folgende Abbildung 16 zeigt die verwendeten Systeme und ihre Verbindungen untereinander. Die bestehende Toolchain der *ePortale2* Welt wird durch sie vollständig ersetzt. Es werden keine Systeme wiederverwendet.

[10] OAuth steht für Open Authorization, ein offener Protokollstandard, der es Nutzern erlaubt, Autorisierungen an andere Anwendungen zu vergeben (Hofmann, 2012). Bei der Allianz Deutschland werden hierfür JWT Token (JSON Web Token) verwendet, die aufgrund der enthaltenen Signatur nicht verändert werden können. Grundlage hierfür ist der Request for Comments 7519 der Internet Engineering Task Force (JWT.io, kein Datum). Sie dienen dazu, eine Nutzeridentität sicher in alle Backends zu transportieren, ohne dass sie auf dem Weg manipuliert werden kann.

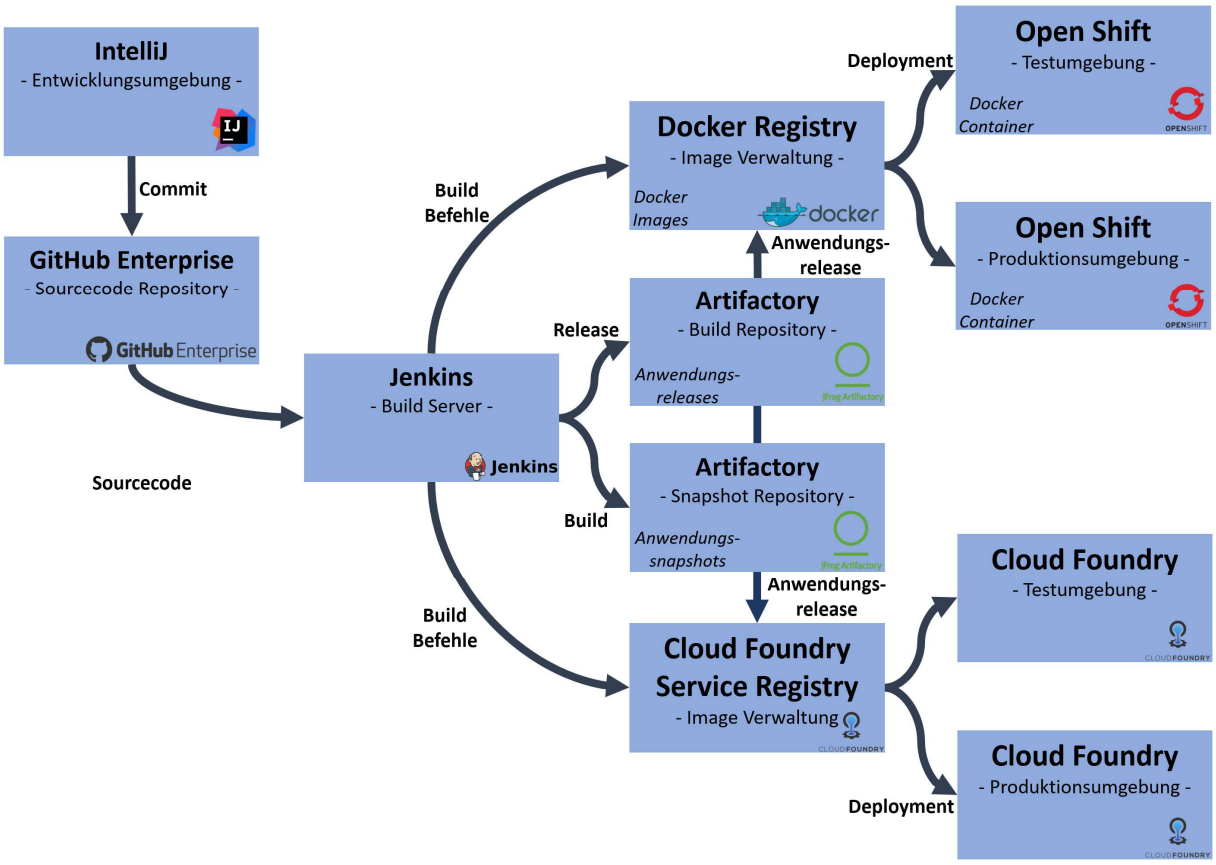

Abbildung 16: Die neue Toolchain der Allianz Deutschland AG

Entwicklungsumgebung: Als Entwicklungsumgebung kommt *IntelliJ* zum Einsatz, ein Produkt der tschechischen Firma *JetBrains*. Mit *IntelliJ* bearbeiten die Softwareentwickler den Quellcode, führen lokale Tests aus und können Quellcode aus dem zentralen Repository abholen oder hinzufügen.

Sourcecode Repository: Das zentrale Repository ist eine *GitHub Enterprise* Installation. *GitHub Enterprise* ist eine Variante des kollaborative Versionsverwaltung *Git*. Es enthält Besonderheiten für Firmenkunden und professionelle Nutzer (GitHub Incorporated, kein Datum). Es verwaltet alle Versionsstände der Anwendungen der *Allianz Deutschland AG*. Gegenüber seinem Vorgänger bei der Allianz, SVN Subversion, zeichnet sich GitHub durch einen starken kollaborativen Aspekt aus. Neben der Quellcodeverwaltung bietet es die Möglichkeit Wiki-Seiten anzulegen und so die Dokumentation von Anwendungen direkt mit dem Quellcode zu verknüpfen. Außerdem können Entwickler anderen Anwendungen Änderungsvorschläge über *GitHub Enterprise* mitteilen.

Build Server: *Jenkins* ist ein webbasiertes Programm zur kontinuierlichen Integration und Deployments von Anwendungen (siehe auch Abschnitt 3.8 dieser Arbeit) (Jenkins User Documentation, kein Datum). Es ist der zentrale Build Server, der Releases von Anwendungen

baut und in *Artifactory* Instanzen speichert. Jenkins steuert auch die Deployments von Anwendungsreleases. Dafür führt *Jenkins* Befehle auf den *OpenShift* und *Cloud Foundry* Instanzen aus. Von dort werden die Container auf die verschiedenen *OpenShift* Instanzen deployt. Auf diese Art werden *Zero Touch Deployments* ermöglicht, also Deployments ohne manuelle Schritte. So werden händische Fehler vermieden, die Prozesse können ohne Spezialwissen angestoßen werden und sind revisionssicher.

Build Repository: *Artifactory* ist ein Repository zur Verwaltung von Build-Ergebnissen (JFrog Limitied, 2018). Es speichert alle Releases und Snapshot- (Test) Versionen von Anwendungen und hält diese bereit, wenn sie auf einer Ablaufumgebung installiert werden sollen.

Zusammengefasst zeigt sich, dass sich die neue Cloud-Umgebung fundamental von der bisherigen Infrastruktur unterscheidet. Für die Migration der aktuellen Komponenten auf *ePortale2* liegen innerhalb der *Allianz Deutschland AG* Pläne vor, wie der Umzug auf die Cloud-Infrastruktur gestaltet werden soll. Sie werden auf *Spring Boot* umgestellt, damit sie unter *Cloud Foundry* lauffähig sind und werden dann auf eben dieser Plattform betrieben. Der folgende Teil der Arbeit soll beispielhaft zeigen, ob und wie auch die Legacy-Anwendungen der Klassik-Zelle migriert werden können.

7 PLANUNG DER MIGRATION DER KLASSIK-ZELLE

In diesem Kapitel werden die Schritte beschrieben, die benötigt werden, um Legacy-Anwendungen auf eine moderne Infrastruktur zu überführen. Das Migrationsvorhaben der *Allianz Deutschland AG* soll dabei als durchgehendes Praxisbeispiel herangezogen werden. Das Vorgehen lehnt sich an den *Chicken Little* Ansatz von Brodie und Stonebraker an, wie er in Abschnitt 5.4 beschrieben ist. Der Kerngedanke, die Migration in kleine Schritte aufzuteilen, wurde mit der Maßgabe aus dem agilen Manifest kombiniert, dass oft funktionierende Software ausgeliefert werden soll, die von Schritt zu Schritt Verbesserungen und Erweiterungen enthält, und die immer lauffähig ist (Beck, et al., 2001).

Der wichtige Grund für die Migration ist die Tatsache, dass die Anwendungen – entgegen den Prognosen von 2009 auch ohne fachliche Änderungen – weiterhin produktiv genutzt werden. In den Jahren seit 2009 wurden wenige Anwendungen ersetzt oder abgelöst. Die noch genutzten Anwendungen werden oft für interne Zwecke verwendet oder bilden Anwendungsfälle ab, die von Kunden und Vertretern vergleichsweise selten genutzt werden.

7.1 Entwicklung des Zielsystems

Zu Beginn einer Migration muss die Zielsetzung des Vorhabens festgelegt und dokumentiert werden. Ein besonderes Augenmerk muss auf die Kernpunkte des Migrationsvorhabens gelegt werden. Die Anforderungen an die migrierten Anwendungen müssen genau festgelegt werden, um den Erfolg des Vorhabens messen zu können. Neben den technischen Aspekten müssen weitere Rahmenbedingungen geklärt sein, die üblicherweise in einem Projektauftrag definiert werden.

Im Fall einer Migration der Ablaufplattform ist es wichtig, die technischen Details festzuhalten. Gibt es bereits eine Beschreibung für ein Zielsystem, weil es bereits vorhanden ist oder ist das Ziel des Migrationsvorhabens schlichtweg alte Infrastruktur abzuschaffen, ohne Vorgabe, wie ein neues System auszusehen hat? Die Beschreibung des Zielsystems muss beinhalten, welche Softwareversionen verwendet werden sollen, möglichst mit dem Hinweis, welche alte Technologie sie ablösen sollen. Beim Wechsel der Laufzeitumgebung ändert sich unter Umständen nicht nur diese direkte Abhängigkeit. Auch geänderte Anforderungen, beispielsweise in Bezug auf die Sicherheit, das Monitoring oder das Logging in der Anwendung, können weitere Anpassungen notwendig machen, da diese Punkte in Legacy-Anwendungen meist nicht auf dem Stand der Technik sind.

Aus der Praxis:

Die Migration der Klassik-Zelle der *Allianz Deutschland AG* soll folgende Zielsetzungen verfolgen. Die Ziele sind nach Priorität geordnet.

- **Ersatz alter Technologie:** Das Hauptziel ist die Ablösung der toxischen *IBM WebSphere Application Server 6.1*, um das Sicherheitsniveau zu erhöhen. Hierfür wurde ein Stichtag gesetzt, da das Risiko durch den Einsatz der Technologie nicht auf unbestimmte Zeit getragen werden kann.

- **Weiternutzung der Anwendungen:** Ein weiteres Ziel ist es, die Anwendungen der Klassik-Zelle ohne Unterbrechung oder Verschlechterung der Service Level auf die Cloud-Infrastruktur umzuziehen, die im Abschnitt 6.4 vorgestellt wurde. Damit trägt die Migration dazu bei, das alte, proprietäre Rechenzentrum abzulösen. Alternativ können die Anwendungen auch außer Betrieb genommen werden, wenn ihre Funktionen nicht weiter benötigt werden oder anderweitig ersetzt werden können.

- **Senkung der Betriebskosten:** Die Migration soll zukünftig Betriebskosten einsparen, weil moderne Technik zum Einsatz kommt, die einem unternehmensweiten Standard entspricht. Teures und seltenes Spezialwissen für den Betrieb der veralteten *WebSphere* Umgebung wird nicht weiter benötigt.

- **Kosten:** Die Migration soll mit dem Einsatz möglichst geringer Mittel erfolgen.

Ein beispielhafter Projektauftrag für das Migrationsprojekt der Allianz kann wie folgt aussehen. Er orientiert sich an einer Vorlage des *Projektmanagement Handbuchs* (Hagen Management GmbH, kein Datum). Einige Punkte, die für diese Arbeit unerheblich sind oder im Rahmen dieser Arbeit nicht behandelt werden können, wurden entfernt.

Projekttitel:	Ablösung der WebSphere Application Server 6.1 Infrastruktur der Klassik-Zelle

A. Projektdaten			
Start:	07/2017	**Projektkategorie:**	Großprojekt
Ende:	12/2018	**Projektnummer:**	tbd

B. Projektorganisation	
Projektauftraggeber:	Das Management der Allianz Deutschland AG
Umsetzende:	Ein zentrales Migrationsteam, mehrere Anwendungsverantwortliche mit Entwicklungsteam

C. Projektbeschreibung	
Ausgangssituation / Projektbegründung:	Auf der Klassik-Zelle, einer Serverzelle bestehend aus mehreren IBM WebSphere Application Servern in Version 6.1, werden 120 Altanwendungen betrieben. Der Hersteller der Server wird in Zukunft den Support für sein Produkt nicht weiterführen. Die Anwendungen müssen deshalb auf eine andere Ablaufumgebung migriert werden.
Projektgesamtziel:	Abschaltung aller WebSphere Application Server 6.1 Installationen ohne Verlust von Funktionen. Betrieb der betroffenen Anwendungen auf der neuen Cloud-Infrastruktur.
Projektteilziele	
1. Ablösung toxischer Technologie / Erhöhung des Sicherheitsniveaus	
2. Nutzung der Cloud-Infrastruktur / Ablösung des alten Rechenzentrums	
3. Einsparung von Betriebskosten durch Einsatz moderner, standardisierter Software	
4. Einsatz möglichst geringer Mittel zur Migration	
Nicht-Ziele:	Anwendungen, die nicht auf der WebSphere Application Server 6.1 Infrastruktur beheimatet sind, werden nicht betrachtet
Wirkung / Nutzen:	Senkung eines bekannten Sicherheitsrisikos, Nutzung neuer Technologie
Projektphasen / Hauptaufgaben:	1. Bestandsaufnahme und Analyse 2. Renovierung der Anwendungen 3. Cloudifizierung der Anwendungen 4. Inbetriebnahme und Überwachung
Projektrisiken:	• **Qualitätsrisiken**: Fachliche Anforderungen sind nicht vollständig bekannt und werden möglicherweise erst später erkannt • **Technische Risiken**: Neue Technologie ist im Unternehmen wenig bekannt, Ansprechpartner für alte Technologie fehlen • **Terminrisiken**: Altanwendungen sind nicht bis in Details bekannt, Migrationsumfang ist möglicherweise nicht vollständig bekannt

D. Projektbudget & Wirtschaftlichkeit[11]

E. Projektkategorisierung	0	1	2	3
strategische Bedeutung	☐	☐	☐	☒
Risikogehalt	☐	☐	☒	☐
Komplexitäts- / Schwierigkeitsgrad	☐	☐	☐	☒

[11] Das Budget für dieses umfangreiche Projekt kann im Rahmen dieser Arbeit nicht abgeschätzt werden. Aufgrund der Vielzahl der Anwendungen muss eine Schätzung mit Hilfe aller betroffenen Anwendungsteams erfolgen.

Neuartigkeitsgrad	☐	☐	☐	☒
Termindruck	☐	☐	☒	☐
Klarheit über Projektziele / Kundenanforderungen	☐	☒	☐	☐

0 = sehr gering, 3 = sehr hoch

F. Sonstiges	
Sonstige Informationen:	Annahmen und Beschränkungen sind Inhalt der folgenden Abschnitte und deshalb hier nicht aufgeführt

Tabelle 2: Beispielhafter Projektauftrag für das Migrationsprojekt der Allianz nach Vorlage des Projektmanagement Handbuchs

Bislang ist noch keine genaue Festlegung hinsichtlich einer konkreten Laufzeitumgebung getroffen worden. Für die migrierten Anwendungen der Klassik-Zelle ergeben sich drei Möglichkeiten unter Beachtung der Bedingung „Nutzung der vorhandenen Cloud-Infrastruktur“:

1. Die Nutzung der *Cloud Foundry* Installationen
2. Die Nutzung eines *Docker Container*s innerhalb der *OpenShift* Installationen
3. Die direkte Nutzung von EC2-Servern

Die dritte Option soll aus Betriebs- und Kostengründen nicht gewählt werden und wird deshalb in der Folge nicht mehr betrachtet. Die Architekturvorgaben der *Allianz Deutschland AG* geben vor, dass *Cloud Foundry* als bevorzugte Laufzeitumgebung gewählt werden soll. Alternativ kann beim Vorliegen besonderer Gründe die *OpenShift* Umgebung genutzt werden. Eine direkte Nutzung der EC2-Instanzen wird von den Betriebseinheiten nicht unterstützt und ist deshalb nicht vorgesehen.

Optimalerweise wird die Entscheidung im Projekt möglichst früh getroffen, damit der Handlungsbedarf im nächsten Schritt genauer bestimmt werden kann. Den Anwendungsteams können jedoch auch mehrere Optionen zur Verfügung gestellt werden. In diesem Fall empfehlen sich zwei Optionen.

1. **Refactoring:** Die erste Option ist ein Refactoring (siehe Abschnitt 5.3 - Arten von Softwaremigration), also die bestmögliche Anpassung an das optimale Zielbild in der Allianz Cloud. In diesem Fall ist das die Nutzung einer *Cloud Foundry* Installation, da sie als Standard in der Allianz Cloud-Infrastruktur gesetzt ist. Zum Einsatz käme in diesem Fall der übliche Buildpack – der Einsatz eines *Apache Tomcat Application Server*, der in den *Cloud Foundry* Containern läuft, und die Umstellung der Anwendung auf die Nutzung von *Spring Boot*. Die erste Option empfiehlt sich für Anwendungen, die voraussichtlich eine längere Lebensdauer haben. Um sie zu nutzen, sind erhöhte Anpassungsaufwände zu erwarten.

2. **Revising:** Die zweite Option ist ein Revising (ebenfalls beschrieben im Abschnitt 5.3). Sie basiert auf der Idee, den Sprung zwischen der vorhandenen Infrastruktur und der Zielumgebung möglichst klein zu halten. Im vorliegenden Fall wäre dies der Einsatz einer aktuellen *IBM WebSphere Application Server* Installation. Die aktuelle Version, und damit der nächstliegende Ansatz, ist der Einsatz des *WebSphere Application Server 9*, der seit Juni 2016 verfügbar ist. Eine mögliche Alternative dazu kann der leichtgewichtigere *IBM WebSphere Liberty* sein. Da letzterer kostenlos verfügbar ist, sollte diese Option bevorzugt untersucht werden. Zudem läuft der IBM WebSphere Liberty in *Docker Containe*m und ist somit kompatibel mit der Zielinfrastruktur. (Weißer, 2018)

Ungeachtet davon bietet sich ebenfalls die Möglichkeit, die Anwendungen zu ersetzen (*Replacement*) oder neu zu implementieren. Wenn ein kostengünstiger Ersatz für einzelne Anwendungen vorhanden ist, sollte dieser Weg allein aus Kostengründen gewählt werden. Die Neuimplementierung (*Rebuilding*) sollte ebenfalls nur gewählt werden, wenn eine Migration per *Refactoring* oder *Revising* teurer wäre als die Neuimplementierung.

Für die Umsetzungsteams ist es wichtig, in diesem Fall eine klare Priorisierung vorzugeben. Das Hauptziel ist die zügige Ablösung der toxischen Infrastruktur. Deshalb empfiehlt sich Option 2 zu wählen, da der technologische Schritt kleiner erscheint und damit schneller durchführbar ist. Eine Weiterentwicklung zu Option 1 ist denkbar, wenn sich Anwendungsteams bewusst dafür entscheiden. Gegebenenfalls bietet es sich auch an, die Option 2 als Zwischenschritt zur Umsetzung des Zielbilds (Option 1) umzusetzen.

Die Option 1 kann gewählt werden, wenn die Anwendungen keine direkten Abhängigkeiten zu Systemfunktionen des *WebSphere Application Server* besitzen oder wenn diese einfach ausgebaut werden können. Laut einer Untersuchung des Unternehmens *MuleSoft* (MuleSoft Corporation, kein Datum) sind die Unterschiede zwischen *WebSphere* oder Tomcat inzwischen auf technischer Eben sehr gering. Die beiden Lösungen unterscheiden sich stattdessen eher in den Dimensionen Kosten, Support und Service.

Eine genaue Festlegung der Migrationspfade findet nach der Analyse der Anwendungen in Abschnitt 7.2 statt.

Berücksichtigung der 12 Factor-Apps

Wenn aus Altanwendungen mit einer Lebenszeit von teilweise mehr als zehn Jahren Cloud-Anwendungen werden, müssen weitere Anpassungen erfolgen. Zur sinnvollen Nutzung der Cloud-Umgebung müssen die Anwendungen einige Punkte von Adam Wiggins *Twelve-Factor-Application* erfüllen (siehe Abschnitt 3.5).

Codebasis und Toolchain

Wiggins schreibt für cloud-native Anwendungen die Verwendung eines *Sourcecode-Management-Systems* vor. Dies ist auch für die Migration der Allianz-Anwendung zu empfehlen. Die migrierten Anwendungen sollen sich in die bestehende Toolchain einfügen und die bereits vorhandenen Hilfsmittel nutzen. Im Falle der Klassik-Zellen-Anwendungen bedeutet das, eine Migration des Sourcecodes auf die *GitHub Enterprise*-Installation der *Allianz Deutschland AG*. Zudem soll zur Build-Automatisierung *Jenkins* verwendet werden, ebenso wie *Artifactory* zur Verwaltung der Build-Ergebnisse genutzt wird. Dies führt zu einem einheitlichen Werkzeugset, das für alle Anwendungen benutzt wird. So entstehen keine Sonderlösungen, die spezielles Wissen erfordern. Zudem ist damit ein Grundstein für mögliche weitere Migrationen gelegt.

Konfiguration

Innerhalb der Anwendungen müssen die Mechanismen für die Konfiguration angepasst werden. Konfigurationseinträge müssen aus dem Sourcecode entfernt werden, gesammelt und als Umgebungsvariable abgelegt werden (Wiggins, Konfiguration, 2017). Der Hintergrund dieser Forderung von Wiggins ist die Fähigkeit, Container von einer Abnahmestufe auf eine andere zu verschieben, ohne in den Containern eine Änderung vornehmen zu müssen. Alle Werte, die sich zwischen den Abnahmestufen unterscheiden, müssen explizit in Umgebungsvariablen ausgelagert werden.

Insbesondere bei Altanwendungen kann für diese Arbeit erheblicher Aufwand anfallen. Je nachdem in welcher Form die Anwendungskonfiguration bislang erfolgt ist, kann es notwendig sein, dass sich Entwickler mit großen Teilen des Sourcecodes intensiv auseinandersetzen, um Konfigurationseinträge zu identifizieren.

In der Praxis ist zudem eine besondere Behandlung von geheimen Informationen notwendig. Hierunter fallen beispielsweise Passwörter, die für technische Benutzer hinterlegt sind, oder die Zertifikate innerhalb der Anwendung schützen. Es ist darauf zu achten, dass diese Informationen nicht im Klartext abgelegt werden. *OpenShift* und *Cloud Foundry* unterstützen beide den Umgang mit verschlüsselten Konfigurationswerten (Spazzoli, 2017 und Cloud-Foundry Foundation, 2017). Zur Entschlüsselung wird beim Start der Plattform ein privater Schlüssel übergeben, der die Werte lesbar macht.

Trennung von Build-, Release und Run-Phase

Die Trennung der drei Phasen *Build*, *Release*, und *Run* wird durch die Nutzung der vorhandenen Toolchain sichergestellt (siehe Abbildung 16 auf Seite 52). Durch die vorgeschriebene Nutzung von Buildservern und festgelegte Deploymentabläufe, können die drei Phasen nicht vermischt oder übersprungen werden. Die Vorgabe von Wiggins wird damit erfüllt.

Port Binding

Wenn sich die Anwendung an Ports bindet, sind diese nicht im Anwendungscode vorgegeben, sondern werden über Platzhalter aus Umgebungsvariablen aufgelöst. Dadurch kann das Port Binding von außerhalb gesteuert und kontrolliert werden. Für *Cloud Foundry* und *OpenShift* muss diese Möglichkeit gegeben werden, damit über die Definition von Routen die Anwendung ansprechbar wird.

Logging

Das Logging in der Anwendung muss angepasst werden, damit es in Kombination mit der neuen Laufzeitumgebung funktionieren kann. Die Anwendungen dürfen keine Logmeldungen mehr in feste Textdateien schreiben, sondern müssen diese an die Ablaufumgebung übergeben. Nur so sind Logmeldungen verfügbar, wenn Container ersetzt werden. Zudem sollten die Anwendungen ein einheitliches Logformat verwenden, um zur Logverwaltung einen ELK-Stack einsetzen zu können. Die Umsetzung dieser Maßnahme ist zwingend erforderlich, da die Anwendungen sonst nicht betrieben werden können. Die Anforderung kann auf zwei Arten umgesetzt werden: einerseits kann das Logging global umgebaut werden, damit es die Anforderungen erfüllt. Andererseits kann versucht werden, das bestehende Logging weiter zu nutzen und lediglich das Ziel der Logeinträge passend zu verändern.

Weitere Anforderungen im Rahmen der Allianz Deutschland AG

Diagnostizierbarkeit

Alle Anwendungen sollen mit einem Healthcheck, einer Statusabfrage, ausgestattet werden. Die Abfrage soll Informationen über die Verfügbarkeit der Anwendung liefern. Gegebenenfalls bezieht sich der zurückgegebene Status auch auf die Verfügbarkeit der benötigten Backendsysteme, um eine Aussage zur Verfügbarkeit ganzer Geschäftsvorfälle treffen zu können. Die Healthchecks sollen von einer Allianz eigenen Anwendung regelmäßig abgefragt werden. Diese Anwendung bündelt die Rückmeldungen, stellt eine Übersicht bereit und alarmiert auf verschiedenen Wegen Anwendungsteams und das Management, wenn festgelegte Schwellwerte überschritten werden.

Die Anforderungen zum Einbau eines Endpunkts für den Healthcheck ist ein Standard innerhalb der Anwendungsentwicklung der *Allianz Deutschland AG* und festgehalten in den übergreifenden Architekturrichtlinien (Allianz Deutschland AG, Referat IT-Architektur, 2018).

Der Einbau von Healthchecks wird auch aus technischen Gründen benötigt. Die Cloud-Umgebungen benötigen eine Schnittstelle, mit der sie den Zustand und die Erreichbarkeit der Services in ihren Containern prüfen können (Weißer, 2018). Im Gegensatz zu den Healthchecks, die die Architekturrichtlinien der *Allianz Deutschland AG* vorschreiben, sollen diese Schnittstellen nur Auskunft über die Anwendung an sich geben, ohne die Einbeziehung von Backendsystemen.

Sicherheit

Die folgenden Anforderungen müssen Anwendungen ebenfalls zwingend erfüllen, da sie sonst nicht den Sicherheitsrichtlinien der *Allianz Deutschland AG* entsprechen. Eine Nichteinhaltung kann zur Verweigerung der Produktionsfreigabe führen.

- Alle Verbindungen zu anderen Anwendungen müssen verschlüsselt erfolgen. Darunter fallen Verbindungen zu Datenbanken, Webservices und Backend-Systemen. Die Vorgabe gilt für alle Protokolle, auch für solche, die vergleichsweise selten eingesetzte werden, wie beispielsweise CORBA (Common Object Request Broker Architecture). Beim Einsatz von TLS (Transport Layer Security früher SSL) soll die neueste Version TLS 1.2 eingesetzt werden.

- Beim Einsatz von TLS hat dies als 2-way-TLS zu erfolgen. Das bedeutet, dass sich nicht nur der gerufene Server mit einem Zertifikat identifiziert, sondern auch der rufende Service. Der aufgerufene Service muss die Identität seiner Rufer kennen und anhand einer Access Control List verwalten. Der Service weiß also, welche Rufer welche Funktionen nutzen dürfen. (Peeples, 2015)

- Vor dem Einsatz jeder Anwendung, die aus dem Internet erreichbar ist, muss zwingend ein Penetrationstest durch einen externen, unabhängigen Dienstleister erfolgen. Für Anwendungen, die nur intern genutzt werden, entscheidet ein Project Security Officer fallabhängig über die Notwendigkeit eines Penetrationstests. Durch diesen Test soll sichergestellt werden, dass die Anwendungen typischen Angriffsszenarien wie SQL-Injection, Cross-Site-Scripting und Phishing widerstehen kann.

Resilienz

Innerhalb der *Allianz Deutschland AG* gibt es eine weitere Anforderung, auf die großen Wert gelegt wird. Die Anwendungen sollen, wenn möglich, resilient gestaltet werden. Das heißt, beim Ausfall eines oder mehrerer Backendsysteme wird versucht, den Geschäftsvorfall dennoch abzuschließen.

Ein Beispiel aus der Praxis: Beim Stellen eines Antrags wird üblicherweise die Postadresse über einen Webservice validiert. Fällt dieser Service aus, wird der Antrag dennoch weiterverarbeitet, erhält aber den Hinweis, dass die Adresse ungeprüft ist. Er wird ausgesteuert und manuell nachbearbeitet. Der Nutzer erhält jedoch keine Fehlermeldung und muss den Prozess nicht erneut starten.

Die migrierten Anwendungen der Klassik-Zelle sollen dieses Prinzip unterstützen, wenn es sich im Rahmen der Migration sinnvoll und kostengünstig einsetzen lässt. Zur Realisierung können Bibliotheken wie *Hystrix* von *Netflix* genutzt werden (Netflix Incorporated, 2017). Die Bibliothek

ermöglicht es, Funktionen zu kapseln und einen Fallback zu definieren, der ausgeführt wird, falls die Funktion fehlschlägt. Gleichzeitig dient die Bibliothek dazu, kaskadierende Fehler in verteilten Systemen zu vermeiden, indem fehlerhafte Aufrufe blockiert werden. Damit können sie sich nicht auf andere Systeme auswirken.

7.2 Analyse der betroffenen Anwendung

An die Entwicklung des Zielbildes des Migrationsvorhabens schließt sich die Analyse der Anwendungen an. Das Ziel ist es, ein Bild über die Arbeitspakete zu erhalten, die umgesetzt werden müssen, um das definierte Zielbild zu erreichen. Da sich beide Phasen gegenseitig beeinflussen, müssen sie nicht zwingend getrennt nacheinander durchlaufen werden.

Für das Gelingen eines Migrationsvorhabens ist es essentiell wichtig, mit einer genauen Analyse der betroffenen Systeme einen guten Grundstein zu legen. Dadurch wird der Arbeitsumfang möglichst genau abschätzbar und Risiken werden erkannt und dadurch gemindert. Die Analyse dient auch dazu, bei mehreren alternativen Migrationsszenarien das passende Vorgehen auszuwählen.

In einem ersten Schritt müssen die betroffenen Anwendungen identifiziert werden. Für alle Anwendungen müssen technische und fachliche Ansprechpartner und der Quellcode gefunden werden. Optimalerweise gibt es auch eine Dokumentation, die die Anwendung und deren Zweck beschreibt. Wichtig ist ebenfalls, die Abhängigkeiten der Anwendung zu identifizieren. Bei der Migration von Legacy-Anwendungen zeigt sich bei der Informationsbeschaffung oft die erste große Hürde.

Im Fall einer Servermigration ist die verlässlichste Quelle zur Suche nach betroffenen Anwendungen die abzulösenden Server. Sie können eine Übersicht von allen installierten Anwendungen liefern. Allerdings liefert der Server in der Regel nur die Anwendungsnamen, ohne weitere Informationen. Wichtig ist, dass diese Abfrage dennoch auf allen Abnahmestufen durchgeführt wird und die Ergebnisse verglichen werden. In seltenen Fällen ist es denkbar, dass Anwendungen nur auf Testsystemen installiert sind, wenn sie beispielsweise der Bereitstellung von Testdaten dienen. Diese Anwendungen dürfen bei einer Migration nicht übersehen werden.

Die Liste der Anwendungsnamen kann mit Hilfe weiterer Systeme um zusätzliche Informationen ergänzt werden. Eine naheliegende Quelle ist ein Deployment-Management-Tool, sofern es existiert. Neben den Anwendungsnamen besitzt es in der Regel Kenntnis über Ansprechpartner, den Ablageort der gebauten Releases und möglicherweise den Ablageort des Sourcecodes. Wenn kein Werkzeug für das Management von Deployments im Einsatz ist, können möglicherweise die Personen Auskunft geben, die Deployments durchführen, da diese Funktion vor dem Aufkommen des DevOps-Konzepts (siehe Abschnitt 3.7) oft in einer zentralen Gruppe gebündelt wurde.

Sollten keine Informationen über Ansprechpartner über die genannten Wege erhältlich sein, hilft als letzter Weg die Analyse der Anwendungen selbst. Hilfreich kann die Datei sein, die zur Steuerung der Build-Automatisierung oder Abhängigkeitsverwaltung dient. Ein Beispiel hierfür ist die Datei pom.xml (Project Object Model), die bei *Apache Maven* eingesetzt wird. In ihr ist der Link auf das Sourcecode Repository enthalten. Außerdem können Ansprechpartner in diesen Dateien vermerkt sein. Durch den Link auf ein Sourcecode Repository lassen sich eventuell über die Historie ebenfalls weitere Ansprechpartner finden.

Aus der Praxis:

Die bei der Allianz eingesetzten *WebSphere Application Server* liefern ohne großen Aufwand eine Liste aller installierten Anwendungen. Dies zeigt die folgende Abbildung 17. Zur Steuerung von Deployments setzt die Allianz eine eigenentwickelte Software namens *Flight Control* ein. Sie kann eine Liste aller Anwendungen exportieren, die auch Ansprechpartner der Entwicklungseinheiten enthält, sowie in Teilen auch fachliche Ansprechpartner für Abnahmetest.

Auswählen	Name	Anwendungsstatus
☐	BaufiBoniEAR-511	
☐	DATAU	
☐	DATAUADM	
☐	ISRAConsole	
☐	Kdg-511	
☐	MISPortalEAR-511	
☐	ODI_Chart	
☐	OOM_war	
☐	RUSHOnline-511	
☐	ZBM-DB_1.1	
☐	aktonlpro3-100	
☐	altaktenanforderung1.0	
☐	ambenutzerverwaltung	
☐	amcontrolling	
☐	amidp	
☐	amisaktonl-100	
☐	amisbaskondamis-400	
☐	amisfirmeninfo-511	
☐	amisfleetwaredvEAR	
☐	amishbd-100	

Abbildung 17: Auszug der Anwendungsübersicht der Klassik-Zelle

Nach dem Finden von Ansprechpartner, Sourcecode und möglicherweise von Dokumentation muss der Änderungsbedarf abgeschätzt werden. Grundlage hierfür ist das Zielbild, das im vorhergehenden Schritt entwickelt wurde. Bei großen Migrationsvorhaben oder wenn es im Unternehmen wenig Know-how zu den betroffenen Anwendungen gibt, bietet es sich an, diese Aufgabe zentral zu erledigen, um eine grobe Indikation zu erhalten. Ein Mittel hierzu kann ein automatisierter Scan über den Sourcecode aller Anwendungen sein. Bei ihm kann gezielt nach Packages oder Funktionen gesucht werden, die problematisch sind und die im Rahmen der Migration behandelt werden müssen. Zwei Beispiele hierfür sind die Suche nach Packages, die Systemfunktionen bereitstellen oder die Suche nach veralteten Bestandteilen, wie http-Clients. Eine Hilfe kann es sein, wenn bei einer Softwaremigration kritische Punkte durch den Hersteller der Technologie benannt sind. *Oracle* beispielsweise veröffentlich solche Hinweise. Auf sie wird im folgenden Abschnitt eingegangen.

Die folgende Abbildung 18 zeigt die Ergebnisse eines Scans über den Sourcecode aller Anwendungen, die von dem Migrationsvorhaben betroffen sind. Gesucht wurde hierbei nach den Paketnamen.

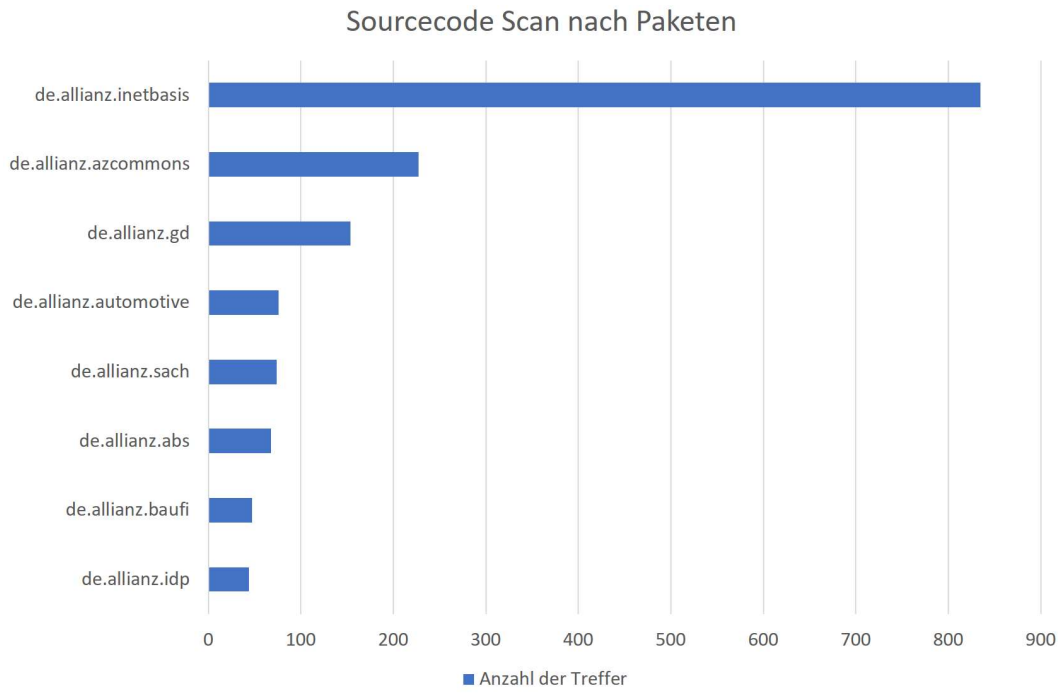

Abbildung 18: Ergebnisse des Sourcecode Scans der Klassik-Zelle

Auffällig bei den Resultaten ist die hohe Trefferzahl bei Paketen mit dem Namen *de.allianz.inetbasis*. Sie enthalten die Framework-Funktionen, die die einzelnen Anwendungen nutzen. Die weiteren Pakete gehören zu einzelnen Anwendungen und werden nicht in verschiedenen Anwendungen wiederverwendet. Der Sourcecode Scan zeigt ebenfalls, dass *I*NET* in fast allen Anwendungen zum Einsatz kommt.

Neben der technischen Analyse werden für die Planung der Migration weitere Informationen benötigt. Sie können mithilfe eines Fragebogens von den Anwendungsteams in Erfahrung gebracht werden. Folgende Fragen können zur Abschätzung des Migrationsaufwands und zur Planung der Umsetzung relevant sein.

Technische Aspekte:

- Welcher Anwendungstyp liegt vor? (Webanwendung oder Webservice – siehe Abschnitt 2.2)
- In welchem Netz ist die Anwendung erreichbar (Internet, Intranet, internes Subnetz)
- Welche Abhängigkeiten besitzt die Anwendung:
    - zum Betriebssystem
    - zu anderen internen Systemen
    - zu externen (Fremd-) Systemen
- Nutzt die Anwendung Kommunikationsprotokolle abseits von TCP/IP und SQL?
- Gibt es weitere Besonderheiten in Bezug auf die Anwendung?
- Wie hoch ist die Komplexität der Anwendung (subjektiv) auf einer Skala von 1 bis 5?

Organisatorische Aspekte:

- Wer sind die Ansprechpartner für die Themen:
    - Anwendungsentwicklung
    - Betrieb der Anwendungen
    - Auftraggeber, Anforderungsgeber oder Product Owner
- Wie sind die Betriebszeiten der Anwendung? Gibt es Service-Level-Agreements oder Verfügbarkeitsvorgaben?
- Wie ist die Perspektive der Anwendung? Ist ein Neubau geplant oder ist die Anwendung in absehbarer Zeit obsolet?
- Gibt es eine Dokumentation für den fachlichen Zweck, die Architektur und die Sicherheitsvorkehrungen der Anwendung?
- Wie hoch ist die Entwicklungskapazität, die im Rahmen des Migrationsvorhabens verfügbar ist?

Der Fragebogen sollte nach Möglichkeit automatisiert ausgewertet werden können, wenn eine größere Zahl an Anwendungen betroffen ist. Die Fragebögen liefern einen Überblick über das Gesamtvorhaben und über mögliche Besonderheiten, sofern sie bekannt sind. Zudem ermöglichen sie anhand der Komplexitätseinschätzung eine Aussage zu den erwarteten Aufwänden. Hierbei hilft auch ein Proof-of-Concept, wie er im folgenden Abschnitt 7.3 beschrieben wird.

Auf Basis der Analyse lassen sich die Anwendungen auf die verschiedenen Migrationspfade verteilen, die in Abschnitt 7.1 beschrieben sind.

- Alle Anwendungen, die *I*NET* verwenden und damit einen *IBM WebSphere Application Server* benötigen, werden einem Refactoring unterzogen. Ihre neue Laufzeitumgebung ist ein *IBM WebSphere Application Server Liberty 16* in einem *Docker Container* auf einer der *OpenShift* Instanzen. Dies trifft auf 82 der 120 Anwendungen zu. Damit kann dieser Weg als Standard bezeichnet werden. Er zeichnet sich durch einen möglichst kleinen Sprung von Status Quo zum Zielsystem aus und trägt der Maßgabe Rechnung, die Kosten gering zu halten. Ein Verzicht auf *I*NET* in den Anwendungen ist aus betriebswirtschaftlichen Gründen kein gangbarer Weg. Die zentralen Funktionen, die *I*NET* den Anwendungen zur Verfügung stellt, können nicht von allen Anwendungen zu akzeptablen Kosten selbst implementiert werden. Zudem würden diese Änderung eine extreme Steigerung der Komplexität des Vorhabens bedeuten und wären damit mit großen Risiken verbunden.

  Aufgrund der breiten Wiederverwendung der *I*NET* Bibliothek bietet es sich an, die Migration von *I*NET* zentral vorzunehmen und die Ergebnisse an die einzelnen Anwendungsteams zu verteilen. Im Folgenden wird diese Version als *I*NET cloud* bezeichnet.

- Von den restlichen Anwendungen haben 13 keine Abhängigkeiten zu *I*NET*. Sie können dadurch einfacher auf den Standard Cloud Stack der Allianz, die *Cloud Foundry* Installationen, migriert werden, also einem Revising unterzogen werden. Dieses Vorgehen lohnt sich insbesondere, wenn absehbar ist, dass die Anwendung nicht kurzfristig abgelöst wird. Ein Refactoring, also die Migration auf eine *OpenShift* Umgebung, ist ebenso denkbar. Da das Revising im vorliegenden Beispiel nur für wenige Anwendungen ein passender Migrationspfad ist, wird es im Verlauf dieser Arbeit nicht weiter betrachtet.
- Es verbleiben 25 weitere Anwendungen. Sie können in absehbarer Zeit außer Betrieb genommen werden und müssen damit im Migrationsvorhaben nicht näher betrachtet werden. Es muss lediglich sichergestellt werden, dass sie abgeschaltet sind, bevor die alten Server abgebaut werden. Der Grund für die relativ hohe Anzahl an ablösbaren Services liegt darin begründet, dass sie viele Funktionen bieten, die auf die Vorgängersysteme des *Allianz Business Systems* zugreifen (siehe Abschnitt 6.1). Diese Vorgängersysteme wurden in den letzten Jahren immer weiter zurückgebaut, weshalb diese Online Anwendungen auf der Klassik-Zelle mit der Zeit obsolet werden.

Ein offener Punkt bei der Festlegung des Zielbildes ist noch, auf welcher *OpenShift* Installation die Anwendungen nach dem Refactoring beheimatet werden. In der Allianz Cloud-Umgebung

sind pro Abnahmestufe zwei *OpenShift* Installationen vorhanden, die sich durch ihre Erreichbarkeit unterscheiden (siehe auch Abschnitt 6.4 auf Seite 47). Die Installation im Outer Layer ist aus dem Internet erreichbar, hat jedoch keinen Zugriff auf Backend-Systeme, wie die Datenbanken. Die *OpenShift* Umgebung im Inner Layer besitzt diese Zugriffsmöglichkeit. Ein direkter Zugriff aus dem Internet wird jedoch durch das API-Gateway unterbunden.

Mit einem Scan über den Quellcode der betroffenen Anwendungen lässt sich erkennen, dass quasi alle Anwendungen Verbindungen zu Datenbanken oder Bestandssystemen nutzen. Aufgrund dieser Tatsache empfiehlt es sich, die Klassik-Anwendungen gesammelt im Inner Layer der Cloud-Infrastruktur zu beheimaten. Damit sie aus dem Internet erreichbar werden, wird im Outer Layer ein Edge Service installiert. Er fungiert als Proxy Service und reichert eingehende Requests mit den notwendigen Sicherheitsinformationen an. Für diese Art von Services existiert bereits eine Beispielimplementierung im Haus Allianz, die auf *Cloud Foundry* lauffähig ist. Abbildung 19 zeigt in gelb die Verortung der Komponenten in der Allianz Cloud-Umgebung mit einem Zugriffspfad aus dem Internet über einen Edge Service.

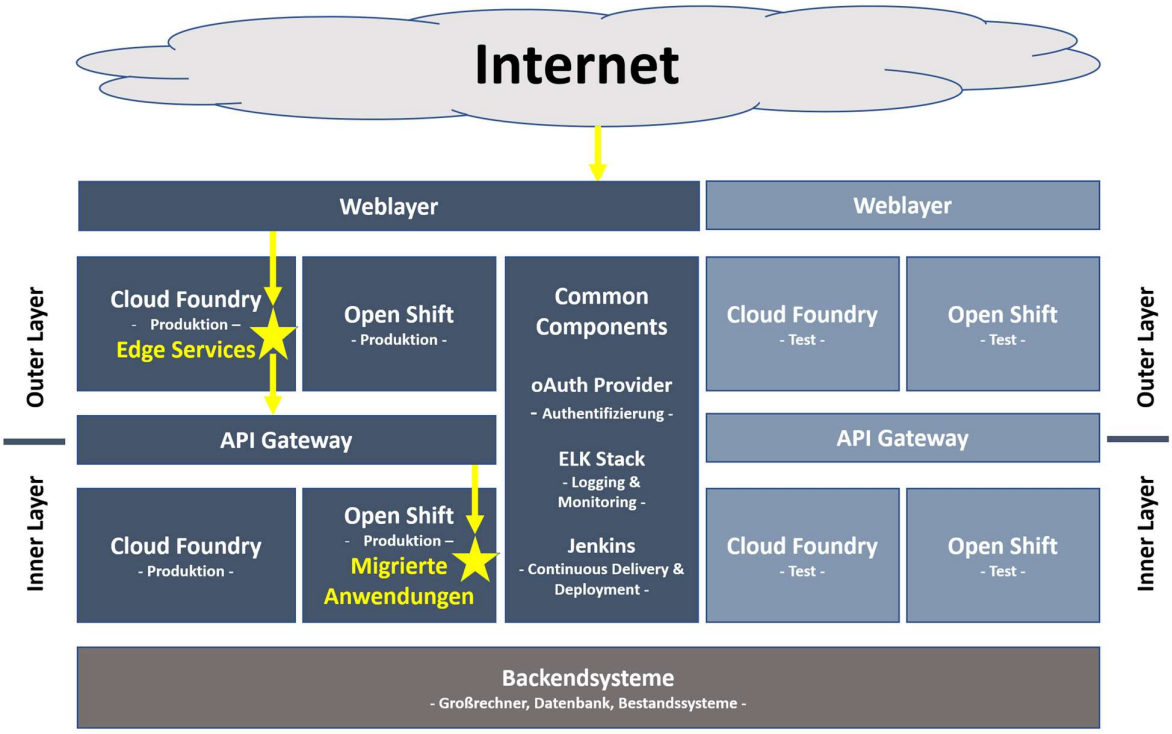

Abbildung 19: Darstellung der migrierten Anwendungen in der Allianz Amazon Cloud mit Zugriffpfad aus dem Internet

Die Zugriffe werden vom zentralen Weblayer zu den Edge Services geleitet. Diese ergänzen die benötigten Sicherheitsinformationen und leiten die Aufrufe über das API-Gateway weiter an die migrierten Anwendungen.

Mit der Auswahl des *IBM WebSphere Application Server Liberty 16* als Zielserver in der Verbindung mit *I*NET cloud* ergibt sich ein neuer Software-Stack für die Klassik-Anwendungen. Abbildung 20 zeigt diesen im Vergleich zum bestehenden Aufbau. Dieser wurde in Abschnitt 6.3 in Abbildung 13 auf Seite 46 beschrieben.

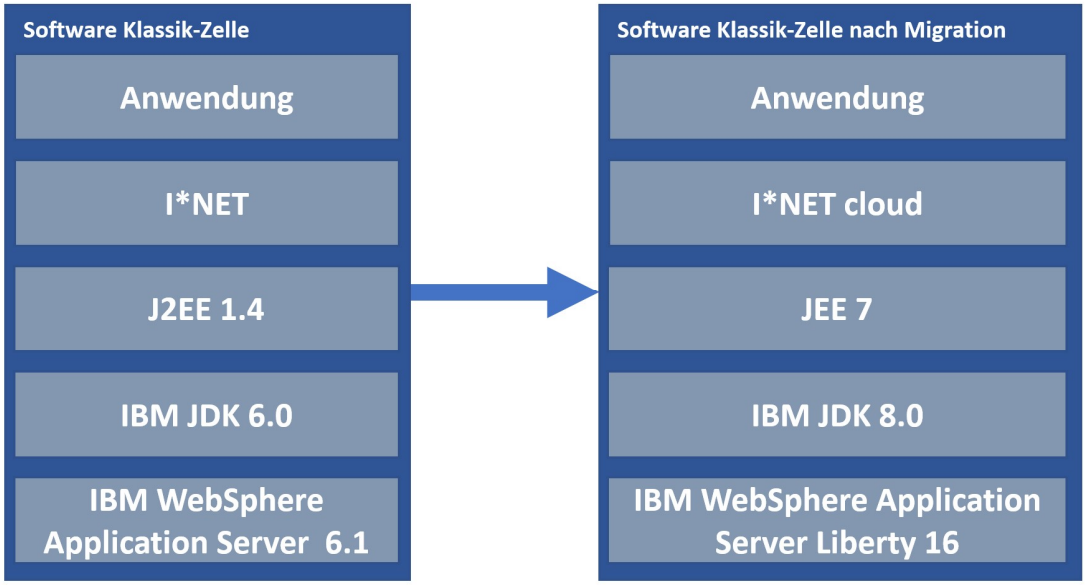

Abbildung 20: Software-Stack der Anwendungen der Klassik-Zelle vor und nach der Migration

Die Abbildung zeigt, dass für die einzelnen Komponenten eine moderne Entsprechung auf der Cloud vorhanden ist. Dies ist ein großer Vorteil, da so tiefgreifende Umbauten in der Architektur vermieden werden. Die Basis ist der Wechsel vom toxischen *IBM WebSphere Application Server 6.1* auf den *IBM WebSphere Application Server Liberty 16*. Dieser stellt ein *Java Development Kit* (JDK) in Version 8 und Unterstützung der *Java Platform Enterprise Edition 7* (JEE) bereit. Das Pendant für *I*NET* kann die migrierte Version *I*NET cloud* sein.

Eine Anmerkung: Es wäre ebenso denkbar, eine andere JDK Version – auch von einem anderen Hersteller – einzusetzen. Die Wahl des *IBM Java Development Kit 8* hat folgende Vorteile: Sie ist die aktuellste Version, die verfügbar ist. Damit werden Migrationsschulden vermieden. Zudem ist es am risikoärmsten, das JDK des Herstellers des Application Servers zu verwenden. Nachteile technischer oder wirtschaftlicher Natur sind durch diese Entscheidung nicht zu erwarten.

7.3 Durchführung der Migration

Die eigentliche Durchführung der Migration wird im Folgenden in Phasen eingeteilt, um verschiedene Schwerpunkte zu setzen. In der Praxis hilft diese Einteilung dem Projektmanagement dabei, überprüfbare Meilensteine zu definieren und so einen Gesamtüberblick bei vielen Anwendungen zu erhalten. Nach jeder Phase sollte die Anwendung in einer funktionsfähigen Form vorliegen. Dadurch wird das Risiko minimiert, dass das Migrationsvorhaben aufgrund zu hoher Komplexität scheitert.

Zur Steuerung der Migration empfiehlt es sich, das Projektmanagement mit einer Software zu unterstützen. In ihr können die folgenden Phasen als Meilensteine abgebildet werden. Wenn mehrere Anwendungen gleichzeitig migriert werden, besteht damit die Möglichkeit, die einzelnen Teams zu steuern und einen Gesamtüberblick zu erhalten.

Wenn mehrere Anwendungen migriert werden sollen, kann es hilfreich sein, zur Risikominimierung einen Proof-of-Concept zu starten. Dabei werden ein oder zwei repräsentative Anwendungen ausgewählt und migriert. Ziel ist es Erfahrungen zu sammeln, wo Schwierigkeiten auftreten und welche Aufwände anfallen können. Bei der Auswahl der Pilotanwendungen ist darauf zu achten, dass die Anwendungen möglichst typisch für alle Anwendungen sind. Es lohnt sich nicht, besonders einfach oder besonders schwer zu migrierende Services auszuwählen, da die Erkenntnisse des Proof-of-Concepts damit nicht mehr Schlüsse auf die Migration aller Anwendungen erlauben.

Unabhängig davon, ob vor der Migration ein Proof-of-Concept durchgeführt wird oder ob darauf verzichtet wird, müssen folgende Phasen durchlaufen werden:

Phase 1: Migration der zentralen Komponenten

Für die Durchführung der Migration bietet es sich an, die Erstellung des *I*NET cloud* durch eine zentrale Einheit vornehmen zu lassen. Das Ergebnis dieser Arbeit kann von allen Anwendungsteams in die eigene Anwendung übernommen werden und vermeidet damit Parallelaufwände. An dieser Stelle bietet es sich zusätzlich an, diese Aufgabe vollständig oder in Teilen an einem externen Dienstleister abzutreten, wenn im eigenen Unternehmen kein Wissen zu diesem Thema verfügbar ist.

Konkret müssen in der bestehenden *I*NET* die Abhängigkeiten zum Application Server untersucht werden und so angepasst werden, dass sie mit dem neuen Application Server funktionieren. Zusätzlich muss die Java Version angehoben werden. Es muss sichergestellt werden, dass *I*NET cloud* mit einem *IBM Java Development Kit 8* gebaut werden kann. Es soll nicht das Ziel sein, alte Funktionen zwingend durch neue Sprachmittel zu ersetzen, es sei denn, dass es unbedingt notwendig ist, weil bestimmte Funktionen von Java 8 nicht mehr unterstützt werden. Die Schnittstellen sollen dabei unverändert beibehalten werden, um keine zusätzlichen Aufwände in den Anwendungsteams zu erzeugen.

Zu Beginn der Arbeiten sollte der Quellcode von *I*NET* in das neue Repository umgezogen werden und es sollte eine entsprechende Build-Pipeline eingerichtet werden. Die Details dazu werden im folgenden Abschnitt beschrieben.

Phase 2: Vorarbeiten

Ein wichtiger Aspekt, der im Vorfeld der Migration erfolgen soll, sind Schulungen für die neuen Technologien. Softwareentwickler werden bei Migrationsvorhaben häufig mit neuen Technologien konfrontiert. Unternehmen sollten Wert auf eine gute Weiterbildung ihrer Mitarbeiter legen. Zudem kann über die Schulungen gesteuert werden, wie neue Technologien eingesetzt werden sollen.

Parallel zur Migration der zentralen Bestandteile können die einzelnen Anwendungsteams beginnen eine Entwicklungsumgebung einzurichten. Nach Wiggins ist ein elementarer Punkt eine zentrale Sourcecode-Verwaltung (Wiggins, Codebase, 2017). Für die Migration der Allianz Deutschland bedeutet das, dass der Quellcode der Anwendungen aus verschiedenen Repositories in der neuen *GitHub Enterprise Installation* zusammengeführt wird. Der Quellcode lag ursprünglich in einer eigenentwickelten Versionsverwaltung. Einzelne Anwendungsteams überführten ihre Artefakte aber im Laufe der Zeit auf *Apache Subversion (SVN)*-Installationen, die im Umfeld *ePortale2* genutzt wurden. Für die Überführung des Quellcodes in das neue Repository kann ein kommerzielles Werkzeug eingesetzt werden. (TMate Software, 2018)

Zur neuen Toolchain gehört auch die konsequente Nutzung von Build-Servern, um Continuous Integration einzuführen. Dies ist ein erster Zwischenschritt zu einem DevOps-Modell (siehe Abschnitt 3.7 und 3.8). Im Rahmen der Vorarbeiten kann das Team diese Toolchain einrichten. In der Praxis müssen beispielsweis Zugänge und Berechtigungen zu den Systemen eingerichtet werden und technische Verbindungen, zum Beispiel Freischaltungen in der Firewall, erfolgen.

Phase 3: Sicherstellen der Lauffähigkeit und der Einbau von *I*NET cloud*

Auf Basis des migrierten *I*NET cloud* und einer funktionierenden Entwicklungsumgebung können die einzelnen Anwendungsteams in die nächste Phase starten. Das Ziel von ihr ist, die Anwendungen mit dem neuen *I*NET cloud* auf der neuen Laufzeitumgebung lauffähig zu machen.

Bevor die Arbeit am Quellcode begonnen wird, sollte sich das ausführende Team mit dem fachlichen Zweck und der technischen Umsetzung der Anwendung auseinandersetzen. Dieses Wissen wird später benötigt, um die Funktionsfähigkeit der Anwendung zu beurteilen und um Fehler zu beseitigen.

Das technische Vorgehen kann in folgende Schritte eingeteilt werden:

1. Die *lokale Buildfähigkeit* stellt sicher, dass die Anwendung lokal gebaut werden kann. Das bedeutet, dass der Quellcode kompiliert[12] werden kann und alle Abhängigkeiten auf Codeebene vorhanden und auflösbar sind.

2. *Lokale Deployfähigkeit* bedeutet, dass die Anwendung (auf der alten Laufzeitumgebung) lokal installiert werden kann und dort ansprechbar ist. Nach dem Deployment werden keine Fehlermeldungen ausgegeben.

3. Die *lokale Lauffähigkeit* ist der dritte Schritt. Er bedeutet, dass die Anwendung auf der alten Laufzeitumgebung betrieben werden kann. Für diesen Schritt muss eine passende Konfiguration vorhanden sein. Außerdem besitzt der Nutzer alle Rechte, die er zur Verwendung benötigt.

Diese drei Schritte sind zu Beginn der Migration von Altanwendungen wichtig, um mögliche Fehler in der Anwendung zu finden, die eventuell seit Jahren verhindern, dass die Anwendung problemlos weiterentwickelt werden kann. Durch die drei genannten Schritte ist sichergestellt, dass der letzte Codestand vollständig ist und funktionieren kann. Eine Überprüfung der fachlichen Anforderungen erfolgt an dieser Stelle nicht.

Als Folgeschritt können die alten *I*NET* Abhängigkeiten durch die migrierte Version ersetzt werden. Dadurch, dass die Schnittstellen des neuen *I*NET cloud* unverändert sind, ist der Aufwand für diesen Schritt minimal.

Zudem muss auch im Quellcode der Anwendung die Java Version erhöht und mögliche Fehler behoben werden. Der Hersteller *Oracle* stellt für jede Java Versionen Release Notes bereit, die *Breaking Changes*[13] auflisten (Oracle Corporation, Java SE 7 and JDK 7 Compatibility, kein Datum und Oracle Corporation, Compatibility Guide for JDK 8, kein Datum). Einige *Breaking Changes* sind für die Migration der Klassik-Zelle nicht relevant. Damit ist es möglich, den Anwendungsteams eine präzise Liste mit weniger als einem Dutzend Einträge zu liefern. Relevante Beispiele sind:

- Das Verhalten der Klassen *TreeSet* und *TreeMap* beim Hinzufügen von ungültigen Elementen hat sich geändert

- Das Verhalten der Klasse *Collection* hat sich in Bezug auf die Sortierung geändert

- Die Formatierung von Monatsnamen wurde verändert

[12] Ein Compiler übersetzt lesbaren Quellcode in einer Programmiersprache in eine ausführbare Version in Maschinensprache.

[13] Breaking Changes beschreiben Änderungen an einer Software-Komponente, die nicht abwärtskompatibel sind. Das bedeutet, dass Nutzer der Software zwingend Anpassungsarbeiten vornehmen müssen, da sich das Verhalten oder die Schnittstellen zwischen der alten und der neuen Version unterscheiden.

- Die Klassen *NumberFormat* und *DecimalFormat* verhalten sich anders beim Aufruf der *format*-Methode

Eine Besonderheit der Klassik-Zellen-Anwendungen zeigt sich beim Thema Konfiguration. Auf der Klassik-Zelle ist eine Globale Konfiguration vorhanden. Dies ist ein zentrales Projekt, das global verfügbare Konfigurationseinträge auf allen Servern zur Verfügung stellt. Problematisch ist, dass die Anwendungen diese Einstellungen lokal überschreiben können, falls dies notwendig ist. Im Rahmen der Migration muss die Globale Konfiguration aufgegeben werden, da sie nicht auf verschiedenen Container gleichzeitig aktualisiert werden kann. Zudem existiert die organisatorische Einheit nicht mehr, die die Pflege der Globalen Konfiguration früher innehatte. Das bedeutet, dass alle Anwendungen ihre Konfiguration komplett selbst bereitstellen müssen. Bei einigen Stichproben zeigt sich, dass nicht oder nur mit großem Aufwand nachvollziehbar ist, welcher Konfigurationseintrag auf welcher Abnahmestufe von der Anwendung tatsächlich verwendet wird.

Nach all diesen Anpassungen muss sichergestellt sein, dass die Anwendung wieder *lokal buildfähig* ist. In der Folge kann der Wechsel der Laufzeitumgebung vorgenommen werden. Im Beispiel der Allianz wäre dies der Wechsel von *WebSphere Application Server 6.1* auf *WebSphere Application Server Liberty 16*. Anschließend sollte die Anwendung wieder *lokal deploy- und lauffähig* sein.

Im Rahmen der Änderungen am Quellcode sollte das Anwendungsteam Unit Tests einbauen, die bestimmte Aspekte der Anwendung automatisch prüfen (siehe Abschnitt 2.5 - Software-Tests). Bestehende Unit Tests sollen dabei erhalten und angepasst werden. Auch neue Funktionen sollten mit Unit Tests überwacht werden. Es wird jedoch in der Regel nicht möglich sein, Unit Tests für die gesamte Anwendung nachträglich einzuführen. Hierfür ist ein detailliertes Wissen über die Technik und die Fachlichkeit nötig, das bei Legacy-Anwendungen zumeist fehlt.

Phase 4: Cloudifizierung

Die vierte Phase ist die *Cloudifizierung* der Anwendung. Erst in diesem Schritt werden die Spezifika der Cloud-Umgebung betrachtet. Die lauffähige Anwendung wird so umgebaut, dass sie in Containern benutzt und auf der Zielumgebung installiert werden kann.

Im Falle der Allianz bedeutet das, dass die migrierte Anwendung erstmals in einem Cloud-Container installiert wird. Um dies zu ermöglichen, müssen die Anforderungen aus Abschnitt 7.1 umgesetzt werden. Das bedeutet beispielsweise die Konfiguration und das Logging der Anwendungen umzustellen. Der *WebSphere Application Server Liberty* wird als vorgefertigtes Image allen Teams zentral bereitgestellt. Er läuft innerhalb eines *Docker Container*s und wird nur von einer Anwendung genutzt.

Zur Erzeugung und zum Betrieb eines *Docker Container*s sind einige Konfigurationseinstellungen notwendig:

- Die Basis ist das Dockerfile. Es enthält die Konfiguration eines *Docker Container*s. Dockerfiles können voneinander abgeleitet werden. Das bedeutet, dass ein Basis-*Docker*file unternehmensweit zur Verfügung gestellt werden, um beispielsweise alle benötigten Informationen zu Softwarelizenzen zu verteilen. Es enthält desweitern die Information, wo die Releases der Anwendung verfügbar sind (im Falle der *Allianz Deutschland AG* ist dies ein Verweis auf das Artifactory) sowie ein Verweis auf die Konfiguration der Anwendung.

- Die Deployment-Konfiguration beschreibt, wie ein *OpenShift* Pod aufgebaut wird. Sie legt fest, wie viele Instanzen gestartet werden, welche Ports verwendet werden und enthält Verweise auf die Anwendungskonfiguration. Zudem steuert die Deployment-Konfiguration das Verhalten eines Pods, beispielsweise ob er im Fehlerfall neugestartet werden soll. (Red Hat Incorporated, OpenShift - Developer Guide - Deployments, kein Datum)

- Um einen Service erreichbar zu machen, muss zusätzlich die Definition von einer Route in *OpenShift* erfolgen. Diese werden ebenfalls in einer Textdatei festgelegt. (Red Hat Incorporated, OpenShift - Developer Guide - Routes, kein Datum)

Eine weitere wichtige Festlegung, die die Entwickler treffen müssen, ist der maximale Speicherplatz, der ihrer Anwendung zur Verfügung steht. Wird dieser Wert vom Container überschritten, erfolgt ein automatischer Neustart des Containers. Der Wert muss deshalb groß genug gewählt werden, damit der Container im Fehlerfall abgelöst wird, jedoch im Normalfall dauerhaft arbeiten kann. In die Berechnung des Speicherbedarfs gehen die Anforderung des *WebSphere Application Server Liberty* ein, ebenso wie der Speicherbedarf des *Docker Container*s selbst.

Neben den Änderungen am Quellcode der Anwendung und der Konfiguration müssen in dieser Phase einige organisatorische Punkte umgesetzt werden. Mit dem Umzug auf die Cloud-Umgebung ändert sich der netzwerktechnische Standort der Anwendungen. Das bedeutet einerseits, dass möglicherweise neue Freischaltungen an Firewalls eingerichtet werden müssen, um vom neuen Standort auf Backend-Systeme zugreifen zu können. Andererseits muss es ermöglicht werden, dass die aufrufenden Anwendungen auf die neue Instanz zugreifen können. Hierfür können ebenfalls Firewall-Freischaltungen nötig sein. Zudem muss die Umstellung der Nutzer geplant und koordiniert werden. Wenn die Kunden nacheinander und unabhängig voneinander auf die Anwendung auf der Cloud umsteigen können, empfiehlt es sich hierfür einen Terminplan zu erstellen, um eine entsprechende Ressourcenverfügbarkeit auf beiden Seiten sicherzustellen. Wenn ein Umstieg von allen Konsumenten gleichzeitig erfolgen muss, ist ein gangbarer Weg, den DNS-Eintrag der Anwendung zu ändern. Das bedeutet, dass die URL (Uniform Resource Locator) der Anwendung identisch bleibt, aber im Hintergrund mit den IP-Adressen der neuen Instanz verbunden wird. Festzuhalten ist, dass dieser Weg durch die Umstellung aller Nutzer gleichzeitig ein hohes Risiko bedeutet.

Die Schritte zu Umstellung der Nutzer werden zwar erst in der Phase *Produktionseinführung* (beschrieben in Abschnitt 7.5) umgesetzt, sollten jedoch frühzeitig geplant und kommuniziert werden.

7.4 Test

Die Basis um erfolgreiche Tests durchführen zu können, ist eine professionelle Testplanung. Sie legt Inhalte und Termine fest, überwacht die Durchführung des Tests und kontrolliert die Behebung der gefundenen Fehler. Kernbestandteil ist die Festlegung von Testfällen und deren Akzeptanzkriterien. Diese Arbeit wird in der Praxis oft von der Auftraggeberseite oder einer eigenständigen Einheit umgesetzt, nicht von den IT-Entwicklungseinheiten. Das Aufstellen von Testfällen ist zeitintensiv und muss deshalb vor der eigentlichen Phase der Testdurchführung begonnen werden. Insbesondere bei Legacy-Anwendungen ist oftmals kein Testset als Basis vorhanden und die genauen Akzeptanzkriterien sind nicht mehr oder nur in Teilen bekannt. Deshalb sollte hier mit erhöhten Aufwänden gerechnet werden. Wenn einzelne Anwendungen in einem funktionsfähigen Zustand sind, kann begonnen werden die Tests durchzuführen, die in Abschnitt 5.4 beschrieben wurden.

Der erste übergreifende Test, der durchgeführt wird, ist üblicherweise ein Integrationstest mit den beteiligten Backendsystemen. Er dient zur Sicherstellung der technischen Funktionsfähigkeit des Gesamtsystems.

Der umfangreichste Test ist ein fachlicher Abnahmetest. Er wird optimalerweise vom Auftraggeber oder Anforderungsgeber durchgeführt. Beurteilt werden die fachlichen Funktionen der Anwendungen, das Verhalten im Fehlerfall, aber auch nicht-funktionale Anforderungen, wie die gefühlte Performanz des Systems.

Anschließend sollte ein Lasttest durchgeführt werden. Er dient zu Sicherstellung der Leistungsfähigkeit der neuen Umgebung und der migrierten Anwendungen. Optimalerweise wird vor der Migration ebenfalls ein Lasttest durchgeführt, damit Vergleichs- beziehungsweise Erwartungswerte bekannt sind.

Auch ein Penetrationstest der Anwendung sollte durchgeführt werden, damit mögliche Sicherheitslücken aufgedeckt werden. Oftmals wird hierfür in der Praxis ein externer Dienstleister beauftragt, da er unabhängig ist und die Mitarbeiter dieser Firmen über spezielles Know-how verfügen.

7.5 Produktionseinführung und Hypercare

Die Produktionseinführung der migrierten Anwendungen muss ebenfalls sorgfältig vorbereitet werden. Wichtig ist eine frühzeitige Kommunikation an die betroffenen Nutzer und zugehörigen Backends.

Für die Allianz empfiehlt es sich, die Inbetriebnahme in verschiedenen Wellen durchzuführen. Durch die große Anzahl an Anwendungen wäre eine Big Bang Einführung (siehe Abschnitt 5.4 - Ablauf einer Softwaremigration) zu komplex und mit hohem Risiko behaftet. Die Einführung in verschiedenen Wellen bietet den Vorteil, dass die Menge der Anwendungen überschaubar ist. Zudem kann bei einer geringen Zahl von Einführungsterminen Support von übergreifenden Einheiten, wie dem Server-/Plattformbetrieb, bereitgestellt werden. Bei der Einführung aller Anwendungen an unterschiedlichen Terminen wäre das nicht möglich.

Im Anschluss an die Produktivsetzung der migrierten Version, sollte das Entwicklungsteam für etwaige Fehlersituationen bereitstehen. Um diese Situationen frühzeitig zu erkennen, hilft einerseits das vorhandene Monitoring (siehe Abschnitt 7.3 - Entwicklung des Zielsystems). Andererseits können aber auch Logfiles proaktiv überwacht werden. Bei sehr kritischen, zentralen Anwendungen kann es auch ratsam sein, für einige Tage eine 24 Stunden Bereitschaft einzurichten.

Für die Produktivschaltung werden in der Praxis oft Termine am Wochenende oder in den frühen Morgenstunden genutzt. Der Hintergrund ist die Wahl eines Termins außerhalb der Hauptnutzungszeit der Anwendung, damit auftretende Problemen keine oder möglichst wenig Nutzer betreffen. Bei der Terminwahl ist jedoch die Verfügbarkeit von Entwicklern und Betriebsexperten abzuwägen.

Nachdem die neue Instanz der Anwendung für einige Tage ihre Zuverlässigkeit im Betrieb demonstriert hat, kann der Rückbau der alten Infrastruktur beginnen. Vorab sollte jedoch geprüft werden, ob keine Zugriffe mehr auf die alten Instanzen der migrierten Anwendungen erfolgen. Dieses Risiko besteht insbesondere bei Legacy-Anwendungen, da möglicherweise nicht alle Nutzer bekannt sind oder informiert wurden.

In der Praxis wird der Rückbau üblicherweise zeitversetzt über die Abnahmestufen durchgeführt. Das bedeutet, die Entwicklungs- und Testinstanzen werden zuerst außer Betrieb genommen und abgebaut, bevor die Produktionsinstanz heruntergefahren wird. Auch dieses Vorgehen dient der Risikominimierung.

Der Rückbau beschränkt sich jedoch nicht nur auf die Dekommissionierung der Infrastruktur. Durch die Inbetriebnahme der neuen Instanz der Anwendung können beispielsweise alte Firewall-Freischaltungen obsolet werden. Diese sollten aus Sicherheitsgründen zügig deaktiviert werden, damit sie nicht für Angriffe genutzt werden können. Zudem können auch die alten Build- und Release-Verwaltungssysteme bereinigt beziehungsweise aufgegeben werden, wenn dies aus regulatorischer Sicht möglich ist (für manche Anwendungen gelten in Deutschland je nach Zweck Aufbewahrungsanforderungen von bis zu 10 Jahren aus Revisionsgründen).

Nicht zu vernachlässigen ist auch die Aktualisierung der vorhandenen Dokumentation. Dies bezieht sich nicht nur auf die Anwendungsdokumentation, sondern auch auf übergreifende Systeme. *Die Allianz Deutschland AG* betriebt beispielsweise eine *Configuration Management Database*, eine Datenbank, die die Systemlandschaft mit allen Abhängigkeiten und Querverbindungen abbildet. Sie dient als Basis für die Beurteilung von Ausfällen und die Steuerung von Änderungen an der Infrastruktur. Zudem ist sie die Basis für einen Wiederaufbau der Anwendungslandschaft

nach einem Katastrophenfall. Alle Änderungen in der Anwendungslandschaft sind deshalb zwingend in dieser Datenbank zu pflegen.

7.6 Übergreifende Themen

Die Handlungsfelder und Aufgaben während einer Migration können zu Beginn des Vorhabens nicht vollständig abgeschätzt werden. Dafür sollte im Projekt Wert auf ein funktionierendes Risikomanagement gelegt werden. Risiken müssen bei der Erkennung direkt transparent gemacht und kommuniziert werden. Zudem sollte dem Management bewusst sein, dass die initiale Kostenschätzung nur eine grobe Abschätzung sein kann und kein Anspruch auf Vollständigkeit besteht. Ebenso können initial erstellte Zeitpläne durch unvorhergesehene Migrationsaufwände schnell überholt sein. Auch hier ist von Seiten des Managements Flexibilität gefordert. Ein iteratives Vorgehen, die Abarbeitung einer Migrationsphase nach der anderen, kann hier helfen, vor allem, wenn nur die Realisierungstermine für das Ende der kommenden Phase im Voraus geplant werden.

Während einer Migration, die mehrere Teams parallel durchführen, sollte eine zentrale Plattform geschaffen werden, über die die Teams Probleme und Lösungen austauschen und diskutieren können. Geeignet sind hierfür beispielsweise ein internes Wiki-System oder andere kollaborative Software wie ein Messenger. Auch regelmäßige Treffen der Entwickler, in denen Probleme diskutiert werden können, können im Rahmen der Migration sehr hilfreich sein.

Ein Proof-of-Concept eignet sich insbesondere bei Migrationsvorhaben mit einer größeren Zahl an Anwendungen. Wenn vorab ein Proof-of-Concept durchgeführt wird, sollten die Erkenntnisse, Herausforderungen und Lösungen für alle Entwickler zugänglich sein. Aus dem Proof-of-Concept kann beispielsweise ein Entwicklungsleitfaden mit Best Practices (oder Good Practices) entstehen.

8 FAZIT UND AUSBLICK

Das Ziel dieser Arbeit ist es, zu untersuchen, ob Legacy-Anwendungen mit einer Laufzeit von mehreren Jahren auf einer modernen Cloud-Umgebung betrieben werden können. Am Beispiel der *Allianz Deutschland AG* zeigt sich: ja, der technologische Sprung auf Cloud-Technologie kann gelingen.

Bei der Planung der Migration wird deutlich, dass anerkannte Methoden zur Softwaremigration auch im vorliegenden Szenario funktionieren können. Ein besonderes Augenmerk muss zu Beginn jedoch auf die Analyse der Anwendungen gelegt werden. Diese Phase ist bei Legacy-Anwendungen deutlich aufwendiger, als bei Anwendungen, die sich noch in der Weiterentwicklung befinden. Ansprechpartner sind schwierig zu finden oder nicht mehr im Unternehmen beschäftigt. Zudem ist das Wissen über Details und Besonderheiten in den Anwendungen oft verblasst. Dies führt auch zu einem erhöhten Risiko bei der Durchführung der Migration, da manche notwendige Arbeitsschritte möglicherweise erst in der Umsetzung erkannt werden. Um diesem Punkt entgegenzuwirken, hilft nur eine gründliche und tiefgehende Analyse der betroffenen Anwendung, sowie ein gutes Risikomanagement. Zudem muss dem Management bewusstgemacht werden, dass unvorhergesehenen Kosten und Zeitverzögerungen auftreten können.

Die eigentliche Migration erfolgt am sinnvollsten, indem sie in viele kleine, überschaubare Phasen aufgeteilt wird. Der *Chicken Little* Ansatz nach Brodie und Stonebraker ist auch nach 15 Jahren nach der Veröffentlichung sehr hilfreich. Der Trennung zwischen Modernisierung und Cloudifizierung hilft, den Fokus zu wahren und das Risiko des Scheiterns zu senken. Durch die Anlehnung an das agile Manifest liegt zudem nach jeder Phase eine nutzbare Anwendung vor. So kann die korrekte Funktionsweise immer wieder überprüft werden, und es können genauere Aussagen über die Qualität der Anwendung getroffen werden.

Wenn zentrale Bibliotheken vorhanden sind, empfiehlt es sich, diese von einer zentralen Einheit migrieren zu lassen. Da diese Arbeit abgeschlossen sein muss, bevor einzelne Anwendungsteams die Arbeit aufnehmen, bietet es sich an, dass das Migrationsteam der zentralen Komponenten aufgrund der gesammelten Erfahrung im weiteren Verlauf als Ansprechpartner bei Problemen zur Verfügung steht.

Nach der Migration zeigen sich die Vorteile einer modernen Cloud-Umgebung: Die verwendete Technologie ist State-of-the-art, und es ist ausreichend Wissen über sie im Unternehmen vorhanden. Durch die Skalierbarkeit der Cloud-Umgebung entstehen keine Probleme mehr in Hochlastsituationen. Unterbrechungsfreie Deployments sind der Standard und die Anwendungen sind hoch verfügbar. Der Service Level der Anwendungen wird also gegenüber der Ausgangslage verbessert. Außerdem wurde der Zweck jeder Anwendung geprüft und einige Anwendungen abgelöst.

Durch die Ablösung der toxischen Technologie wird das Sicherheitsniveau signifikant erhöht. Alle verwendeten Komponenten werden in aktuellen Versionen genutzt und erhalten weiterhin Updates, wenn Sicherheitslücken gefunden werden. Dadurch wird die Gefahr von Hackerangriffen gesenkt.

Bei der Durchführung der Migration zeigt sich jedoch auch, dass sich die Anforderungen an einen Softwareentwickler im Sinne des DevOps Gedanken erweitern. Die Zuständigkeit, eine passende Ablaufumgebung für eine Anwendung zur Verfügung zu stellen, liegt nun in der Anwendungsentwicklung. Die Entwickler übernehmen die Konfiguration von Containern und die Einrichtung einer Continuous Deployment-Umgebung. Dafür müssen die Mitarbeiter unbedingt geschult werden. Das bedeutet aber mitnichten, dass der Serverbetrieb seine Aufgaben verliert. Ihm kommt weiterhin die Kernaufgabe zu, die Ablaufumgebungen an sich, beispielsweise die *Cloud Foundry* oder *OpenShift* Instanzen, zu betreiben.

Die Cloud-Technologie an sich wird sich in den nächsten Jahren weiterverbreiten. Die technologischen Vorteile, wie die automatische Skalierung und die damit einhergehenden betriebswirtschaftlichen Vorteilen, wie die Kostensenkungen durch eine bessere Auslastung der Ressourcen machen Cloud-Umgebungen für sehr viele Firmen interessant. Insbesondere Platform-as-a-service Angebote, die eigenen Rechenzentren obsolet machen, erhöhen die Attraktivität von Cloud-Umgebungen auch für kleine und mittelständige Unternehmen. Sie versprechen den Unternehmen eine höhere Geschwindigkeit in der Entwicklung und eine nicht gekannte Flexibilität. Durch die weitere Verbreitung ergibt sich zwangsläufig, dass viele Unternehmen ähnliche Migrationsvorhaben wie die *Allianz Deutschland AG* durchführen werden. Diese Migrationen können gelingen, wenn der Änderungswille und die Finanzierung des Vorhabens vorhanden sind. Durch die entstehenden Kosten kann es in vielen Fällen jedoch sinnvoller sein, eine Anwendung abzulösen oder neuzubauen – insbesondere, wenn sie noch eine erwartete Lebensdauer von mehreren Jahren hat.

ABKÜRZUNGSVERZEICHNIS

ABS	Allianz Business System
AG	Aktiengesellschaft
API	Application Programming Interface
BI	Business Intelligence
CIO	Chief Information Officer
COBOL	Common Business Oriented Language
CSS	Cascading Style Sheets
ELK	Elasticsearch, Logstash & Kibana
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IBM	International Business Machines
IDP	Identity Provider
IP	Internet Protocol
IT	Informationstechnologie
IaaS	Infrastructure-as-a-Service
JEE / J2EE	Java Platform, Enterprise Edition
JDK	Java Development Kit
JRE	Java Runtime Environment
JSF	JavaServer Faces
JSON	JavaScript Object Notation
JWT	JSON Web Token
NIST	National Institute of Standards and Technology
OAuth	Open Authorization
PaaS	Platform-as-a-Service
POM	Project Object Model
POODLE	Padding Oracle On Downgraded Legacy Encryption
REST	Representational State Transfer
SaaS	Service-as-a-Service
SE	Societas Europaea

SOA	Service-Oriented Architecture (Serviceorientierte Architektur)
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
SSL	Secure Sockets Layer
SVN	Apache Subversion
TCP	Transmission Control Protocol
TLS	Transport Layer Security
URL	Uniform Resource Locator
WSDL	Web Services Description Language
W3C	World Wide Web Consortium
XML	Extensible Mark-up Language

ABBILDUNGSVERZEICHNIS

TABELLENVERZEICHNIS

LITERATURVERZEICHNIS

Allianz Deutschland AG. (2016). *Geschäftsbericht 2016.* München: Allianz Deutschland AG.

Allianz Deutschland AG. (Januar 2018). ePortale 2 - Server- und Netzwerkübersicht. München.

Allianz Deutschland AG, Referat IT-Architektur. (2018). *Architekturrichtlinien für Online Anwendungen.* München.

Allianz SE. (31. Dezember 2017). *Wer wir sind: Auf einen Blick*. Abgerufen am 31. Januar 2018 von Allianz SE: https://www.allianz.com/de/ueber_uns/wer_wir_sind/auf-einen-blick/

Allianz SE. (29. Januar 2018). *Allianz baut offene Plattform auf - Mit Open Source die Intelligenz der gesamten Branche nutzen*. (A. SE, Herausgeber) Abgerufen am 2. Februar 2018 von Allianz SE: https://www.allianz.com/de/presse/news/geschaeftsfelder/versicherung/180129_allianz-baut-offene-plattform-auf/

Amazon Web Services Incorporation. (kein Datum). *Amazon EC2*. Abgerufen am 30. Januar 2018 von https://aws.amazon.com/de/ec2/

Amazon Web Services Incorporation. (kein Datum). *Was ist Amazon VPC?* Abgerufen am 30. Januar 2018 von Amazon Virtual Private Cloud: https://docs.aws.amazon.com/de_de/AmazonVPC/latest/UserGuide/VPC_Introduction.html

Andreessen, M. (20. August 2011). *Why Software Is Eating The World*. Abgerufen am 20. Juli 2017 von The Wall Street Journal: http://www.wsj.com/articles/SB

Augsten, S. (4. Januar 2017). *Was ist DevOps?* Abgerufen am 3. August 2017 von Dev Insider: https://www.dev-insider.de/was-ist-devops-a-570286/

BBC. (1. Januar 2000). *Y2K bug fails to bite*. Abgerufen am 17. Oktober 2017 von BBC News: http://news.bbc.co.uk/2/hi/science/nature/585013.stm

Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., . . . Tho. (2001). *Principles behind the Agile Manifesto*. Abgerufen am 3. Februar 2018 von Manifesto for Agile Software Development: http://agilemanifesto.org/principles.html

Bernstein, D. S. (2015). *Beyond Legacy Code.* Raleigh: The Pragmatic Programmers.

Bisbal, J., Grimson, J., Lawless, D., O'Sullivan, D., Richardson, R., Wade, V., & Wu, B. (1997). *A Survey Of Research into Legacy System Migration.* Dublin: Computer Science Department, Trinity College.

Bock, A. (15. März 2015). *Hat der Großrechner bei Versicherungen bald ausgedient?* Abgerufen am 2. März 2018 von BearingPoint: https://www.bearingpoint.com/de-de/ueber-uns/pressemitteilungen-

und-medienberichte/pressemitteilungen/hat-der-grossrechner-bei-versicherungen-bald-ausgedient/

Brodie, M., & Stonebraker, M. (1993). *DARWIN: On the Incremental Migration of.* Berkeley: o. A.

Bundesamt für Sicherheit in der Informationstechnik. (27. Juni 2017). *Erneut weltweite Cyber-Sicherheitsvorfälle durch Ransomware*. Abgerufen am 2. Oktober 2017 von Bundesamt für Sicherheit in der Informationstechnik: https://www.bsi.bund.de/DE/Presse/Pressemitteilungen/Presse2017/PM_petya_global_27062017.html

Cloud Foundry Foundation. (16. November 2017). *Understanding Cloud Foundry Security*. Abgerufen am 10. März 2018 von Cloud Foundry Documentation: https://docs.cloudfoundry.org/concepts/security.html

Cunningham, L. (21. November 2008). *Toolbox.com - Cloud Computing Defined*. Abgerufen am 20. Juli 2017 von IT Toolbox: http://it.toolbox.com/blogs/oracle-guide/cloud-computing-defined-28433

Eder, N. (Februar 2017). *Wie gehe ich mit Legacy Code um?* Abgerufen am 21. Juli 2017 von Norbert Eder: https://www.norberteder.com/wie-gehe-ich-mit-legacy-code-um/

Erl, T. (2008). *SOA - Principles of Service Design.* Upper Saddle River: Prentice Hall.

Feathers, M. (2011). *Effektives Arbeiten mit Legacy Code.* Frechen: mitp Verlag.

Gierow, H. (16. August 2017). *Not-Petya-Angriff kostet Maersk 200 Millionen US-Dollar*. Abgerufen am 5. September 2017 von Golem: https://www.golem.de/news/ransomware-not-petya-angriff-kostet-maersk-200-millionen-us-dollar-1708-129525.html

Gimnich, R., & Winter, A. (2005). *Workflows der Software-Migration*. Abgerufen am 19. September 2017 von Universität Koblenz Landau: https://www.uni-koblenz.de/~ist/documents/Gimnich2005WDS.pdf

GitHub Incorporated. (kein Datum). *Bring GitHub to work*. Abgerufen am 13. Februar 2018 von GitHub Enterprsie: https://enterprise.github.com/features

Grance, T., & Mell, P. (2011). *The NIST Definition of Cloud - Special Publication 800-145.* Gaithersburg: National Institute of Standards and Technology.

Hagen Management GmbH. (kein Datum). *PM-Vorlagen*. Abgerufen am 30. Januar 2018 von Projektmanagement Handbuch: http://www.pm-handbuch.com/pm-vorlagen/

Haselmann, T., Hoeren, T., & Vossen, G. (2012). *Cloud Computing für Unternehmen.* Heidelberg: dpunkt Verlag.

Heilmann, H., Sneed, H., & Wolf, E. (2010). *Softwaremigrationen in der Praxis.* Heidelberg: dpunkt Verlag.

Hoffmann, K. (2016). *Beyond the Twelve-Factor App.* Sebastopol: O'Reilly Media.

Hofmann, B. (13. August 2012). *So funktioniert OAuth2 - Der Standard für Clientautorisierung im Überblick*. Abgerufen am 30. Januar 2018 von Entwickler.de: https://entwickler.de/online/agile/so-funktioniert-oauth2-134316.html

Horn, T. (2014). *JavaServer Faces (JSF)*. Abgerufen am 13. Februar 2018 von http://www.torsten-horn.de/techdocs/jsf.htm

International Business Machines Corporation. (10. September 2012). *End of Support for WebSphere Application Server V6.1 was September 30, 2013 and End of Life on September 30, 2016.* (I. B. Corporation, Herausgeber) Abgerufen am 19. September 2017 von IBM Support: http://www-01.ibm.com/support/docview.wss?uid=swg21570083

Izrailevsky, Y. (11. Februar 2016). *Cloud-Migration bei Netflix abgeschlossen*. Abgerufen am 26. November 2017 von Netflix Media: https://media.netflix.com/de/company-blog/completing-the-netflix-cloud-migration

Jenkins User Documentation. (kein Datum). Abgerufen am 13. Februar 2018 von Jenkins User Documentation: https://jenkins.io/doc/

JFrog Limitied. (2018). *JFrog Artifactory Features*. Abgerufen am 13. Februar 2018 von https://jfrog.com/artifactory/features/

JWT.io. (kein Datum). *Introduction to JSON Web Tokens*. Abgerufen am 30. Januar 2018 von JWT.io: https://jwt.io/introduction/

Kappel, G., Pröll, B., Reich, S., & Retschitzegger, W. (2003). *Web Engineering - Systematische Entwicklung von Webanwendungen.* Heidelberg: dpunkt.verlag.

Lewis, G., Plakosh, D., & Seacord, R. (2003). *Modernizing Legacy Systems.* Boston: Addison-Wesley.

Mahmoud, Q. (Dezember 2003). *The All New J2EE 1.4 Platform*. Abgerufen am 10. Februar 2018 von Oracle Technology Network: http://www.oracle.com/technetwork/articles/java/new1-4-142627.html

Maier, F., & Rubens, R. (30. September 2015). *Wofür Sie Container brauchen*. Abgerufen am 20. Juli 2017 von Computerwoche Tec Channel: https://www.tecchannel.de/a/container-vs-virtualisierung,3201845,2

Meck, G. (4. Mai 2016). *Hat die Deutsche Bank die IT vernachlässigt?* (F. Allgemeine, Herausgeber) Abgerufen am 26. Juli 2017 von Frankfurter Allgemeine Zeitung: http://www.faz.net/aktuell/wirtschaft/unternehmen/hat-die-deutsche-bank-die-it-vernachlaessigt-kim-hammonds-klagt-es-wurde-zu-viel-ausgelagert-14220345.html

Moore, G. E. (19. April 1965). Electronics . *Cramming more components onto integrated circuits*, S. 114ff.

MuleSoft Corporation. (kein Datum). *Tomcat Websphere Comparison - Choosing The Right Java Server Solution*. Abgerufen am 15. Februar 2018 von MuleSoft: https://www.mulesoft.com/de/tcat/tomcat-websphere

Ndegwa, A. (31. Mai 2016). *What is a Web Application?* Abgerufen am 10. Oktober 2017 von MaxCDN One: https://www.maxcdn.com/one/visual-glossary/web-application/

Netflix Incorporated. (3. Juli 2017). *Hystrix Wiki - Defend Your App*. Abgerufen am 30. Januar 2018 von GitHub: https://github.com/Netflix/Hystrix/wiki

Oracle Corporation. (2012). *Differences between Java EE and Java SE*. Abgerufen am 13. Febraur 2018 von Oracle Technology Network: https://docs.oracle.com/javaee/6/firstcup/doc/gkhoy.html

Oracle Corporation. (kein Datum). *Compatibility Guide for JDK 8*. Abgerufen am 2. März 2018 von http://www.oracle.com/technetwork/java/javase/8-compatibility-guide-2156366.html

Oracle Corporation. (kein Datum). *Java SE 7 and JDK 7 Compatibility*. Abgerufen am 2. März 2018 von http://www.oracle.com/technetwork/java/javase/compatibility-417013.html

Patel, J. (7. Juni 2016). *Software is still eating the world*. Abgerufen am 25. Mai 2017 von Techcrunch: https://techcrunch.com/2016/06/07/software-is-eating-the-world-5-years-later/

Peeples, K. (2. März 2015). *One Way and Two Way SSL and TLS*. Abgerufen am 1. Februar 2018 von The Open Universe: http://www.ossmentor.com/2015/03/one-way-and-two-way-ssl-and-tls.html

Peschlow, P. (Januar 2016). *Die DevOps Bewegung*. Abgerufen am 26. Juli 2017 von Codecentric: https://www.codecentric.de/publikation/die-devops-bewegung/

Pettey, C., & van der Meulen, R. (16. Mai 2011). *Gartner Identifies Five Ways to Migrate Applications to the Cloud*. Abgerufen am 10. Oktober 2017 von Gartner: http://www.gartner.com/newsroom/id/1684114

Plate, J. (6. Mai 2017). *Grundlagen Netzwerkprogrammierung*. Abgerufen am 5. November 2017 von Netzmafia: http://www.netzmafia.de/skripten/inetprog/inetprog1.html

Pletter, R., & Storn, A. (2. April 2016). *Deutsche Bank: "Das geht mir nahe"*. Abgerufen am 26. Juli 2017 von Zeit online: http://www.zeit.de/2016/13/christian-sewing-deutsche-bank-privatkundenvorstand-stellenabbau

Red Hat Incorporated. (kein Datum). *Developer Guide - Deployments*. Abgerufen am 10. März 2018 von OpenShift: https://docs.openshift.com/enterprise/3.1/dev_guide/deployments.html#creating-a-deployment-configuration

Red Hat Incorporated. (kein Datum). *Developer Guide - Routes*. Abgerufen am 10. März 2018 von OpenShift: https://docs.openshift.com/enterprise/3.0/dev_guide/routes.html

Rohmann, C. (Februar 2016). *Elasticsearch, Logstash & Kibana*. Abgerufen am 13. Februar 2018 von Linux - Magazin: http://www.linux-magazin.de/ausgaben/2016/02/elk-stack/

Rosenberger, M. (2013). *Software-Migrationen.* Hamburg: Diplomica Verlag.

Schilli, M. (Oktober 2006). *Automatische Regressionstests mit Selenium*. Abgerufen am 5. November 2017 von Linux-Magazin: http://www.linux-magazin.de/Ausgaben/2006/10/Browser-ferngesteuert

Schmidt, J. (15. Oktober 2014). *Poodle: So funktioniert der Angriff auf die Verschlüsselung*. Abgerufen am 26. November 2017 von Heise Security: https://www.heise.de/security/artikel/Poodle-So-funktioniert-der-Angriff-auf-die-Verschluesselung-2425250.html

Schmidt, J. (15. Oktober 2014). *So wehren Sie Poodle-Angriffe ab*. Abgerufen am 26. November 2017 von Heise Security: https://www.heise.de/security/meldung/So-wehren-Sie-Poodle-Angriffe-ab-2424327.html

Siemens, D. H. (2009). *Ablösung I*Net durch AzCommons.* München: Allianz Deutschland AG.

Soparkar, S. (11. Februar 2017). *Continuous Integration/Delivery(CI/CD) with Heroku*. Abgerufen am 10. Oktober 2017 von Linked in: https://www.linkedin.com/pulse/continuous-integrationdeliverycicd-heroku-shweta-soparkar-

Soroker, T. (6. Dezember 2017). *Pivotal Cloud Foundry vs Kubernetes: Choosing The Right Cloud-Native Application Deployment Platform*. Abgerufen am 29. Januar 2018 von Takipi Blog: https://blog.takipi.com/pivotal-cloud-foundry-vs-kubernetes-choosing-the-right-cloud-native-application-deployment-platform/

Spazzoli, R. (9. Mai 2017). *Managing Secrets on OpenShift – Vault Integration*. Abgerufen am 10. März 2018 von OpenShift Blog: https://blog.openshift.com/managing-secrets-openshift-vault-integration/

Startplatz. (kein Datum). *InsurTech*. Abgerufen am 30. Oktober 2017 von Startplatz: http://www.startplatz.de/startup-wiki/insurtech/

Statista GmbH. (2017). *Größte Versicherungen weltweit nach der Bilanzsumme im Jahr 2016*. Abgerufen am 4. März 2018 von Statista: https://de.statista.com/statistik/daten/studie/169732/umfrage/groesste-versicherungen-weltweit-nach-bilanzsumme/

Stine, M. (2015). *Migrating to Cloud-Native Application Architectures.* Sebastopol: O'Reilly Media.

Takai, D. (2017). *Architektur für Websysteme.* München: Hanser.

The Linux Foundation. (2018). *What is Kubernetes?* Abgerufen am 18. März 2018 von Kubernetes: https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/

TMate Software. (2018). *SVN to Git Migration*. Abgerufen am 4. März 2018 von SubGit: https://subgit.com/

Vaske, H. (25. Juli 2016). *Der große Cloud-Computing-Check*. Abgerufen am 26. November 2017 von Computerwoche : https://www.computerwoche.de/a/der-grosse-cloud-computing-check,3314540

Weißer, J. (16. März 2018). Die Cloud Umgebung der Allianz Deutschland AG. (Y. Boley, Interviewer) Stuttgart.

Wiggins, A. (2017). *Abhängigkeiten*. Abgerufen am 2. Oktober 2017 von The Twelve-Factor App: https://12factor.net/de/dependencies

Wiggins, A. (2017). *Admin-Prozesse*. Abgerufen am 2. Oktober 2017 von The Twelve-Factor App: https://12factor.net/de/admin-processes

Wiggins, A. (2017). *Bindung an Ports*. Abgerufen am 2. Oktober 2017 von The Twelve-Factor App: https://12factor.net/de/port-binding

Wiggins, A. (2017). *Build, Release, Run*. Abgerufen am 2. Oktober 2017 von The Twelve-Factor App: https://12factor.net/de/build-release-run

Wiggins, A. (2017). *Codebase*. Abgerufen am 2. Oktober 2017 von The Twelve-Factor App: https://12factor.net/de/codebase

Wiggins, A. (2017). *Dev-Prod-Vergleichbarkeit*. Abgerufen am 2. Oktober 2017 von The Twelve-Factor App: https://12factor.net/de/dev-prod-parity

Wiggins, A. (2017). *Die zwölf Faktoren*. Abgerufen am 2. Oktober 2017 von https://12factor.net/de/

Wiggins, A. (2017). *Einweggebrauch*. Abgerufen am 2. Oktober 2017 von The Twelve-Factor App: https://12factor.net/de/disposability

Wiggins, A. (2017). *Konfiguration*. Abgerufen am 2. Oktober 2017 von The Twelve-Factor App: https://12factor.net/de/config

Wiggins, A. (2017). *Logs*. Abgerufen am 2. Oktober 2017 von The Twelve-Factor App: https://12factor.net/de/logs

Wiggins, A. (2017). *Nebenläufigkeit*. Abgerufen am 2. Oktober 2017 von The Twelve-Factor App: https://12factor.net/de/concurrency

Wiggins, A. (2017). *Prozesse*. Abgerufen am 2. Oktober 2017 von The Twelve-Factor App: https://12factor.net/de/processes

Wiggins, A. (2017). *Unterstützende Dienste*. Abgerufen am 2. Oktober 2017 von The Twelve-Factor App: https://12factor.net/de/backing-services

Wilkens, A. (24. Dezember 2013). *IBM übernimmt Rechenzentren der Allianz-Versicherung*. Abgerufen am 31. Januar 2018 von Heise: https://www.heise.de/newsticker/meldung/IBM-uebernimmt-Rechenzentren-der-Allianz-Versicherung-2072282.html

Winn, D. (2016). *Cloud Foundry: The Cloud Native Platform.* Sebastopol: O'Reilly Media.

Wolff, E. (6. März 2017). *Was sind eigentlich Microservices?* Abgerufen am 10. Oktober 2017 von JAXenter: https://jaxenter.de/was-sind-microservices-40571