MASTERARBEIT

Ökonomische und technische Betrachtung von Clouddiensten für die Bereitstellung von White-Label Plattformen

ausgeführt an der

am Studiengang

Wirtschaftsinformatik

Von: Thomas Draxler

Personenkennzeichen: 1910320032

Graz, am 5. Dezember 2020 .....

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

Digitale Plattformen ermöglichen Softwareentwicklungsunternehmen und digitalen Geschäftsmodellen ein schnelles Wachstum. Dies erfordert skalierbare und flexible Softwarelösungen wie White-Label Plattformen die als Software-as-a-Service bereitgestellt werden. Das Ziel dieser Arbeit ist, Software Architekturen für digitale Plattform zu analysieren und wie Plattformen durch die Verwendung von Clouddiensten bereitgestellt werden können. Die Analyse untersucht drei gängige Software Architekturen und vergleicht die Architekturen für unterschiedliche Ausprägungen von digitalen Plattformen. Die Auswertung zeigte, dass eine Serverless-Architektur die beste Möglichkeit für die Bereitstellung einer Plattform ist, die unregelmäßig verwendet wird oder für Plattformen bei denen die Kunden stark reguliert sind und eigene Instanzen benötigen. Eine Microservice Architektur bietet sich für Systeme mit einer hohen Auslastung an oder für Plattformen, die eine große Anzahl an Benutzer benötigen. Zusammenfassend kann gesagt werden, dass Clouddienste Unternehmen eine einfache Möglichkeit bieten um flexible digitale Geschäftsmodelle zu entwickeln, die gegebenenfalls uneingeschränkt skaliert werden können und in kürzester Zeit auf die Bedürfnisse des Marktes angepasst werden können.

Abstract

Digital platform solutions support rapid growth in software development and digital businesses. A scalable and flexible solution, such as Software-as-a-Service (SaaS) white-label platforms, is required. This thesis analyses digital platforms' architecture and identifies useful cloud services. The research considers different platforms' characteristics and how they affect cloud infrastructure selection. Three standard software architectures are compared for different platforms' characteristics. The comparison indicates that serverless architecture is ideal for irregularly used platforms or those with strict restrictions between customers. Microservice architecture is preferable for systems with a high load and number of users. Overall, cloud services enable companies to build large-scale flexible digital business models with a rapid time-to-market that allows quick response and adaptation to economic pressures.

Inhaltsverzeichnis

1 Einleitung

Die Anforderungen an eine Softwarelösung haben sich in den letzten Jahren laufend verändert und die Relevanz für den Einsatz von digitalen Diensten nahm erheblich zu. Aufgrund dessen, bieten die meisten Softwareanbieter deren Software als Software-as-a-Service in der Cloud an. Der Einstieg in die digitale Welt wird für Unternehmen mit wenig IT-Knowhow erleichtert, da keine eigene IT-Infrastruktur mehr benötigt wird. Dies stellt Softwareentwicklungsunternehmen vor Herausforderungen, passende Infrastrukturlösungen für deren Projekte auszuwählen und zu adaptieren.

Ein weiterer Trend zeichnet sich dadurch ab, dass digitale Services genutzt werden, um Dienste oder Produkte eines Unternehmens anzubieten, aber auch Dienste von verschiedenen Unternehmen über eine Plattform bereitzustellen. Digitale Plattformen werden genutzt, um Synergien mit anderen Unternehmen zu schaffen und von einem gegenseitigen Wissensaustausch zu profitieren. Dieser Anwendungsbereich von Software führt zu weiteren Herausforderungen für die Entwicklung der digitalen Plattformen. Mehrere Unternehmen verwenden eine gemeinsame Plattform, wobei jedes Unternehmen unterschiedliche Anforderungen oder Wünsche haben könnte. Für diesen Anwendungsfall bieten Technologieunternehmen individualisierbare Standardprodukte oder White-Label Lösungen an, um die Software auf die verschiedenen Anforderungen anzupassen.

1.1 Aufgabenstellung

Die Ausgangssituation für digitale Plattformen und deren Bereitstellung wurde im vorhergehenden Abschnitt beschrieben. Die primäre Aufgabenstellung dieser Arbeit besteht darin, durch eine Literaturrecherche und Analyse von Referenz-Architekturen eine ökonomische und technische Bewertung von Clouddiensten für die Bereitstellung von White-Label Plattformen durchzuführen. Aufgrund dieser Anforderungen unterteilt sich die Problemstellung auf die zwei Bereiche digitale Plattformen und Clouddienste. Es werden die folgenden Aspekte in der Arbeit erläutert:

- Was sind digitale Plattformen und wie werden diese eingesetzt?

- Welche Software-Konzepte werden eingesetzt, um eine Plattform für mehrere Unternehmen bereitzustellen?

- Welche Clouddienste werden aus ökonomischer und technischer Perspektive empfohlen, um diese Konzepte dem Kunden bereitzustellen?

- Welche Kriterien führen dazu, die jeweiligen Dienste von Cloudanbietern zu verwenden?

1.2 Zielsetzung der Arbeit

Die Bereitstellung einer Applikation als Software-as-a-Service erfordert während der Konzeption einige Entscheidungen aus der langfristige erhebliche Auswirkungen auf die Software resultieren. Die Zielsetzung der Masterarbeit ist, auf Basis von erarbeiteten Softwarekonzepten für individualisierbare Standardprodukte oder White-Label-Software Lösungen von digitalen Plattformen, Empfehlungen für die Bereitstellungsmöglichkeiten durch den Einsatz von Clouddiensten auf Basis von wirtschaftlichen und technischen Kriterien zu erstellen. Das Ergebnis soll die Entscheidungsfindung für die Auswahl einer Cloudlösung für ein Softwareentwicklungsunternehmen erleichtern, um Plattformen für deren Kunden bereitzustellen und Möglichkeiten zu bieten, um die Plattform kundenspezifisch zu erweitern. Die Arbeit umfasst keine detaillierte Software Architektur, um eine Plattform zu implementieren, wie beispielsweise Implementierungsvorschläge für eine Multi-Tenancy Funktion oder Methoden für die Integration von Userinterfaces. Es werden Solution Designs und Konzepte erläutert, anstatt auf spezifische Implementierungen von Softwarelösungen einzugehen.

Durch die Betrachtung der Ausgangssituation und der resultierenden Herausforderungen bei der Umsetzung einer digitalen Plattform als Software-as-a-Service, ergibt sich folgende Forschungsfrage: „Welche Clouddienste sind für die Bereitstellung von digitalen White-Label-Plattformen aus ökonomischer und technischer Perspektive empfehlenswert?“

1.3 Vorgehen und Methodik

Die zentrale Fragestellung dieser Arbeit befasst sich mit der Bereitstellung von digitalen Plattformen, die durch eine ökonomische und technische Betrachtung erarbeitet

wird. Diese Arbeit umfasst die zwei Bereiche: das Konzept von White-Label Plattformen und Clouddiensten. Die Basis des Ergebnisses wurde durch eine literarische Recherche erarbeitet. Es wurde eine Literaturrecherche gewählt, um eine unternehmensunabhängige Grundlage zu erhalten. Die Literatur bietet eine objektive Betrachtung der Ausgangssituation. Aus diesem Grund wurden die Daten nicht durch eine Umfrage oder ein Experteninterview erhoben. Expertenwissen von unternehmensspezifischen Anforderungen kann die gesamtheitliche Betrachtung von digitalen Plattformen beeinflussen.

Zu Beginn werden im Kapitel 2 grundlegende Begriffe erläutert, um ein einheitliches Verständnis der darauffolgenden Analyse zu erhalten. Das folgende Kapitel 3 analysiert durch die Literaturrecherche, die Grundlage für den Entwurf der Bewertung und der Konzeption von Plattformen. Die Analyse betrachtet mögliche Ausprägungen von Plattformen und welche Anpassungsmöglichkeiten White-Label Lösungen bieten können. Die Recherche befasst sich zudem mit Architekturen, die sich bei digitalen Plattformen ergeben. Abschließend werden bei der Analyse Clouddienste und Bereitstellungsmöglichkeiten für Plattformen erarbeitet.

Die Ergebnisse der zuvor erstellten Literaturrecherche werden dazu genutzt, um im Kapitel 4 die beiden Themengebiete, White-Label Software Konzepte und Clouddienste, zusammenführen. Es werden Plattform Architekturen für ein Einsatzgebiet von Plattformen in den unterschiedlichen Ausprägungen durch Referenz-Architekturen von namhaften Unternehmen wie Microsoft oder Amazon erstellt. Referenz-Architekturen wurden gewählt, um eine bereits verifizierte Architektur zu verwenden, zu der es bereits ein Sizing der dazu benötigten Hardwareressourcen für die genutzten Clouddienste gibt. Aus diesem Grund wurden als Teil dieser Arbeit keine weitere Validierungen und Performancetests einer individuellen Implementierung durchgeführt. Die Auswertung der erstellten Konzepte und Kriterien erfolgt im Kapitel 5. Die Bewertung erfolgt auf Grund der Analyse der Plattformen aus der Literaturrecherche. Die Kriterien werden somit objektiv für die jeweiligen Architekturen bewertet. Die Bewertung erfolgte unabhängig von den Architekturen, es wurde kein Vergleich bei der Bewertung durchgeführt. Das Ergebnis der Bewertung kann jedoch anschließend als Vergleich genutzt werden. Eine weitere Betrachtung bei der Auswertung ist die ökonomische Betrachtung, die durch eine Kostenberechnung der Ausprägungen der Plattform erstellt wurde. Die Berechnung erfolgt mit dem Cloudanbieter mit dem größten Marktanteil. Das Ergebnis der gesamten Analyse und Auswertung ist eine Entscheidungsmatrix für unterschiedliche Architekturen von White-Label Plattformen die mit Clouddiensten bereitgestellt werden.

Abschließend wird im Kapitel 6 das Ergebnis zusammengefasst und reflektiert. Zudem wird ein Ausblick auf mögliche Ergänzungen und weiterführende Arbeiten gegeben.

2 Grundlagen und Definitionen

In diesem Kapitel werden grundlegende Begriffe und Definitionen erläutert, die für diese Arbeit relevant sind. Es werden Eigenschaften und Konzepte für die Bereitstellung von Softwareanwendungen erläutert, sowie die Revolution der Softwareentwicklung in den letzten Jahren. Abschließend wird darauf eingegangen was eine digitale Plattform ist und was der Begriff White-Label in Bezug auf Software bedeutet.

2.1 Softwarebereitstellungsmöglichkeiten

Die Bereitstellung einer Software als Software-as-a-Service kann in verschiedenen Varianten erfolgen. Es wird dabei zwischen einem Cloud Service Model und einem Cloud Deployment Model unterschieden. Das Service Model definiert welche Dienste genutzt werden, um eine Software betreiben zu können. In den letzten Jahren hat es sich dahin entwickelt, dass sämtliche Dienste als Service („Something-as-a-Service“) angeboten werden, wie beispielsweise Database-as-a-Service, Künstliche Inteligent-as-a-Service. Die Hauptkategorien der Service Modelle sind klassische Datencenter, Infrastructure-as-a-Service, Plattform-as-a-Service und Software-as-a-Service. Das Deployment Model definiert wie die Dienste bereitgestellt werden. Diese werden in vier Modellen gruppiert, Private Cloud, Public Cloud, Hybrid Cloud und Community Cloud. Die Modelle unterscheiden sich dabei wo sich die Infrastruktur befindet bzw. wer Zugriff auf die Infrastruktur hat. Als Anbieter bzw. Betreiber des Dienstes müssen nicht zwingend Clouddienste für die Bereitstellung genutzt werden. Der Dienst kann auch mit Hilfe einer eigenen Infrastruktur bereitgestellt werden und für einen Kunden als Clouddienst zur Verfügung gestellt werden (Klaffenbach, Klein & Sundaresan, 2019).

2.1.1 Cloud Service Modelle

Cloud Service Modelle werden wie bereits genannt in drei Modelle aufgeteilt, Infrastructure-as-a-Service (IaaS), Plattform-as-a-Service (PaaS) und Software-as-a-Service

(SaaS). Zusätzlich gibt es die Möglichkeit, wenn keine Clouddienste verwendet werden sollen, ein Softwaresystem in einem eigenen Datencenter laufen zu lassen, auch On-Premises Lösung genannt. In diesem Zusammenhang wäre eine On-Premises Lösung mit den meisten Wartungsarbeiten verbunden, da alle Infrastrukturebenen von Netzwerk-, Speicher, Server- und Software-Konfigurationen selbst gewartet und verwaltet werden müssen. Bei Clouddiensten werden die verschiedenen Ebenen an den Cloudprovider übergeben, wie beispielsweise bei Iaas wird ein Server bereitgestellt auf dem das Speicher- und Netzwerkmanagement vom Provider übernommen wird bis hin zu einem SaaS-Dienst, bei dem der Provider die gesamte Wartung übernimmt. Die Abbildung 2.1 zeigt die unterschiedlichen Modelle und die verschiedenen Ebenen die verwaltet werden müssen bzw. bereitgestellt werden. Bietet ein Unternehmen eine eigene Software als SaaS-Lösung an, bedeutet dies nicht, dass die eigene Architektur keine weiteren SaaS-Lösungen intern verwenden kann. Die verschiedenen Service Modelle können für unterschiedliche Anwendungszwecke eines Systems kombiniert werden. Es kann beispielsweise eine bereitgestellte Infrastruktur (IaaS) von Amazon (EC2) für die Ausführung der Applikation genutzt werden, eine Datenbank die als Plattform-as-a-Service bereitgestellt wird, beispielsweise eine Microsoft Azure SQL Database, verwendet werden und für die Überwachung der Systeme wird Datadog als SaaS-Lösung eingebunden (Salam, Gilani & Haq, 2015).

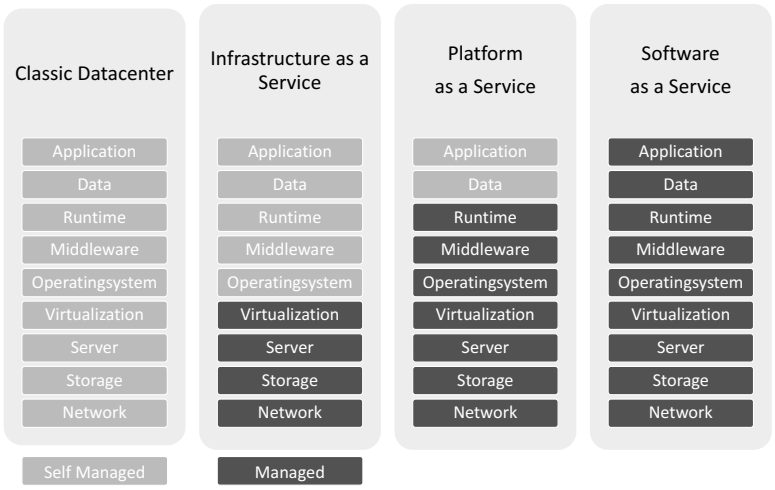

Abbildung 2.1: Übersicht der Cloud Service Modelle (in Anlehnung an Klaffenback, 2019)

2.1.2 Cloud Deployment Modelle

Im vorherigen Abschnitt wurde erläutert welche Services genutzt werden können, um ein System zu betreiben, in diesem Abschnitt wird ergänzt, wie diese Services bereitgestellt werden können. Die eingeschränkteste Variante ist eine Private Cloud, die Infrastruktur bzw. die Umgebung wird nur intern für ein Unternehmen genutzt. Es kann einerseits eine selbstgewartete Hardware sein, aber auch ein extern gehostetes

System für ein Unternehmen, welches dediziert für ein Unternehmen genutzt wird und der Zugriff nur für dieses Unternehmen ermöglicht wird. Ein Produkt wäre beispielsweise OpenStack, ein System das On-Premises genutzt werden kann oder von einem Cloudprovider (Infrastructue-as-a-Service) bereitgestellt werden kann. OpenStack ist ein Open-Source Software Projekt, um eine Infrastruktur zu verwalten und Instanzen von virtuellen Maschinen bereitzustellen. Clouddienste, die öffentlich verfügbar sind, werden Public Cloud genannt, wie beispielsweise Clouddienste von Amazon Web Services (AWS). Die Cloud-Infrastruktur bzw. die Dienste sind für jegliche Unternehmen verfügbar und nicht auf ein Unternehmen eingeschränkt. Eine weitere Möglichkeit ist es eine Private Cloud und Public-Cloud zu kombinieren. Ein Beispiel dazu wäre, dass Daten von bestimmten Services nur intern gespeichert werden (Private Cloud), aber Clouddienste einer Public Cloud rufen Daten von diesem Service ab, dieses Konzept wird Hybrid Cloud genannt. Ein praktisches Beispiel wäre, das ein Microsoft Active Directory Dienst in einer Private Cloud läuft, aber Clouddienste von Microsoft Azure Dienste den Active Directory Dienst integrieren. Die vierte Variante ist eine Community Cloud, dies ist eine spezielle Form der Public Cloud, die nur für eine eingeschränkte Zielgruppe verfügbar ist, wie beispielsweise den öffentlichen Dienst. Ein weiterer Begriff in Kombination mit Public-Cloud ist eine Multi-Cloud, dabei handelt es sich um ein System, das Dienste von verschiedenen Cloudprovidern verwendet. Die Abbildung 2.2 zeigt die unterschiedlichen Cloud Deployment Modelle und wie diese interagieren (Klaffenbach et al., 2019).

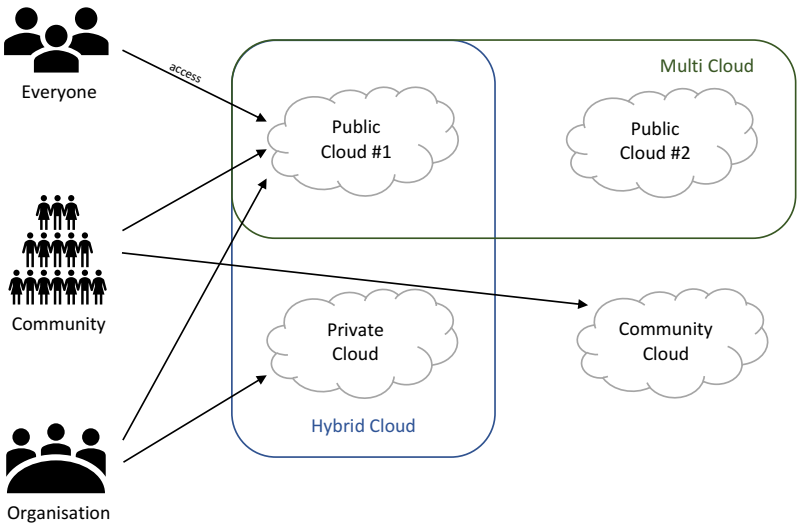

Abbildung 2.2: Übersicht der Cloud Deployment Modelle (in Anlehnung an Klaffenback, 2019)

2.2 Revolution von Softwareentwicklung

Die Relevanz für die Verwendung von Software im Geschäftsumfeld stieg über die letzten Jahre bzw. Jahrzente erheblich. Durch diesen Aufschwung hatten am Anfang große Unternehmen wie Google oder Amazon einen großen Vorteil, da bereits viel Eigenentwicklung und Optimierungen durchgeführt worden sind. Durch eine stark wachsende Community werden diese Erfahrungen ausgetauscht und veröffentlicht. Der Community-Gedanke ermöglicht es, dass kleine wie auch große Unternehmen, gegenseitig voneinander profitieren können. Die Veränderung der Softwareentwicklung über die letzten Jahre ist nicht nur auf Technologie basierend, es bezieht die verschiedenen Gegebenheiten eines Unternehmens mit ein. Es hat sich die gesamte Vorgehensweise und das Mindset verändert bzw. wird dies in der Vorgehensweise mehr integriert und ist nicht nur rein technologisch getrieben (Laszewski, Arora, Farr & Zonooz, 2018). Die aktuelle Transformation der Softwareentwicklung nennt sich Cloud Nativ. Diese Transformation zielt immer mehr auf bereitgestellten Infrastruktur Komponenten ab, damit sich die Entwicklung auf die Umsetzung der Anforderungen fokussieren kann. Pini Reznik, Jamie Dobson und Michelle Gienow haben sich mit dieser Transformation auseinandergesetzt und eine sogenannte „Maturity Matrix“ bzw. ein Reifegrademodell der Softwareentwicklung erstellt. Die Abbildung 2.3 zeigt diese Matrix, in der die Reifegrade eines Unternehmens für die unterschiedlichen Bereiche der Softwareentwicklung abgelesen werden können. Die Entwicklung begann mit einer Vorgehensweise, die wenig Planung und ohne Prozesse ausgeführt wurde, über starre Prozesse mit dem Wasserfallmodel zu einer iterativen bzw. agilen Vorgehensweise bis hin zu einem experimentelleren Vorgehen mit Cloud Native und deren Weiterentwicklungen. Die verschiedenen Phasen wurden in mehrere Kategorien aufgeteilt, die Einfluss auf den Reifegrad haben, wie beispielsweise die Kultur, der Teamaufbau, die Architektur oder Infrastruktur, mit der die Software bereitgestellt wird. Die unterschiedlichen Kategorien und Reifegrade werden im nächsten Abschnitt noch genauer erläutert, zu beachten ist dabei, dass es kein starres Model ist. Der Reifegrade eines Unternehmens kann sich in den unterschiedlichen Kategorien in unterschiedlichen Phasen befinden, wie beispielsweise wird eine Cloud Native Infrastruktur und Provisionierung verwendet, jedoch mit einem agilen / iterativen Vorgehen gearbeitet (Reznik, Dobson & Gienow, 2019).

2.2.1 Vorgehensweise – Kultur, Design, Team, Prozess

Dieser Abschnitt beschäftigt sich mit den vier Kategorien, Kultur, Produkt und Service Design, Team und Prozesse. Das Zusammenspiel der vier Kategorien bildet die Vorgehensweise in den unterschiedlichen Phasen.

Die erste Phase, „Kein Prozess“, zeichnet sich dabei ab, wie der Name bereits aussagt, dass die verschiedenen Anforderungen ohne ein vordefiniertes System und individu-

Stage	KEIN PROZESS	WASSERFALL	AGILE	CLOUD NATIVE	ZUKUNFT
KULTUR	Individuell	Voraussagend	Iterative	Collaborative	Experimentell
PROD/SERVICE DESIGN	Beliebig	Langzeit Plan	Getrieben von Anforderung	Getrieben von Daten	Alle Faktoren werden mit einbezogen
TEAM	Keine Organisation, Einzelne Mitarbeiter	Hierarchisch	Funktionsübergreifende Teams	DevOps / SRE	Interne Supply-Chains
PROZESS	Zufällig	Wasserfall	Agile (Scrum / Kanban)	Design Thinking + Agile + Lean	Verteilt, selbstorganisiert
ARCHITEKTUR	Entwicklung mit Trial und Error	Monolithisch	Client Server	Microservices	Functions
MAINTENANCE	Reagieren auf Kundenbeschwerten	Ad-hoc Monitoring	Automatische Benachrichtigungen	Beobachtung des gesamten Systems & Self-Healing	Vorbeugende Maschine Learning, KI
BEREITSTELLUNG	Unregelmäßige Releases	Periodische Releases	Continous Integration	Continous Delivery	Continous Deployment
PROVISIONIERUNG	Manuell	Scripted	Konfigurationsverwaltung (Puppet/Chef/Ansible)	Orchestration (Kubernetes)	Serverless
INFRASTRUKTUR	Einzelner Server	Mehrere Server	VMs	Container / Hybrid-Cloud	Edge-Computing
			Aktuelle Situation		Ziel

Abbildung 2.3: Maturity Matrix der Softwareentwicklung(in Anlehnung an Reznik, 2019)

ell bearbeitet werden. Es wird vorwiegend bei kleinen Unternehmen eingesetzt da es bei wachsenden Unternehmen zu Problem führt, wenn jeder Mitarbeiter zu einem bevorzugten Mitarbeiter geht, um an die benötigte Information zu gelangen. Dasselbe Vorgehen wird meist beim Produkt und Service Design angewendet. Neue Ideen werden von einzelnen Mitarbeitern erarbeitet und umgesetzt, was zu unterschiedlichen Lösungen für dieselben Anforderungen führen kann und kaum zu Wissensaustausch bzw. Produktoptimierung führt. Dieses Vorgehen kann bei kleinen Unternehmen oder Startup durchgeführt werden, da meist nur eine niedrige Anzahl an Entwickler an der Umsetzung arbeiten und kein striktes Management vorhanden ist (Wiegers, 2013).

Im Gegensatz dazu wird beim Wasserfallmodell nach einem strikten Prozess gearbeitet. Die Planung der Umsetzungen wird langfristig durchgeführt, bei dem am Anfang des Projekts genau ein Design für das gesamte System definiert wird und vorausgeplant wird. Die Produkte oder Services werden auf Basis von bestehenden Kunden, potenziellen Kunden und den Benutzern des Systems konzipiert. Das Vorgehensmodell verwendet einen sequenziellen Prozess bei dem nach der Planung, die Umsetzung, Testen und zu Ende die Auslieferung erfolgt. War ein Prozessschritt nicht erfolgreich, wird erneut mit der Planung begonnen, um anschließend den gesamten Prozess erneut durchführen zu können. Die Entwicklung der Software wird hierarchisch geleitet, somit erfolgt Umsetzung in organisierten Teams, jedoch erfolgt die Kommunikation zwischen den Teams meist über einen Manager bzw. Teamleiter. Die Teamaufteilung erfolgt funktionsbezogen wie beispielsweise eine Aufteilung in Entwickler und Tester (Specker, 1998).

Um die Durchlaufzeit und den Planungsaufwand zu reduzieren wird bei einer agilen Vorgehensweise wie Scrum oder Kanban, eine Planung für kleinere und kurzfristigere Anforderungen vorgenommen. Dieses Vorgehen führt dazu flexibler und schneller auf Änderungen reagieren zu können. Durch eine funktionsübergreifende

Teamkonstellation entsteht eine flache Hierarchie und eine offene Arbeitskultur. Ein Team beinhaltet die verschiedenen Rollen, die für die Entwicklung einer Software benötigt werden wie beispielsweise Architekten, Entwickler und Tester. Die Produktentwicklung erfolgt auf Basis der aktuell benötigten Features, da durch die schnelle Reaktionszeit auf die Anforderungen, die aktuellen Gegebenheiten regelmäßig angepasst werden können (Balzert & Ebert, 2008).

Der Trend der Vorgehensmodelle ist bereits ersichtlich, da versucht wird, die verschiedenen Rollen und Aufgabenbereiche immer mehr zu vereinen und somit auch das Vertrauen in die Mitarbeiter zu zeigen. Dieser Wechsel des Mindsets wirkt sich einerseits bei der Arbeitskultur mit mehr Verantwortung der einzelnen Mitarbeiter aus, aber wirkt sich auch bei der Entwicklung der Produkte und Services mit Cloud Native aus. Produkte werden auf Basis von gewonnenen Daten für Bereiche designt, bei denen ein Kundenbedürfnis vorhanden und auch bekannt ist. Durch den großen Funktionsumfang eines Teams wird es ermöglicht in kürzester Zeit neue Ideen zu Entwickeln und diese in Form eines Proof of Concepts bei den Kunden zu testen, um frühzeitig die Akzeptanz und Relevanz eines Produkts evaluieren zu können. Das Risiko von großen Projekten wird mit diesem Vorgehen versucht zu reduzieren, da wie bereits erwähnt wurde, ein kleiner Teil des gesamten Projekts umgesetzt und getestet wird, bevor die Realisierung des gesamten Systems erfolgt. In Zukunft wird dieser Trend noch weiter gehen, da durch künstliche Intelligenz und die vorhandenen Daten, Produkte spezifischer auf Kundenbedürfnissen definiert werden können. Durch einen unabhängigen und klar abgegrenzten Funktionsbereich wird es ermöglicht, dass die Entwicklung der Produkte verteilter durchgeführt werden kann. Der Einsatz von Clouddiensten sollte es erleichtern den Aufwand und das Risiko für die Entwicklung von Prototypen zu reduzieren, dazu mehr im nachfolgenden Abschnitt über die Bereitstellung und Infrastruktur (Reznik et al., 2019).

2.2.2 Umsetzung der Software – Architektur, Maintenance, Bereitstellung, Infrastruktur

Der vorherige Abschnitt befasste sich mit der organisatorischen Vorgehensweise der Softwareentwicklung, in diesem Abschnitt geht es darum wie eine Software in den verschiedenen Reifegraden aufgebaut, gewartet und bereitgestellt wird.

Die vorhin beschriebenen Vorgehensweisen spiegeln sich in der technischen Umsetzung wider. Die Architektur berücksichtigt kaum Erweiterbarkeiten für mögliche zukünftige Anforderungen, wenn zuvor kaum eine Planung durchgeführt und die Software nach Belieben erstellt bzw. auf den aktuellen Anforderungen erstellt wurde. Die Wartung und Bereitstellung erfolgen vorwiegend manuell und unregelmäßig nach Bedarf des Kunden. Wohingegen der langfristige Plan des Wasserfallmodells zu einer monolithischen Architektur führt, da bereits am Anfang die Planung für das

gesamt System erfolgt und jeweiligen Entwicklungsschritte in die Architektur einfließen. Die Laufzeit eines Projekts ist in den meisten Fällen dementsprechend länger. Die Releases werden periodisch geplant und um die Infrastruktur für eine langen Zeitraum dimensionieren zu können, wird diese meist für eine hohe Last vorbereitet. Die Provisionierung des Systems erfolgt meist von selbst erstellten Skripts und um gewähren zu können, dass das System ordnungsgemäß läuft wird das System auf kritische Probleme laufend überwacht (Balzert, 2011).

Agile Vorgehensmodelle versuchen dabei eine Architektur zu schaffen, bei der die Entwicklung parallel und in kurzen Intervallen durchgeführt werden kann. Es werden verteilte Systeme oder eine Microservice Architektur eingesetzt. Bei Fehlern oder Problemen am System werden automatisiert Benachrichtigungen versendet. Die kurzen Release Intervalle erfordern täglich eine hohe Qualität der Software, um die Qualität überprüfen zu können wird Continous Integration verwendet, bei der die Software automatisiert getestet und validiert wird. Durch die regelmäßigen Releases gibt es mehrere Versionen, die potentiell gleichzeitig laufen, somit werden mehrere Systeme benötigt auf denen die Software läuft. Um dies zu vereinfachen werden VMs und Konfigurationsverwaltungs-Tools verwendet um System automatisiert zu deployen. Die Weiterentwicklung bei Cloud Native verwendet dazu Container und Orchestrations-Tools, um die Software systemunabhängig zu erstellen und somit eine weitere Automation zu ermöglichen. Continous Delivery führt dazu, dass Änderungen nicht nur überprüft werden, sondern auch automatisiert auf ein System deployt werden. In Kombination mit einer Überwachung des gesamten Systems und automatisierten Systemkorrektion, kann sich das System teilweise selbstheilen, um den Wartungsaufwand zu verringern. Wie bereits bei der organisatorischen Vorgehensweise ersichtlich war, zeigt die technische Umsetzung einer Software einen Trend hin zu einer Umgebung, bei der Softwareentwickler sich mit der Umsetzung der Anforderungen beschäftigt, die durch bereitgestellte Services unterstützt wird. Es wird eine bereitgestellte Infrastruktur genutzt, auf der die Anforderungen in Funktionen implementiert werden und somit einen Großteil der Wartung an Partner bzw. Cloud-Providern abgegeben werden kann (Reznik et al., 2019).

Die Erläuterung der verschiedenen Reifegrade der Softwareentwicklung zeigt wie sehr sich die Vorgehensweise und Umsetzung sich über dem Lauf der Zeit weiterentwickelt hat. Diese Arbeit beschäftigt sich dabei vorwiegend mit der Phase Cloud Native und den zukunftsorientierten Vorgehensweisen mit Clouddiensten und Serverless-Architekturen. Wie bereits hervorging sind nicht nur technologische Faktoren zu beachten, sondern auch die organisatorische Vorgehensweise bzw. die Denkweise bei der Konzeption sind relevant.

2.3 Solution Architektur

Dieser Abschnitt erläutert die Definition der Solution Architektur, da diese Arbeit primär die Solution Architektur einer Softwareanwendung mit einem SaaS Model behandelt. Die Solution Architektur befasst sich mit unterschiedlichen Aspekten einer Geschäftslösung, es brachtet die Lösung als System in einer strategischen und taktischen Perspektive. Dies bedeutet, ein Solution Architekt befasst sich nicht nur mit der Softwarelösung, sondern muss strategische Faktoren wie Budget oder Meilensteine mit betrachten. Die folgenden Aspekte zeigen, die für eine Organisation bei der Erstellung oder Einführung eins neuen Systems relevant sein sollten und bei der Konzeption der Solution Architektur beachtet werden sollen:

- Business Anforderungen, Funktionale und Nicht-Funktionale Anforderungen werden auf den technischen Möglichkeiten evaluiert
- Anforderungen der Benutzer, wie wird der Enduser das Produkt verwenden und welche Funktionalitäten werden benötigt?
- Betrachtung von Regularien, wie Datenschutzgrundverordnung oder ISO Standards
- Übersicht eines möglichen Lösungsansatzes, der Solution Architekt erstellt eine High-Level Übersicht der unterschiedlichen Komponenten, die für das Produkt relevant sind und wie diese integriert werden können, um die Planung und Schätzungen genauer zu machen und später mit den tatsächlichen Zahlen evaluieren zu können.
- IT Infrastruktur, es muss betrachtet werden wie der Lösungsansatz mit verschieden IT-Ressourcen bereitgestellt werden kann
- Technologie Auswahl, definiert welche Technologien für die Umsetzung eingesetzt werden, wie zum Beispiel die Verwendung einer zugekauften Software Anwendung oder einer Eigenimplementierung von verschiedenen Komponenten
- Wartung der Solution, es wird nicht nur betrachtet wie die Lösung umgesetzt werden kann, sondern auch wie diese langfristig eingesetzt werden kann, wie beispielsweise mögliche Skalierung der Anwendung oder Softwarewartung oder Disaster Recovery.
- Kostenschätzung, auf Basis des Lösungsansatzes, wird eine Kostenschätzung für die Investitions- und Betriebskosten gegeben

- Projektzeitpläne, die Anforderungen werden geschätzt, um eine Basis für die Projekt Meilensteine liefern zu können

- Benötigtes Personal, welche Personen werden für die Umsetzung benötigt werden und wie werden diese zusammenarbeiten (Distributed Teams)

Die unterschiedlichen Aspekte zeigen, dass die Solution Architektur wirtschaftliche und technische Anforderungen kombiniert. Die Solution Architektur hat meist bereits in frühen Verhandlungsphasen (Pre-Sales) eine Relevanz, um potenzielle Kunden bereits in der Anfangsphase, ein realistisches Konzept für eine mögliche Umsetzung mit einem Kosten und Zeitplan liefern zu können (Shrivastava, Srivastav & Arora, 2020).

2.4 White-Label Produkt

White-Label Produkte in Bezug auf digitale Produkte oder Services sind Software Produkte, die von einem Softwarelieferant bereitgestellt werden und in einem unternehmenseigenen Branding genutzt werden können oder auch weiterverkauft werden können. Es werden dabei unterschiedliche Geschäftsmodelle genutzt, ein Unternehmen verkauft die Software unter dem eigenen Namen (Rebranding) oder eine Möglichkeit ist, dass die White-Label Software als Dienst genutzt wird und dem Endkunden als unternehmenseigener Dienst bereitgestellt wird. Der Vorteil einer solchen Lösung ist, dass die vorwiegende Wartung der Software beim Softwarehersteller ist und das Unternehmen kein Fachwissen für die Entwicklung und Betreibung benötigt. White-Label kann verschiedene Ausprägungen der Software bedeuten, es beginnt mit einer statischen Software, die im Firmeneigenen Design adaptiert werden kann, bis hin zu Standardprodukten, die für das Unternehmen erweitert werden können (Kelly, 2012).

2.5 Digitale Plattform

Den Begriff „Plattform“ gibt es in unterschiedlichen Ausprägungen, in Bezug auf digitale Plattformen wird dieser von der IT geprägt. Unternehmen wie Microsoft, Amazon, Walmart oder viele andere kleine und große Hardware- und Softwareanbieter betreiben Geschäftsmodelle auf Basis einer Plattform. Die Plattform bildet die zentrale Komponente des Geschäftsmodells, es werden verschiedene Produkte, Services oder Technologien in einem System vereint. Die Plattform sollte dabei Kunden / Unternehmensproblem von mehreren Unternehmen in einer Branche lösen. Dadurch kann eine digitale Plattform angeboten werden, die die Bedürfnisse von mehreren Unternehmen in einem System löst und zusätzlich noch Synergien zwischen

den Unternehmen bieten kann. Die Plattform bildet durch den zentralen Wissensgewinn von unterschiedlichen Unternehmen ein Ökosystem. Deshalb werden digitale Plattformen mit Netzeffekten in Verbindung gebracht, da sich der Wert einer Plattform erhöht, je mehr Benutzer die Plattform nutzen. Digitale Plattformen sind nicht nur Softwareprodukte sondern es vereint Geschäftsmodelle, Produktentwicklung und Technologische Architektur und kombiniert ökonomische und technische Herausforderungen (Gawer, 2011).

3 Analyse

Nachdem im Kapitel 2 die wichtigsten Begriffe für die Erarbeitung einer digitalen Plattform und deren Bereitstellung mit Clouddiensten erläutert wurden, werden in diesem Kapitel relevante Eigenschaften und Information für digitale Plattformen und Clouddienste erarbeitet. Dies soll die Grundlage für den darauffolgenden Entwurf von Anwendungen bieten. Es werden mögliche Ausprägungen und Architekturen von Plattformen erläutert sowie eine Übersicht von Cloudanbietern und Architekturausprägungen bei der Bereitstellung von Plattformen in der Cloud gegeben.

3.1 White-Label Lösungen bei digitalen Plattformen

Die Definition einer digitalen Plattform wurde bereits in der Einleitung und bei den Grundlagen erläutert, in diesem Abschnitt wird nun analysiert in welcher Form digitale Plattformen eingesetzt werden und wie sich dabei das System verhält. Wie bereits in der Definition von digitalen Plattformen hervorging, umfasst das Thema nicht nur technische Herausforderungen, sondern muss auch ökonomisch überlegt werden in welcher Ausprägung die Plattform zielbringend ist. Die Analyse geht vorwiegend auf die Grundlage der Forschungsfrage ein, in welcher Form bzw. welche Konzepte von White-Label Softwarelösungen in die digitale Plattform einfließen.

3.1.1 Ausprägungen

Um die Möglichkeiten einer White-Labeling Lösung betrachten zu können, wird zuerst eine Übersicht der unterschiedlichen Ausprägungen von Plattformen aufgezeigt. In der Literatur findet man viele unterschiedliche Definitionen und Klassifizierungen von Plattformen. Betrachtet man die Unterteilungen genauer, wie beispielsweise eine Unterteilung nach einen der beiden folgenden Kategorien, zeigt sich ein einheitliches Konzept, bei dem sich die Zielgruppe unterscheidet (OECD, 2019):

- Funktionalität – Plattformen für Finanztransaktionen (z.B.: PayPal) oder Plattformen für Gastronomie (z.B.: UberEats)

- User – Plattformen für beispielsweise Käufer, Wiederverkäufer oder Softwareentwickler

3.1.1.1 Kundengruppen

Das Konzept der Plattformen unterteilt sich bei genauerer Betrachtung, wie viele unterschiedliche Anwender die Plattform nutzen können. Erfolgt auf der digitalen Plattform eine Interaktion innerhalb einer Gruppe nennt man es One-Sided Plattform. Ein Beispiel für eine One-Sided Plattform ist der Beginn von Facebook. Es war eine Plattform, die eine Interaktion zwischen verschiedenen Studenten ermöglichte. Umfasst die Plattform zwei oder mehrere unabhängige Kundengruppen, spricht man von einer Two-Sided bzw. Multi-Sided Plattform. Um eine Multi-Sided Plattform zu ermöglichen muss die Plattform Interaktionen zwischen unterschiedlichen Kundengruppen ermöglichen. In der Literatur wird der Begriff Two-Sided Plattform genutzt, wenn zwei oder mehrere Kundengruppen in der Plattform interagieren, somit wird zwischen Two-Sided und Multi-Sided nicht unterschieden. Ein Beispiel dazu ist Facebook in der aktuellen Version. Facebook ist eine Plattform, die es einerseits Benutzer ermöglicht untereinander zu kommunizieren bzw. Informationen auszutauschen, aber auch Unternehmen ermöglicht Werbung zu schalten, damit andere Benutzer der Plattform auf Produkte oder Services eines Unternehmens aufmerksam werden. Facebook hätte in diesem Beispiel eine Two-Sided Plattform, da eine Interaktion zwischen der Kundengruppe „Unternehmen“ und dem „Enduser“ erfolgt. Eine digitale Plattform versucht verschiedene Netzeffekte in einem digitalen Produkt abzubilden (Yablonsky, 2018).

Wie bereits erläutert wurde, kann eine digitale Plattform, One-Sided oder Two-Sided sein und durch die entstehenden Netzeffekte innerhalb einer Plattform kann zwischen vier unterschiedlichen Ausprägungen unterschieden werden:

- One-Sided Plattform ohne Netzeffekte – z.B.: Netflix, bietet in einer eigenen Plattform Filme an, die von Kunden konsumiert werden können, jedoch gibt es keinen Austausch zwischen den Usern.

- One-Sided Plattform mit direkten Netzeffekten – z.B.: Whatsapp, bietet eine Plattform, in der User untereinander kommunizieren können

- Two/Multi-Sided Plattform mit indirekten Netzeffekten – z.B.: Amazon (Online Shop), bietet eine Plattform, in der Kunden, Produkte von unterschiedlichen Unternehmen, die deren Produkte im Amazon Online Shop anbieten, kaufen können. Es erfolgt keine Kommunikation zwischen den Kunden bzw. den Unternehmen. Kunden profitieren indirekt davon, dass Amazon von vielen unterschiedlichen Unternehmen anbietet und somit eine große Auswahl an Produkten zur Verfügung haben, ohne dazu auf verschiedenen Online Shops zu suchen.

- Two/Multi-Sided Plattform mit direkten und indirekten Netzeffekten – z.B.: Facebook, bietet eine Plattform, in der sich User untereinander austauschen können und zusätzlich können Unternehmen Werbung in der Plattform schalten. Durch die Interaktion zwischen den Usern entstehen direkte Netzeffekte und indirekte Netzeffekte entstehen, da Unternehmen davon profitieren, je mehr User auf der Plattform sind, dass die Werbungen effektiver sind und mehr Personen anspricht. Die Plattform wird für Unternehmen attraktiver umso mehr User auf der Plattform sind.

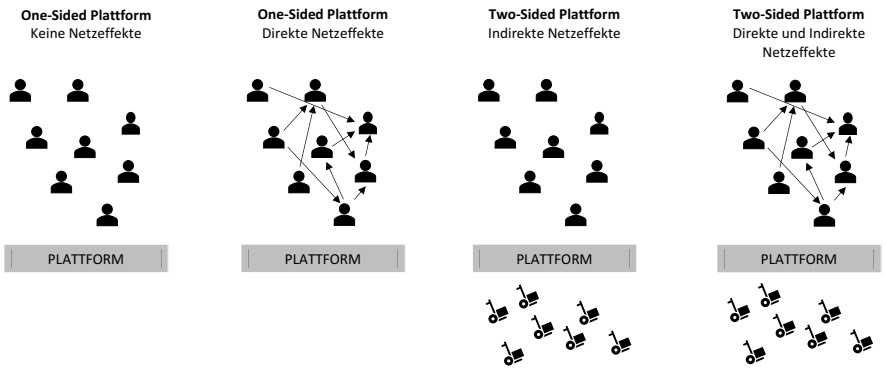

Abbildung 3.1: Übersicht der Ausprägungen einer digitalen Plattform (in Anlehnung an Ecorys, 2016)

In der Abbildung 3.1 sind die unterschiedlichen Ausprägungen der digitalen Plattformen nochmals grafisch zusammengefasst. Es zeigt die Interaktionen zwischen Usern bzw. die indirekten Effekte zu einer anderen Kundengruppe über die Plattform, wie zuvor erläutert wurde.

3.1.1.2 Netzeffekte

Die Konzeption einer neuen Plattform bei der Erarbeitung eines neuen Geschäftsmodells muss nicht zwingend alle Möglichkeiten einer Plattform am Anfang implementieren. Die Plattform kann über einen längeren Zeitraum ergänzt werden. Viele Plattformen starten mit einer Kundengruppe um diese auch maßgeschneidert ansprechen zu können. Durch die Weiterentwicklung und durch mehrere Optimierungen eines Geschäftsmodells kann eine Plattform zu einer Two-Sided Plattform, die indirekte und direkte Netzeffekte nutzt, erweitert werden. Ökonomisch betrachtet ist es nicht nur wichtig wie man einen Nutzen den Kundengruppen mit einer Plattform bieten kann, sondern auch wie ein Geschäftsmodell nachhaltig wirtschaftlich bestehen kann. Die Einnahmequellen einer Plattform sollten deshalb genau definiert

werden. Plattformen können direkt oder indirekt finanziert werden. Umsatzgenerierung durch direkte Bezahlung bedeutet, dass der Benutzer der Plattform eine Gebühr für die Benützung der Plattform bezahlt. Repräsentativ zu diesem Modell kann man Netflix betrachten, bei der Plattform bezahlt jeder User eine monatliche Gebühr in Form eines Abos, somit steht die Plattform nur Benutzern zur Verfügung, die die Gebühr bezahlen. In Gegensatz dazu wird die Plattform mit einer indirekten Finanzierung, gratis zur Verfügung gestellt und die Einnahmen werden durch eine andere Kundengruppe generiert, wie beispielsweise durch Werbungen. Ein Beispiel dazu ist Facebook, die Plattform ist für alle Benutzer gratis, wobei Facebook Einnahmen durch Unternehmensuser generiert, die bezahlte Werbungen auf der Plattform schalten. Aufgrund der unterschiedlichen Vorgehensweisen bei der Umsatzgenerierung sollten diese Kriterien bei der Konzeption der digitalen Plattform mit einfließen. Eine Plattform, die direkte Einnahme erwirtschaftet, kann meistens bereits mit einer kleineren Kundengruppe gewinnbringend sein. Ein Geschäftsmodell, das indirekt Einnahmen durch indirekte Netzeffekte erreichen möchte, benötigt meist eine größere Community bzw. muss eine kritische Masse überwinden, die die Plattform nutzt, um einen Mehrwert für die unterschiedlichen Zielgruppen bieten zu können (Nooren, van Gorp, Eijk & Fathaigh, 2018).

Bei einer ökonomischen Betrachtung eines Plattform Geschäftsmodells, ergibt sich durch die Netzeffekte, die eine Plattform ermöglicht, ein immaterieller Mehrwert für die Plattform mit jedem User. Der Wert der Plattform erhöht sich pro User dadurch meist exponentiell. Im Gegensatz dazu nehmen die Kosten für jeden Benutzer auf einer Plattform mit jedem weitern Benutzer ab. Die Betriebskosten für eine Infrastruktur ist für eine bestimmte Anzahl an User gleich. Aus diesem Grund nehmen die Kosten pro User ab, jedoch funktioniert dies auch nur bis zu einer Grenze, bei der die Infrastruktur weiterskaliert werden muss. Die Abbildung 3.2 stellt den Wert und die Kosten pro User bei der Verwendung einer Plattform gegenüber. Dies ist ein entscheidender Punkt, der bei der weiteren Analyse der Bereitstellungsmöglichkeiten einer White-Labeling Plattform in dieser Arbeit mit einbezogen wird.

3.1.1.3 Anpassungsmöglichkeiten

Die zuvor erläuterten Ausprägungen einer Plattform haben sich auf die Installation und Verwendung von einem Unternehmen als Betreiber der Software bezogen. Das Geschäftsmodell von digitalen Plattformen beruht darauf, dass ein Produkt für unterschiedliche Unternehmen eingesetzt werden kann. Somit werden die zuvor genannten Ausprägungen auf das jeweilige Unternehmen angewendet. Wie in der Begriffsdefinition erläutert wurde, können für diesen Anwendungsfall White-Label Produkte bereitgestellt werden. Der Plattformanbieter kann die Plattform als Software-as-a-Service System einem Unternehmen zur Verfügung stellen. Die Plattform kann darauf im firmeneigenen Design adaptiert werden und das Unternehmen kann somit für die Mitarbeiter oder den Kunden eine eigene Softwareanwendung bereitstellen. Unternehmen können die Vorteile eines One-Sided oder Multi-Sided Plattform für deren

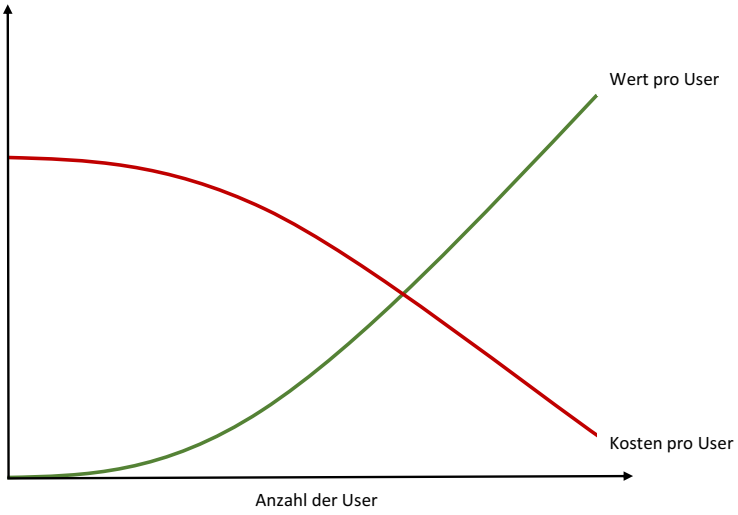

Abbildung 3.2: Vergleich von Wert und Kosten pro User bei einer steigende Anzahl an User einer Plattform (vgl. Loikkanen, 2018)

Anwendungsgebiet einsetzen oder der Anbieter stellt eine Multi-Sided Plattform zur Verfügung, bei der die einzelnen Unternehmen in der Plattform interagieren können. Die zentrale Nutzung einer digitalen Plattform ermöglicht eine Kooperation unterschiedlicher Unternehmen, bei der jedes Unternehmen seine Stärken nutzen kann, und gemeinsam ein Ökosystem gebildet werden kann. Ein Ökosystem ergibt sich, da verschiedene Unternehmen von einander profitieren. Die Bereitstellung einer digitalen Plattform kann auf Basis der genannten Möglichkeiten in den folgenden Ausprägungen unterteilt werden:

- White-Labeling des Produkts durch Anpassung des Designs an das Unternehmen wie die Verwendung des firmeneigenen Logos oder gesamten Corporate Designs in der Applikation. Durch das Branding der Plattform ist für die Anwender des Unternehmens bzw. dem Kunden des Unternehmens nicht zwingend ersichtlich von wem die Software stammt. Die digitale Plattform kann als Firmeneigentum weiterverkauft werden. Der Vorteil dabei ist, dass das Unternehmen nicht selbst die Entwicklung durchführen muss und für die Wartung der Software der Hersteller aufkommen muss. Das Unternehmen kann sich auf deren Kerngeschäft fokussieren und durch die Verwendung einer Plattform deren Geschäft einfach skalieren.

- Integration von bestehenden Systemen bzw. Erweiterung von kundenspezifischen Funktionen – Eine weitere Möglichkeit von White-Label Software ist es, dass ein Standardprodukt erweitert wird. Vordefinierte Prozesse oder Programmschnittstellen werden mit bestehenden Systemen eines Unternehmens integriert, um die bereits vorhandenen Daten des bestehenden Systems zentral in einer digitalen Plattform nutzen zu können. Bietet eine digitale Plattform nicht den

gesamten benötigten Funktionsumfang Out-of-the-box, können Systeme von Drittanbietern in die Plattform integriert werden. Diese Möglichkeit bietet den Unternehmen die Möglichkeit, unterschiedliche Tools für unterschiedliche Prozessschritte zu nutzen, um den Anwendern die Vorgehensweise bei der Erarbeitung von Daten zu erleichtern, ohne zwischen verschiedenen Tools zu wechseln. Ein Beispiel dazu ist eine zentrale Plattform für den Verkauf von Produkten, die zu einem Webshop angebunden ist und Benutzer direkt Produkte in der Plattform verkaufen können. Der Verkauf von Produkten für Großkunden benötigt jedoch einen längeren Verkaufsprozess, dieser wird in einer Software wie Salesforce.com abgebildet, wird ein Angebot zu einem Auftrag wird dieser Auftrag in die zentrale Plattform übernommen. In diesem Beispiel wurden die Aufträge von Salesforce.com in eine Plattform integriert, die für den Verkauf von Produkten genutzt wird.

- Interaktionen von mehreren Unternehmen in einer Plattform – Eine digitale Plattform kann für ein Geschäftsmodell genutzt werden bei dem ein Produkt oder eine Dienstleistung nicht nur von einem Unternehmen angeboten wird. Die Plattform kann genutzt werden indem mehrere Unternehmen, deren eigene Produkte oder Dienstleistungen separat abbilden oder es werden die Stärken der Unternehmen durch Interaktionen und Kooperationen der Unternehmen in einem gemeinsamen Produkt oder Dienstleistung bereitgestellt. Bei einer Plattformnutzung von unterschiedlichen Unternehmen kann es dazu kommen, dass jedes Unternehmen unternehmenseigene Systeme in die digitale Plattform einbinden möchte. Ein Beispiel dazu wäre ein Onlineshop der Produkte von verschiedenen Anbietern verkauft und jedes Unternehmen für den Versand zuständig ist. Somit könnte jedes Unternehmen ein eigenes System für den Versand der Produkte anbinden. Bei einer Bestellung über die digitale Plattform wird die Bestellung automatisch auf die unterschiedlichen Unternehmen aufgeteilt und automatisiert versendet. Die zweite genannte Ausprägung (Integration der Plattform) kann die gleichen Funktionen nutzen, zusätzlich muss beachtet werden, dass Interaktionen zwischen den Unternehmen erfolgen können. Als Beispiel dazu kann das bereits genannte Beispiel mit einem Onlineshop erweitert werden. Das Unternehmen, das eine White-Label Plattform nutzt, bietet Produkte an, die Produkte von verschiedenen Anbietern kombiniert, somit würde sich die Verfügbarkeit des Produktes auf Basis der Verfügbarkeit der Kooperationspartner ergeben. Ähnliche Beispiele finden sich auch in verschiedenen Segmenten in den digitalen Plattformen eingesetzt werden, wie Marketing Plattformen, Consulting Anbietersysteme oder Buchungssysteme.

Die genannten Ausprägungen können nicht nur getrennt voneinander auftreten, sondern können auch kombiniert werden. Je nach Anwendungsgebiet bzw. Art der Software kann sich das Konzept unterscheiden. Beispielsweise eine Verkaufsplattform die von mehreren Unternehmen genutzt wird, kann alle drei Ausprägungen kombinieren. Indem ein Unternehmen, Produkte in deren eigenen Design bereitstellen, die Produktpakete aus eigenen Erzeugnissen und Dienstleistungen eines weiteren

Plattform Nutzers kombiniert und den Bestellprozess mit einer Integration zu einem Customer-Relationship-Management (CRM) System und eine Integration zu einem Lieferanten abschließt (Strader, 2010).

3.1.2 Architektur

Im vorherigen Abschnitt wurden die möglichen Ausprägungen einer digitalen Plattform erläutert. Die Ausprägungen wurden in ökonomische Kriterien untergliedert, dabei wurden drei unterschiedliche Betrachtungsweisen erläutert:

- Wert einer Plattform mit vier unterschiedlichen Modellen von einer One-Sided Plattform ohne Netzeffekte bis hin zu einer Multi-Sided Plattform mit direkten und indirekten Netzeffekten

- Umsatzmodelle, die verwendet werden können, um eine Plattform zu refinanzieren, sind eine indirekte Finanzierung durch eine zahlende Kundengruppe und eine Kundengruppe, denen die Plattform frei von Gebühren zur Verfügung gestellt wird oder eine direkte Finanzierung pro User auf der Plattform.

- White-Label Ausprägungen, die verschiedene Anpassungen der White-Label Plattform ermöglichen, reichen von einer Plattform, bei der das Design an das Unternehmen angepasst werden kann bis hin zu Integration von bestehenden Systemen bzw. Erweiterung von kundenspezifischen Funktionen und Umsetzung von Interaktionsmöglichkeiten von mehreren Unternehmen innerhalb der Plattform

In diesem Abschnitt werden die genannten Ausprägungen technisch betrachtet und Architekturkonzepte von den Ausprägungen einer digitalen Plattform für die Betreibung als Software-as-a-Service erarbeitet. Gemeinsamkeiten der erläuterten Ausprägungen, auf Basis von ökonomischen bzw. konzeptionellen Kriterien, werden gegebenenfalls in einer Architektur kombiniert. Die Architektur bietet die Grundlage für die nachfolgende Analyse von Anbietungsmöglichkeiten mit Hilfe von Clouddiensten.

Die Bereitstellung einer Softwareanwendung als Software-as-a-Service verbindet ein Geschäftsmodell mit einer Softwareanwendung, die es den Kunden ermöglicht eine Software zu nutzen, ohne eine eigene Hardware anschaffen zu müssen. Es wird nur die Benutzung der Applikation bezahlt bzw. es wird anstatt einer Kopie der Anwendung, eine Lizenz zur Benutzung der Anwendung erworben. Diese Art eine Software zu betreiben bringt einige Herausforderungen mit sich und diese müssen in der Architektur der Software berücksichtigt werden. Aus diesem Grund wird zu Beginn erläutert welche technischen Konzepte für die Bereitstellung von SaaS Anwendungen genutzt werden und welche Herausforderungen beachtet werden sollen.

3.1.2.1 Multi-Tier

Die Herausforderung bei einer SaaS Lösung ist, dass bei der Einführung der Plattform schwer abschätzbar ist, wie viele Benutzter auf die Plattform zugreifen werden. Aus diesem Grunde sollte die Architektur bereits eine einfache Skalierung der Anwendung ermöglichen. Ein weiterer Faktor bei der sich die Architektur für eine Plattform mit einer SaaS Bereitstellung von herkömmlichen Softwareanwendungen unterscheidet ist, dass die Software für mehrere unterschiedliche Unternehmen bereitgestellt wird und somit unterschiedliche Anforderungen und Ausprägungen bei der Nutzung der Plattform entstehen können. Die unterschiedlichen Nutzungsbereiche von den Unternehmen kann dazu führen, dass verschiedene Module oder Funktionen einer digitalen Plattform zu verschiedenen Zeiten die höchste Last am System verursachen. Um dieses Problem in die Architektur aufzunehmen, wird eine Applikation in mehrere Schichten (Tiers) und Komponenten aufgeteilt, diese Aufteilung wird auch verteile Architektur bzw. Applikationen genannt. Die Aufteilung erfolgt in drei Schichten, Client oder Präsentations-Tier, Business/Applikations-Tier und Data-Tier. Jede Schicht besteht aus mehreren Komponenten, die unterschiedliche Funktionen oder Module der Plattform abbilden. Die Präsentationsschicht beinhaltet die Benutzeroberfläche und stellt somit das Frontend des Systems zur Verfügung. Das Userinterface ist meist eine Weboberfläche oder ein Mobiles-App. Die Kommunikation und der Datenaustausch vom Userinterface erfolgt über API-Calls zum Applikations-Tier. Die Applikationsschicht beinhaltet, wie bereits die alternative Bezeichnung aussagt, die Businesslogik. Wie bereits zuvor erwähnt wurde, erfolgt auch in dieser Schicht eine weitere Aufteilung in verschiedene Komponenten bzw. Services, um eine Architektur zur erschaffen die Feingranular skaliert werden kann. Die Applikationsschicht verarbeitet nur die Daten, die vom Userinterface übergeben oder von der Datenschicht geladen werden. Eine Komponente der Applikationsschicht übergibt die verarbeiten Daten in den Data-Tier in dem die Daten gespeichert werden. Ansonsten kann eine Komponente in der Applikationsschicht die Daten nur an eine weitere Komponente weitergeben in der die Daten in derselben Weise verarbeitet werden. Die Datenschicht beinhaltet somit Datenbanken oder andere Datastores. Wie bereits erwähnt wurde, kann jede Komponente einer Schicht unterschiedlich skaliert werden, da jede Komponente ein unabhängiges Service darstellt und auf unterschiedlichen Systemen laufen kann. Jedes Service kann unterschiedlich viele Instanzen haben, die durch die Verwendung eines Load-Balancer verteilt werden. Die Abbildung 3.3 zeigt eine exemplarische Aufteilung einer Applikation in die drei Schichten (Fowler, 2003).

3.1.2.2 Mandantenfähigkeit

Ein weiterer Faktor der für die Entwicklung einer Software-as-a-Service Software relevant ist, ist die Entscheidung wie die Anwendung für unterschiedliche Unternehmen bereitgestellt wird. Die Möglichkeiten eine Softwareanwendung bereitzustellen werden jedoch von mehreren Einflussfaktoren und Plattformcharakteren mit-

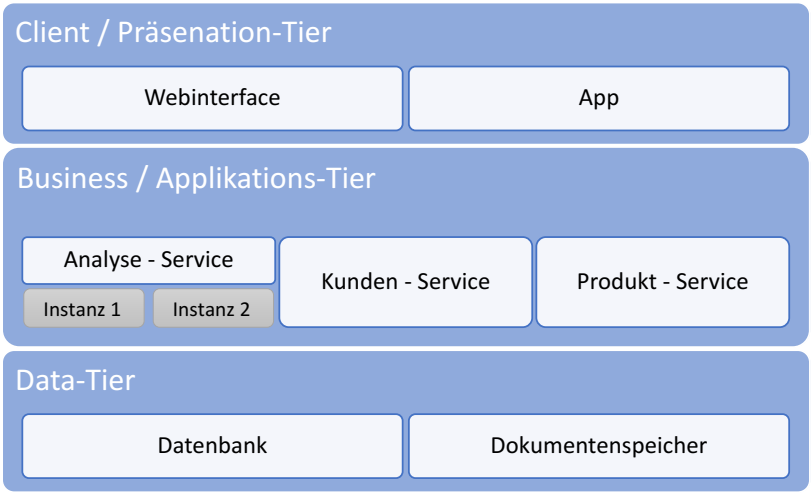

Abbildung 3.3: Multi-Tier Architektur (in Anlehnung an Aknin, 2012)

bestimmt. Einerseits kann es regulatorische Einflussfaktoren geben, die die Auswahl einer Lösung bereits einschränkt. Zum Beispiel regulatorische Anforderungen, dass Daten von unterschiedlichen Unternehmen getrennt abgelegt werden müssen. Dies würde bereits bedeuten, dass unterschiedliche Instanzen für die Unternehmen benötigt werden. Anderseits ist es relevant welche Eigenschaften die Plattform abbildet, wie die Skalierbarkeit, welche Eigenschaften konfiguriert oder parametrisiert werden können und die Integrationsmöglichkeiten, die einem Unternehmen zur Verfügung stehen. Ein Hauptfaktor für die Bereitstellung ist, ob die Software mandantenfähig ist, auch Multi-Tenancy genannt. Die Mandantenfähigkeit bedeutet, dass eine Instanz einer Plattform in mehrere Tenants bzw. Mandanten aufgeteilt wird. Ein Mandant repräsentiert eine Gruppierung von mehreren Benutzern in einer Organisationseinheit wie ein Unternehmen. Es wird die Nutzung der Plattform auf ein Unternehmen isoliert, obwohl die Plattform von mehreren Unternehmen genutzt wird und der Zugriff nur auf den eigenen Bereich erlaubt ist. Je nach Ausprägung der Plattform kann sich jeder Mandant von einem anderen unterscheiden und Anpassungen durchgeführt werden, wie das Design anpassen oder die Plattform mit einem bestehenden System zu integrieren. Multi-Tenancy ermöglicht es mehrere unterschiedliche Ausprägungen einer Applikation in eine Plattform zu kombinieren, anstatt wie bei einer herkömmlichen Anwendung für jede Ausprägung bzw. Unternehmen eine eigene Instanz zu warten. Dieses Konzept kann nicht nur für die Trennung von Unternehmen genutzt werden, die Aufteilung kann auch hierarchisch und feingranular erfolgen, wie beispielsweise eine Aufteilung eines Unternehmens in einzelne Abteilungen oder Partner. Nachdem nur eine Instanz der Plattform unabhängig von der Anzahl an Mandanten laufen muss, ist es in kurzer Zeit möglich weiter Kunden in die Plattform aufzunehmen, ohne dabei die Kosten für die Infrastruktur erheblich zu steigern. Die Entscheidung, dass eine mandantenfähige Anwendung für ein Geschäftsmodell genutzt werden soll, sollte bei ersten Architekturentwürfen inkludiert werden, da dieses Konzept in allen drei Ebenen der Architektur berücksichtig werden

muss. Wie bereits zuvor erwähnt wurde ist es wichtig zu beachten wie die Isolierung der Mandanten erfolgen soll und wie eine Datensicherheit für die gesamte Plattform ermöglicht werden kann. Die Unterteilung muss es ermöglichen, dass keine Daten der Standardsoftware unter den Mandanten ausgetauscht werden können bzw. zugegriffen werden kann, aber auch Konfigurationen und Erweiterungen für einen spezifischen Mandanten erfolgen können. Die Anpassungen können Erweiterungen im Userinterface sein, wie beispielsweise Daten von einem externen System anzuzeigen oder Designanpassungen, die mit der Out-of-the-Box Konfiguration nicht ermöglicht werden. Die Anpassungen können nicht nur im Präsentation-Tier erfolgen, sondern können auch Auswirkungen auf dem Applikation- und Data-Tier haben. Es sollten in allen drei Ebenen Integrationsmöglichkeiten erschafft werden, damit eine gemeinsame Codebasis für die Plattform genutzt werden kann und die Erweiterungen kundenspezifisch sind. Erweiterungen können im Applikation-Layer Anpassungen von Prozessen sein bzw. Änderungen der Business Rules (Prithviraj, 2015).

3.1.2.3 SaaS Datenmodell

Das Datenmodell bzw. das Modell wie Daten für einen Tenant in einer mandantenfähigen Softwareanwendung abgelegt werden, ist für die gesamte Architektur der Software entscheidend. Das Datenmodell bietet die Grundlage für den Aufbau des Applikation-Tiers. Das Design der Datenablage hat keinen direkten Einfluss auf die Funktionen der Applikation, jedoch Auswirkungen auf die Skalierbarkeit, Isolierung der Daten, Performance der Applikation, Kosten für die Betreibung von mehreren Tenants oder Erweiterbarkeit und Aktualisierbarkeit einer Applikation. Damit das Datenmodell die Mandantenfähigkeit implementiert hat, gibt es die Möglichkeit, für jeden Tenant eine eigene Datenbank-Instanz zu verwenden und dasselbe Konzept beim Applikation-Layer zu verwenden. Es wird für jeden neuen Tenant eine eigene Instanz der Software betrieben. Diese Entscheidung kann getroffen werden, wenn eine SaaS Software entwickelt wird, die ausschließlich für Unternehmen bereitgestellt wird, die regulatorisch gezwungen sind Daten physisch separiert von anderen Unternehmen abzulegen. Eine weitere Option eine Applikation für mehrere Tenants zu betreiben ist, dass die einzelnen Datenbanken und Datastores für jeden Tenant getrennt betrieben werden. Die Applikation verwaltet dabei die Verbindung zu einer separaten Datenbank für jeden Tenant. Zusätzlich kann eine Datenbank, für die den gemeinsamen Kontext der Applikation verwendet werden, um Standardwerte für die Applikation ablegen zu können und eine Übersicht der vorhandenen Tenants zu verwalten. Dieses Modell ermöglicht es, dass nur eine Instanz der Applikation benötigt wird und die Daten der Tenants getrennt voneinander abgelegt werden. Dies ermöglicht es, je nach Auslastung eines Tenants die Datenbanken unterschiedlich zu optimieren. Ein weiterer Vorteil dieser Lösung ist, dass eine Tenant-spezifische Anpassung des Datenmodells unabhängig von anderen Tenants erfolgen kann. Die dritte Variante verwendet eine Applikation und eine Datenbank, die für alle Tenants im System genutzt wird. Bei dieser Variante muss in der Applikation und in der Daten-

bank der Tenant verarbeitet werden. Es gibt somit nur eine Verbindung zwischen der Applikation und der Datenbank. Die Auftrennung der Daten für die einzelnen Tenants erfolgt mit einem Tenant Identifier für jeden Dateneintrag wie zum Beispiel eine Spalte mit einem Tenant Key. Die Applikation muss die logische Unterteilung der Tenant implementieren, um die Daten voneinander zu separieren. Der Vorteil dieser Lösung ist es das nur eine Instanz der Datenbank und Applikation benötigt wird. Jedoch bedeutet es auch, dass es bei unterschiedlich großen Datenmengen der Tenants zu einer Herausforderung für eine Performanceoptimierung einzelner Tenants führen kann. Wie bereits bei der ersten Variante erwähnt, müssten die Anforderungen der Zielgruppe analysiert werden, da die Daten von unterschiedlichen Unternehmen bzw. Tenants nur logisch durch die Applikation getrennt werden. In der Abbildung 3.4 sind die drei unterschiedlichen Modelle des Data-Tiers bei einer mandantenfähigen Applikation grafisch dargestellt. Es wird bei SaaS Anwendungen nicht immer zwingend eine Reinform der Modelle verwendet, sondern auch Kombinationen der Varianten. Es könnte für Tenant mit hoher Performanceanforderungen und Datenisolation eine eigene Datenbank verwendet werden und die restlichen Tenants verwenden eine gemeinsame Datenbank (Kavis, 2014).

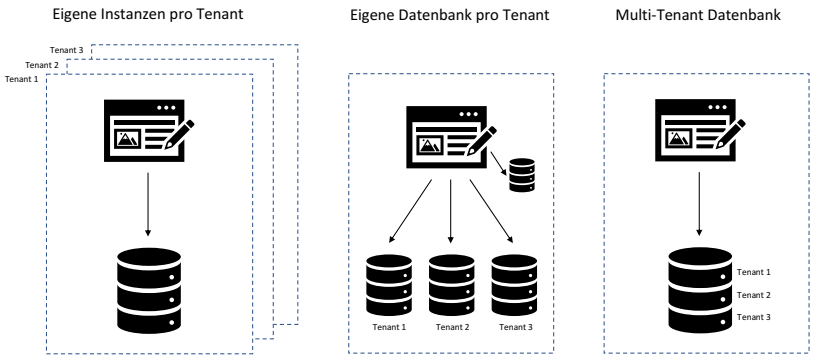

Abbildung 3.4: Data-Tier Modelle bei einer mandantenfähigen Applikation (in Anlehnung an Microsoft, 2018)

3.1.2.4 SaaS Reifegradmodell

Das Datenmodell einer mandantenfähigen Applikation, wie im vorherigen Abschnitt erläutert, hat durch die enge Koppelung der beiden Ebenen Abhängigkeiten zueinander. Die Applikationsschicht muss je nach Datenmodell keine Implementierung der Tenant-Struktur beinhalten bis hin zu einer Implementierung, bei der die Aufteilung der verschiedenen Tenants in der Applikation erfolgt. Eine Tenant Aufteilung im Applikation-Tier ist nicht notwendig, wenn für jeden Tenant eine eigene Instanz der Datenbank und der Applikation betrieben wird bzw. die Applikation muss die Aufteilung der Tenants handhaben, wenn nur eine Datenbank für alle Tenants verwendet wird. Microsoft hat die Möglichkeiten eine Applikation als Software-as-a-Service Lösung bereitzustellen in einem Reifegradmodell dargestellt. Das SaaS Reifegradmodell

teilt die Möglichkeiten in vier Levels bzw. Reifegrade auf. Die unterschiedlichen Levels bringen einem SaaS Anbieter gewisse Vor- bzw. Nachteile bei der Bereitstellung und Betreibung einer Anwendung. Die Abbildung 3.5 zeigt die unterschiedlichen Möglichkeiten. Level eins und zwei bieten eine Applikation für mehrere Tenants an, wobei die Applikation als Single Tenant Applikation betrieben wird und die unterschiedlichen Tenants mit einzelnen Instanzen betrieben werden.

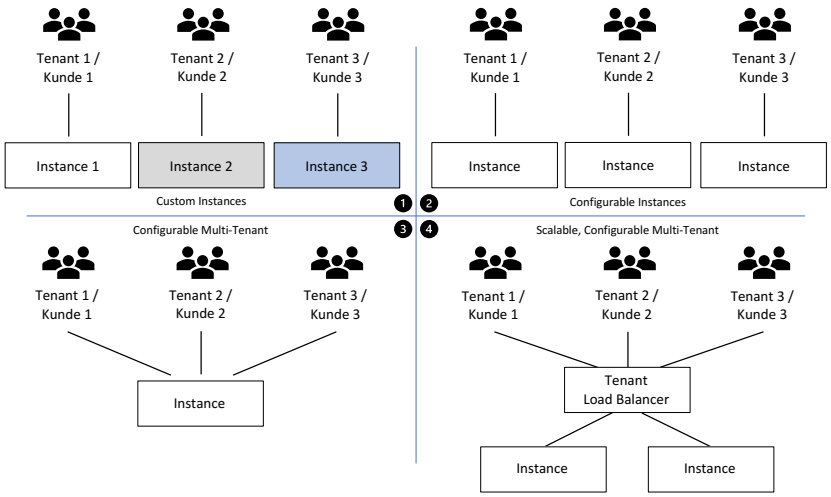

Abbildung 3.5: SaaS Maturity Model (in Anlehnung an Chong, 2006)

Level 1: Custom Instanz

Die einfachste Variante, Level 1, ist eine Architektur, die einer individuellen Softwarelösung ähnelt. Für jeden Kunden / Tenant wird eine angepasste Version der Standardsoftware betrieben. Der Source-Code der Anwendung wird für kundenspezifische Anforderungen individuell angepasst. Die angepasste Anwendung wird unabhängig von anderen Instanzen betrieben, aus diesem Grund muss bei der Implementierung die Mandantenfähigkeit nicht beachtet werden. Es wird die Applikation sowie der Data-Tier als eigene Instanz betrieben. Dieses Modell kann Softwareanbieter den Einstieg in ein SaaS basierendes Geschäftsmodells erleichtern, wenn bereits eine Software vorhanden ist und nur an wenigen Kunden als Software-as-a-Service angeboten werden soll. Eine Neuentwicklung einer Software sollte jedoch bereits bei der Konzeption beachten, dass die Anwendung als SaaS angeboten werden soll. Der Grund dafür ist, dass ein individueller Codestand den Support und den laufenden Betrieb einer Anwendung erschwert, da sich Bugs bei jedem Kunden unterschiedlich auswirken bzw. nur vereinzelnd auftreten können. Außerdem bedeutet es für die Weiterentwicklung der Anwendung, dass die Änderung möglicherweise für jeden Kunden erneut adaptiert werden muss. Wie bereits bei vorherigen Abschnitten erläutert, kann es eine regulatorische Anforderung sein, dass die Daten eines Kunden physikalisch voneinander getrennt sein müssen. Diese Anforderungen bedeuten aber nicht zwingend, dass eine Implementierung erfolgen muss. Es können die

nachfolgenden Reifegrade auch als On-Premises Lösung, also beim Kunden installiert werden, oder die Anwendung wird getrennt von anderen Kunden in der Cloud als dedizierte Instanz bereitgestellt (Chong & Carraro, 2006).

Level 2: Konfigurierbare Instanz

Eine Weiterentwicklung der Single-Tenant Lösung ist der Reifegrad 2. Bei diesem Reifegrad wird jeder Kunde unabhängig von den anderen betrieben, jedoch ist die Applikation auf der gleichen Codebasis. Die Applikation ermöglicht eine individuelle Konfiguration und Parametrisierung des Systems, um kundenspezifische Anforderungen zu erfüllen. Dieses Vorgehen erleichtert die Verwaltung der Software, da nicht unterschiedliche Ausprägungen der Implementierung vorhanden sind. Anpassungen und Erweiterungen der Applikationen müssen somit nur einmal erfolgen. Der Mehraufwand bei diesem Modell ist, dass eine Vielzahl von Instanzen derselben Applikation und Datenbanken verwaltet und upgedatet werden müssen. Die Umstellung einer herkömmlichen Anwendung zu diesem SaaS Modell könnte bereits größeren Aufwand im Vergleich zum Reifegrad 1 bedeuten, da die Parametrisierung und Konfigurationsmöglichkeit in der Softwarearchitektur mit eingebracht werden müssen. Die ersten beiden Levels behandeln die Mandantenfähigkeit nicht auf Applikationsebene, dies beutet für einen Anbieter von Software-as-a-Service Anwendungen, dass die Zuordnung der unterschiedlichen Kunden auf einer höheren Ebene erfolgen muss, wie beispielsweise dass unterschiedliche Zugangspunkte bereitgestellt werden. Außerdem muss beachtet werden, dass die benötigten Ressourcen für die Betreibung der Anwendung für jeden Kunden steigen, was bei Multi-Tenant Systemen nicht zwingend der Fall sein muss (Chong & Carraro, 2006).

Level 3: Konfigurierbares Multi-Tenant System

Level 3 des SaaS Reifegradmodells implementiert die Mandantenfähigkeit im Applikation-Tier. Die Applikation muss somit für verschiedene Tenants verwendet werden können und die Daten für jeden Tenant isoliert werden. Die Datenverarbeitung kann mit den beiden beschriebenen Data-Tier Ausprägungen erfolgen, indem eine eigene Datenbank pro Tenant verwendet wird oder eine Multi-Tenant Datenbank verwendet wird. Die Datenverarbeitung muss somit unterschiedliche Datenverbindungen verwenden oder die Daten logisch auf den Tenants trennen bzw. filtern. Die Herausforderung eines Multi-Tenant Systems liegt jedoch nicht nur in der Auftrennung der Daten, es sollte auch ermöglicht werden, dass die Funktionen pro Tenant angepasst oder erweitert werden können. Einerseits erfolgt dies über die Bereitstellung von Konfigurationen und Parametern pro Tenant, wobei die Anpassungen nicht nur bedeuten, dass sich das Standardprodukt für jeden Tenant unterschiedlich verhält, sondern auch ermöglicht wird, dass vordefinierte Prozesse oder Funktionen mit externen Applikationen oder Funktionen kombiniert werden. Die Möglichkeit eine Plattform für

jeden Kunden anzupassen wird auch in den unterschiedlichen White-Label Möglichkeiten widergespiegelt. Somit sollte diese Funktionalität bei der Erarbeitung des Geschäftsmodells und des Anwendungskonzept genau betrachtet werden, da es nicht nur Auswirkung auf die Softwarearchitektur hat sondern auch erheblichen Mehraufwand beim Support und der Verwaltung jedes Tenants bedeuten kann. Wie bereits im Datenmodell für Multi-Tenant Systeme erläutert wurde, kann jeder Kunde unterschiedliche Ausprägungen auf ein Daten-Objekt haben, dies muss auch in der Applikation unterstützt werden in Kombination mit der bereits genannten Customization eines Mandanten. Aus diesen Gründen sollte die Applikation vordefinierte Möglichkeiten bieten, die für einen Mandanten angepasst werden können oder durch bestehende Kundensysteme ersetzt werden können. Eine Möglichkeit dafür ist, dass die Applikation Metadaten des Mandanten verwaltet, die für den Aufbau von Prozessen die Applikation dynamisch für den Tenant adaptiert. Eine Anpassung oder Erweiterung des Systems kann dazu führen, dass das Userinterface und somit Änderungen im Präsentation-Tier pro Tenant erfolgen können. Dies führt zu einer Abhängigkeit zwischen allen drei Tiers, die vom Applikation-Tier gesteuert werden sollten. Eine genauere Betrachtung erfolgt im Kapital 4. Eine digitale Plattform mit einer Tenant-spezifischen Customization könnte beispielsweise ein SaaS Hotelbuchungssystem sein. Jedes Hotel könnte unterschiedliche Eigenschaften für ein Hotelzimmer benötigen, das Auswirkungen auf das Out-of-the-Box Datenmodel hat. Es sollte einem Hotel ermöglicht werden, dass ein Tenant individuell die vorhandenen Eigenschaften erweitern kann. Das Buchungssystem kann eine Integration ermöglichen, die bei einer erfolgreichen Buchung ausgeführt wird. Dies würde einem Hotel ermöglichen ein bestehendes Türschließungssystem mit der SaaS Buchungsplattform zu kombinieren. Damit können die Schlüssel für eine Buchung vom bestehenden System automatisiert vorbereitet werden und das Hotel muss sich keine Individualsoftware Lösung entwickeln lassen. Der Reifegrad 3 ermöglicht somit einem SaaS Anbieter mit nur einer Instanz der Anwendung, viele Kunden zu betreiben, ohne dass es signifikante Auswirkungen auf die bereits verwendeten Ressourcen hat. Diese Möglichkeit kann bei vielen kleinen als auch großen Kunden einen erheblichen Kostenvorteil bringen, da die Mindestanforderungen der Applikation nicht für jeden Kunden erneut benötigt werden (Chong & Carraro, 2006).

Level 4: Skalierbares und konfigurierbares Multi-Tenant System

Die vierte Ausprägung des SaaS Reifegradmodells nutzt die Stärken des dritten Levels und ergänzt das Konzept mit einer einfachen Skalierung der Ressourcen. Es wird versucht die Last des gesamten Systems auf eine kostengünstige Variante aufzuteilen. Die Anzahl der Tenants und die aktuelle Auslastung auf das System erhöht bzw. verringert die Ressourcen in einem dynamischen Pool automatisiert. Dieses Modell ermöglicht einem Anbieter einer Software-as-a-Service-Anwendung die Kosten gering zu halten, unabhängig davon ob nur ein Kunde oder hunderte Kunden am System sind. Die benötigten Ressourcen werden auf die aktiven User am System angepasst,

anstatt die Ressourcen an die gesamten User von allen Tenants auszurichten (Chong & Carraro, 2006).

3.1.2.5 Userinterface für Mandanten

Die Präsentationsschicht ist die Schnittstelle zum Anwender der Applikation bzw. stellt die Funktionen für den Kunden bereit. Aus diesem Grund müssen die Tenant-spezifischen Funktionen auch im Präsentation-Tier dementsprechend dynamisch erweiterbar sein. Sei es in einem Webinterface, einer Desktop-Applikation, mobilen App oder nur als API für die Implementierung eines unternehmenseigenen Userinterface bzw. eine Integration in eine bestehende Applikation des Kunden. Je nach Ausprägung der Präsentationsschicht sollte es eine Möglichkeit geben das Look-and-Feel des Userinterfaces anzupassen, das Layout zu restrukturieren, weitere UI Komponenten hinzuzufügen oder die Seiten- bzw. Funktionsfolge zu adaptieren. Wie bereits beim Applikation-Tier erwähnt wurde, sollte auch die Repräsentierung des Systems konfigurierbar und parametrierbar sein, um die Funktionen auf einfache Weise für einen Kunden zu adaptieren. Aber auch auf dieser Ebene gibt es unterschiedliche Vorgehensweisen um ein System als Software-as-a-Service Lösung mehreren Kunden anzubieten. Sei es ein eigener Webserver mit individuell angepassten Frontendcode bzw. je nach Bereitstellungsmethode (Webinterface, App, API), eine Lösung, dass für jeden Kunden eine eigene Instanz betrieben wird. Andererseits gibt es Möglichkeiten, dass auf Basis von Metadaten, die als Konfiguration von der Applikationsschicht bereitgestellt werden, die Bereitstellung bzw. der Aufbau der Interfaces dynamisch pro Mandanten aufgebaut wird (Pradeep, V. & Palaniswami, 2010).

3.2 Clouddienste

Die Nachfrage und der Umsatz von Clouddiensten stieg über die letzten Jahren erheblich, laut einer Studie von Gartner Inc. im Jahr 2020 stieg der Umsatz von 2018 bis 2019 um 37,3%. Der Umsatz für Infrastruktur-as-a-Service Dienste der weltweiten Public Cloudanbieter erreichte 2019 einen Umsatz von 44,5 Mrd. $. Dieses Wachstum zeigt die Nachfrage an Clouddiensten, um digitale Geschäftsmodelle zu unterstützen. Viele digitale Geschäftsmodelle benötigen eine Infrastruktur für deren Systeme, die hochverfügbar, einfach skalierbar und dynamisch erweiterbar sind. Die Abbildung 3.6 zeigt eine Übersicht der fünf größten Cloudanbieter, die 80% des gesamten Marktes ausmachen, im Vergleich von 2018 zu 2019. Der größte Anbieter mit 45% Marktanteil ist Amazon mit Amazon Web Services (AWS), gefolgt von Microsoft Azure und Alibaba Cloud. In dieser Analyse werden die drei führenden Cloudanbieter analysiert und verglichen.

Unternehmen	*Umsatz 2019 (Mrd. $)*	*Marktanteil 2019 (%)*	*Umsatz 2018 (Mrd. $)*	*Marktanteil 2018 (%)*	*2018-2019 Growth (%)*
Amazon	19,990.4	45.0	15,495.0	47.9	29.0
Microsoft	7,949.6	17.9	5,037.8	15.6	57.8
Alibaba	4,060.0	9.1	2,499.3	7.7	62.4
Google	2,365.5	5.3	1,313.8	4.1	80.1
Tencent	1,232.9	2.8	611.8	1.9	101.5
Others	8,858	19.9	7,425	22.9	19.3
Total	**44,456.6**	**100.0**	**32,382.2**	**100.0**	**37.3**

Abbildung 3.6: IaaS Public Cloud Services Marktanteile (vgl. Gartner, 2020)

3.2.1 Übersicht

Die meisten Cloud Provider bieten vergleichbare Produkte im Bereich Infrastructure-as-a-Service und als Plattform-as-a-Service an, wie beispielsweise Datenbanken, Applikationsserver, Dienste für künstliche Intelligenz oder Dienste für Maschine Learning. Die Dienste werden in verschiedenen Regionen und Standorten zur Verfügung gestellt. Die Auswahl eines Cloudanbieters wird meist aufgrund verschiedener Kriterien getroffen, wie beispielsweise:

- Sicherheit und Compliance: Ein Unternehmen wird meist zu Beginn überprüfen, ob die Sicherheitsvorkehrungen des Cloudanbieters mit den unternehmensinternen Richtlinien übereinstimmen. Die Sicherheit der Daten ist ein kritischer Faktor bei digitalen Geschäftsmodellen, deshalb sollte dieser mit hoher Priorität betrachtet werden. Dies sollte auch eines der wichtigsten Punkte eines Anbieters sein, dass Dienste mit hohen Sicherheitsmaßnahmen bereitgestellt werden. Außerdem ist es entscheidend, dass die Sicherheitsmaßnamen und die Richtlinien mit den Anforderungen des Unternehmens übereinstimmen, wie beispielsweise das die Verwendung der Dienste die Anforderungen der Datenschutzgrundverordnung (DSGVO) erfüllen. Je nach Industriestandard muss dabei auf verschiedene Compliance-Richtlinien geachtet werden. Dabei sollte bedacht werden, ob die Sicherheitsfunktionen Out-of-the-Box bei den Diensten inkludiert sind oder eine Erweiterung der jeweiligen Dienste benötigt wird.

- Skalierbarkeit: Eine Applikation wird meist in einer Cloud betrieben, um uneingeschränkt die Applikation skalieren zu können, ohne eine gesamte Serverinfrastruktur zu warten bzw. erweitern zu müssen. Aus diesem Grund sollte bei der Auswahl beachtet werden, welche Skalierungsmöglichkeiten pro Dienst geboten werden bzw. wie weit ein Dienst skaliert werden kann. Ein weiterer Faktor der Skalierung ist, in welchen geografischen Gebieten die Dienste be-

trieben werden, um ein System in verschiedenen Regionen anbieten zu können, ohne dass sich die Daten auf einem anderen Kontinent befinden. Damit ein System laufend erweitert werden kann und aktuelle Technologien eingesetzt werden können, sollte darauf geachtet werden, dass die Dienste des Cloudanbieters laufend erweitert bzw. aktualisiert werden.

- Architektur: Die Architektur des Systems, das in der Cloud betrieben werden soll, aber auch der Aufbau der Clouddienste kann einen Einfluss auf die Auswahl des Cloudanbieters haben. Dienste die unterschiedliche Anforderungen an eine Infrastruktur haben, können sich bei den verschiedenen Anbietern unterschiedlich verhalten, wie beispielsweise Datenbanken oder Maschine Learning Dienste, die bei der Ausführung von Operationen je nach Anwendungsfall variieren können. Bei einer großen Anzahl an verwendeten Clouddiensten ist es relevant, dass die Dienste übersichtlich verwaltet werden können. Je nach Umfang des Systems kann es somit entscheidend sein, dass die Verwaltung oder Automatisierung von der Orchestrierung der Dienste auf einfacher Weise gegeben ist. Der Vorteil von Clouddiensten sollte sein, dass in kurzer Zeit und ohne Wartung einer Infrastruktur, ein System bereitgestellt werden kann.

- Kosten: Es sollte nicht der entscheidende Faktor bei der Auswahl des Cloudanbieters sein, jedoch wird es trotzdem eine große Rolle bei der Auswahl spielen. Die Kosten bestehen nicht nur aus den laufenden Kosten für die verwendeten Dienste, sondern auch welche zusätzlichen Kosten für die Verwaltung oder für mögliches zusätzliches Personal oder Schulungen, je nach Anbieter, anfallen. Die Kosten können durch unterschiedliche Nutzung und Auslastung des Systems stark variieren, dabei sollte laufend geachtet werden welche Art von Service genutzt wird. Eine dedizierte Ressource für einen Dienst hat einen höheren Preis im Vergleich zu einer Ressource aus einem Pool. Wird ein Funktionsbereich nur für einen kurzen Zeitraum im Monat benötigt, jedoch in dieser Phase mit einer sehr hohen Auslastung, könnte dies zu einer anderen Auswahl eines Clouddienstes führen, als wäre die Auslastung dauerhaft. Die Kosten der Dienste sollten laufend überwacht werden, um gegebenenfalls die Architektur ein wenig zu adaptieren.

Die Auflistung möglicher Kriterien die bei der Auswahl eines Cloudanbieters herangezogen werden kann, zeigt bereits die Vielzahl an möglichen Ausprägungen, die sich durch die Verwendung von Clouddiensten ergeben. Wie bereits erwähnt, bieten die meisten Cloudanbieter ähnliche Dienste an. Jedoch sollte darauf geachtet werden, wenn mehrere Provider eingesetzt werden, dass die Architektur der Anwendung mit den unterschiedlichen Diensten kompatibel ist oder die Clouddienste eine standardisierte Schnittstelle bieten. Damit ist der Wechsel oder eine zusätzliche Instanz bei einem anderen Cloudanbieter nicht mit großen Änderungen der Applikation verbunden. Ein weiterer Vorteil bei der Verwendung von Clouddiensten ist, dass sich die Projektkosten verändern im Vergleich zu einer eigenen Infrastruktur. Die entstehenden Kosten bei Clouddiensten werden als Betriebskosten (OpEx) aufgewendet, da

die Kosten variabel auf Basis der Kundenanforderungen anfallen. Bei einer eigenen Infrastruktur fallen Investitionskosten (CapEx) für die Infrastruktur an. Einem neuen digitalen Geschäftsmodell wird es somit erleichtert, dass zu Beginn keine große Investition für die Bereitstellung einer Anwendung benötigt wird. Es werden nur die verwendeten Ressourcen verrechnet (Mahmood & Hill, 2011).

3.2.1.1 Clouddienste

In diesem Abschnitt wird eine Übersicht von verschiedenen Clouddienste in den unterschiedlichen Anwendungsbereichen erläutert. Die unterschiedlichen Dienste werden mit exemplarischen Produkten der drei Cloudanbietern mit dem größten Marktanteil ergänzt – Amazon Web Services (AWS), Microsoft Azure und Alibaba Cloud. Die Übersicht wird in die folgenden Anwendungsbereiche aufgeteilt:

- Computing: Dienste, die für die Bereitstellung von einzelnen Applikationen genutzt werden können oder um Daten zu verarbeiten.

- Netzwerkfunktionen: Dienste für die Verwaltung eines Netzwerks in der Cloud bzw. um mehrere Cloud Netzwerke oder private Netzwerke zu vereinen und abzusichern.

- Datenablage: Dienste für die Ablage von unterschiedlichen Datenformaten bzw. unterschiedlichen Datenanbindungen.

- Applikationsdienste: Dienste, die innerhalb einer Applikation benötigt werden, damit Frameworks oder Designmuster nicht selbst implementiert werden müssen.

- Sicherheitsdienste: Dienste, die es ermöglichen sensible Daten zu schützen und den Zugriff auf Applikationen einzuschränken.

- System-Integrationsdienste: Dienste, die für die Entwicklung und Verwaltung von Softwaresystemen benötigt werden oder Funktionen, die als Service bereitgestellt werden und in ein bestehendes Softwaresystem integriert werden.

- Marketplace: Dienste von Cloudpartnern, die Applikationen bzw. Lösungen als Service zur Verfügung stellen. Softwarehersteller verwenden Clouddienste, um deren gesamte Lösung in der Cloud zur Verfügung zu stellen.

Computing

Die Datenverarbeitung in der Cloud kann mit virtuellen Maschinen, Containern oder Serverless-Diensten bereitgestellt werden. Die Bereitstellung sollte in jeder Variante eine hohe Zuverlässigkeit und Leistungsfähigkeit haben. Die Produkte lassen sich auf die folgenden Eigenschaften aufteilen. (Dabei wird nur auf die Bereitstellung der Applikation eingegangen und nicht wie beispielsweise die Installation auf einer virtuellen Maschine gelangt, wie ein Build-Prozess für ein Container-Image erfolgt oder eine Serverless-Function deployt werden kann):

- Virtuelle Maschinen (VM): Eine Möglichkeit eine Applikation in der Cloud bereitzustellen ist die Nutzung von virtuellen Maschinen. Diese Möglichkeit wird auch bei On-Premises Lösungen eingesetzt. Der Dienst wird als Infrastructure-as-a-Service bereitgestellt. Eine VM kann mit verschiedenen Linux oder Windows Images erstellt werden. Bei VMs in der Cloud kann die Rechenkapazität flexibel gewählt werden, damit die Entwicklung und die Bereitstellung von Applikationen uneingeschränkt erfolgen kann. Die Konfiguration einer Instanz erfolgt aus einer Auswahl an Prozessor, Speicher, Netzwerk, Betriebssystem und Kaufmodell. Unternehmen, die Applikationen in einem eigenen Rechenzentrum anbieten, bieten virtuellen Maschinen einen einfachen Umstieg in die Cloud. Durch die Eigenschaften einer VM muss beachtet werden, dass das jeweilige Betriebssystem bereits einen Overhead für die Ausführung hat, dies sollte auch beim Sizing der Instanz beachtet werden. Da nur die Infrastruktur bereitgestellt wird, muss zusätzlich die Provisionierung der Instanz gehandhabt werden. Das bedeutet zusätzlichen Aufwand bei der Betreibung eines Systems, um die Instanz nach der Bereitstellung des Cloudanbieters nutzen zu können. Produkte der bereits genannten Cloudanbieter sind: Amazon Elastic Compute Cloud (EC2), Microsoft Azure Virtuelle Computer und Alibaba Cloud Elastic Compute Service (ECS).

- Automatische skalierbare VMs: Das Grundkonzept ist dasselbe wie bei einzelnen virtuellen Maschinen, der Vorteil von automatisch skalierenden VMs ist, dass auf Basis von vordefiniertem Bedingen die Rechenkapazität der VM erhöht oder verringert wird bzw. weitere Instanzen gestartet oder gestoppt werden. Bei einer unregelmäßigen Nutzung einer Applikation kann mit diesem Dienst die Leistung dynamisch gesteuert werden und somit die Kosten optimiert und die Fehlerrate reduziert. Die Aufteilung kann auf dem Zustand einer Instanz oder der Auslastung einer Instanz, über mehrere verfügbaren Zonen verteilt werden, um die Verfügbarkeit der Applikation zu erhöhen. Ein Software-as-a-Service Dienst sollte den Usern ein optimale User Experience geben, unabhängig von der Auslastung des Betreibers, dazu bieten die Cloudprovider, Amazon EC2 Auto Scaling, Microsoft Azure Virtual Maschine Scale Sets oder Alibaba Cloud Auto Scaling.

- App Service: Ein App Service ist ein Dienst, der als Plattform-as-a-Service bereitgestellt wird. Webanwendungen können ohne die Konfiguration und die Verwaltung einer Infrastruktur betrieben werden. Der Dienst deployt, skaliert, adaptiert und überwacht die Applikation automatisiert. Die Webanwendung kann meist in den gängigsten Programmiersprachen wie Java, Python, Node.js, PHP, Go oder NET entwickelt und bereitgestellt werden. Im Vergleich zu einer virtuellen Maschine ist der Vorteil, dass der Dienst die Konfiguration und das Management der Infrastruktur automatisiert übernimmt, sowie das Deployment der Anwendung auf den Server. Dienste sind beispielsweise AWS Elastic Beanstalk, Microsoft Azure App Service oder Alibaba Web App Service.

- Container: Ein Betriebssystem unabhängige Variante bieten Container. Ein Container bietet die Möglichkeit eine Applikation mit allen benötigten Abhängigkeiten, wie eine Laufzeitumgebung, Libraries oder Konfiguration, isoliert voneinander auszuführen. Die Bereitstellung auf unterschiedlichen Systemen erfolgt somit auf dieselbe Weise. Die erstellten Images können auf den unterschiedlichen Anbietern ausgeführt werden. Dazu gibt es in der Cloud, Dienste wie Amazon Elastic Container Service, Microsoft Azure Container Instances oder Alibaba Elastic Container Instance (ECI).

- Container Orchestrierung: Ein Softwaresystem besteht meist aus mehreren Service bzw. müssen Service auf verschiedene Weisen skaliert werden. Damit mehrere Container gemeinsam verwaltet werden können bietet Kubernetes die Möglichkeit Container in einem Cluster zu betreiben. Container können in der Cloud beispielsweise mit Amazon Elastic Kubernetes Service (EKS), Microsoft Azure Kubernetes Service (AKS) oder Alibaba Container Service for Kubernetes (ACK) orchestriert werden.

- Serverless: Die Ausführung von Code für einzelne Funktionen kann mit einem Serverless-Dienst erfolgen. Die Verwaltung eines Servers oder das Deployment einer Applikation wird vom Cloudprovider übernommen. Es wird nur der Code für die Funktionalität bereitgestellt und dieser wird als einzelne Funktion, die einen Input und Output haben kann, definiert. Der Vorteil ist, dass somit kein System laufend zur Verfügung gestellt werden muss, denn der Clouddienst wird bei jedem Aufruf ausgeführt. Die Skalierung muss somit nicht gewartet werden, dies wird vom Cloudanbieter übernommen. Es werden nur Ausführungen verrechnet, somit würden keine Kosten anfallen, wenn das System nicht verwendet wird. Das Konzept von Serverless-Functions sind auch unter den Namen Backend-as-a-Service (BaaS) oder Function-as-a-Service (FaaS) bekannt. AWS Lambda, Microsoft Azure Functions oder Alibaba Function Compute repräsentieren exemplarisch diese Technologie (García & Biggs, 2019).

Netzwerkfunktionen

Die Bereitstellung einer Anwendung aus mehreren Clouddiensten oder auch Services in einem bestehenden Datacenter benötigen Netzwerkkomponenten. Netzwerkkomponenten sind somit essentiell, um ein System bereitstellen zu können und zugleich Komponenten voneinander zu trennen bzw. von der Öffentlichkeit zu schützen. Außerdem werden Netzdienste genutzt, um die Last und den Traffic auf mehrere Instanzen oder Regionen aufzuteilen, um eine Ausfallsicherheit zu gewähren. Die folgenden Dienste zeigen die gängigsten Netzwerkfunktionen in der Cloud:

- Virtuelles privates Cloud-Netzwerk – VPC: Ein virtuelles privates Netzwerk ermöglicht, Clouddienste in eigenen Netzwerken zu isolieren. Komponenten in unterschiedlichen VPC haben keinen Netzwerkzugriff zueinander. Es müssen somit separate Netzwerke für jedes VPC konfiguriert werden. Ein Dienst der Internetzugriff benötigt kann von internen Diensten mit eingeschränktem Zugriff getrennt werden. Ein Dienst der Internetzugriff benötigt, kann somit von internen Diensten mit eingeschränktem Zugriff getrennt werden. Ein virtuelles Netzwerk kann mit weiteren Cloud-Netzwerkdiensten kombiniert werden, wie beispielsweise VPN, um einem Netzwerk den Zugriff auf einem On-Premises Dienst zu gewähren. Produkte hierfür wären Amazon Virtual Private Cloud (Amazon VPC), Microsoft Azure Virtual Network oder Alibaba Virtual Private Cloud (VPC).

- VPN Gateway: Bei der Verwendung von mehreren Anbietern oder bei der Verwendung von einer Cloud und On-Premises Diensten müssen die beiden Netzwerke miteinander verbunden werden, dazu können wie in anderen Umgebungen, VPN-Verbindungen genutzt werden. Die meisten Cloudprovider bieten die Möglichkeit eine Site-to-Site oder Punkt-zu-Site Verbindung für eine sichere Verbindung zwischen Netzwerken aufzubauen mit Diensten wie AWS VPN, Microsoft Azure VPN Gateway oder Alibaba VPN Gateway.

- Load-Balancer: Ein Softwaresystem kann aus mehreren Instanzen, Containern oder Serverless-Functions bestehen, die auch in unterschiedlichen geografischen Bereichen ausgeführt werden. Anfragen sollten, um eine hohe Verfügbarkeit bzw. schnelle Responsezeit garantieren zu können, gleichmäßig verteilt werden. Die Verteilung mit Load-Balancer in der Cloud kann auf unterschiedlichen Ebenen durchgeführt werden. Wie bereits die unterschiedlichen Bereitstellungsarten genannt wurden, kann die Kommunikation zwischen den Services auf unterschiedlichen Protokollen geschehen. Load-Balancer von Cloudprovidern ermöglichen es deshalb in den meisten Fällen, dass der Lastausgleich für unterschiedliche Protokolle wie HTTP, HTTPS, SMTP oder andere TCP/UDP-basierende Protokolle durchgeführt werden kann. Dienste wie Amazon Elastic Load Balancing, Microsoft Azure Load Balancer oder Alibaba Cloud Server Load Balancer (SLB) ermöglichen es in einfachster Weise ein hochverfügbares System bereitzustellen (Klaffenbach et al., 2019).

Datenablage

Die meisten Applikationen müssen Daten in unterschiedlichen Formen ablegen, dazu werden Datenbanken oder andere Datenablagen benötigt. Die Cloud bietet dazu verschiedene Möglichkeiten Daten abzulegen, ohne sich um eine eigene Infrastruktur zu kümmern. Der Vorteil dabei ist, dass die Ressourcen meist ohne große Einschränkungen skaliert und erweitert werden können.

- File Storage: Daten können in der Cloud auf unterschiedlichste Weise abgelegt werden. Die Unterscheidung erfolgt meist danach ob die Daten strukturiert oder unstrukturiert sind. Außerdem wird unterschieden mit welchem Protokoll auf die Datenablage zugegriffen wird, wie beispielsweise HTTPS oder SMB. Unstrukturierte Daten können in einem Objektspeicher abgelegt werden. Objektspeicher legen Daten in einer flachen Struktur ab, jedes Objekt kann mit einer eindeutigen ID abgerufen werden. Der Datenaustausch wird mit einer REST-API durchgeführt. Eine Aufteilung von Objekten kann mit einem Block-Storage erfolgen, dabei werden wie der Name bereits aussagt in mehreren Blöcken aufgeteilt. Beispiele für einen Objektspeicher ist Amazon Simple Storage Service (Amazon S3), Microsoft Azure Blobspeicher oder Alibaba Cloud Object Storage Service (OSS). Strukturierte Daten können in einem File-Storage abgelegt werden, die es ermöglichen Daten in einer herkömmlichen Verzeichnisstruktur abzulegen. Ein File-Storage in der Cloud ermöglicht es Dateien in einem skalierbaren und elastischen Dateisystem abzulegen. Der Zugriff erfolgt über herkömmliche Datenzugriffsprotokolle, der in den verschiedenen Bereitstellungsmethoden ermöglicht wird. Repräsentativ zu unterschiedlichen Cloud-Dateisystemen bietet Amazon Elastic File System (Amazon EFS), Microsoft Azure Files und Alibaba File Storage NAS an.

- Relationale Datenbanken: Die meisten Anwendungen benötigen eine Datenbank, um Daten zu speichern, dazu werden häufig relationale Datenbanken wie Oracle Database, MySQL oder PostgreSQL eingesetzt. Die Verwaltung großer Datenbanken erfordert viel Knowhow von Datenbank-Administratoren (DBA) in einem Unternehmen. Um dies zu erleichtern werden Datenbanken auch als Clouddienste angeboten. Die meisten Datenbank-Clouddienste sind mit den gängigsten Datenbank-Engines kompatibel und können problemlos mit herkömmlichen Datenbanken ausgetauscht werden, wie beispielsweise mit Amazon Relational Database Service (Amazon RDS), Microsoft Azure SQL-Datenbank oder Alibaba Cloud ApsaraDB for PolarDB.

- NoSQL Datenbanken: Applikationen mit einem flexiblen Datenmodell setzen häufig auf NoSQL Datenbanken, um Daten abzulegen. Damit der Datenbank-Dienst skalierbar ist und eine hohe Verfügbarkeit hat bieten Cloudprovider, NoSQL-Datenbanken als Plattform-as-a-Service an. Dienste wie Amazon DynamoDB, Microsoft Azure Cosmos DB oder Alibaba Cloud Tablestore bieten

eine hohe Leistung bei einer hohen Verfügbarkeit mit einer Kompatibilität zu herkömmlichen NoSQL-Datenbanken wie MongoDB oder Apache Cassandra.

- In-Memory Cache: Echtzeitanwendungen oder Anwendungen mit vielen Read-Operation setzen In-Memory Caches wie Redis ein, um die Latenz der Ausführung niedrig zu halten und die Datenverarbeitung zu optimieren. Dasselbe gilt für Anwendungen mit einer hohen Last, bei der eine hohe Anzahl an Benutzer auf ein System zugreifen. Die Verarbeitung kann optimiert werden indem ein Zwischenspeicher (Cache) genutzt wird. Cloudprovider bieten dazu In-Memory-Datenspeicher, die mit den gängigen Zwischenspeichern kompatibel sind, wie beispielsweise Amazon ElastiCache for Redis, Microsoft Azure Cache for Redis oder Alibaba Cloud ApsaraDB for Redis (Laszewski et al., 2018).

Applikationsdienste

Eine Softwarearchitektur mit Verwendung von Clouddiensten ermöglicht es, die Kommunikation von verschiedenen Komponenten wie Microservice, Serverless-Functions oder andere Anwendungen mit bereitgestellten Diensten zu vereinen. Eine Trennung von verschiedenen Functions erleichtert die Anpassung von Funktionen und vereinfacht die Umsetzung. Verschiedene Designpattern müssen nicht erneut in einer Eigenimplementierung umgesetzt und betrieben werden. Clouddienste wie beispielsweise API-Gateways oder Queues können für die Konzeption eines Softwaresystems verwendet werden.

- API-Gateway: Eine Architektur mit verschiedenen Microservices, Serverless-Functions oder andere Funktionsendpunkten können auf unterschiedliche Weisen zugänglich gemacht werden. Eine Möglichkeit ist einen API-Gateway zu verwenden, der die öffentliche Schnittstelle zu einer internen Architektur mit mehreren Services darstellt. Die Schnittstelle kann mit RESTful-APIs oder Websocket-APIs abgebildet werden. Der API-Gateway verwaltet APIs, autorisiert und steuert den Zugriff, sowie Überwachung der Datenübertragungen. Beispiele dazu wären Amazon API Gateway, Microsoft Azure API Management oder Alibaba Cloud API Gateway.

- Workflows: Bei einer feingranularen Aufteilung von APIs wie bei Serverless-Functions müssen die einzelnen Funktionen in einen gesamten Prozess zusammengefasst werden, dazu können verschiedene Workflow-Frameworks bzw. Functions-Orchestratoren verwendet werden. Cloudprovider bieten dazu Dienste die es ermöglichen, die Logik von der Implementierung der einzelnen Functions zu entnehmen und in eigenen Workflows unabhängig zu verwalten, wie beispielsweise mit AWS Step Functions, Microsoft Azure Logic Apps oder Alibaba Cloud Function Compute.

- Messaging: Ergebnisbasierende Funktionen werden asynchron in einem System abgearbeitet. Asynchrone Prozesse können Funktionen sein, die eine Information an mehrere Services weitergeben oder Prozesse werden in einer Queue abgearbeitet. Die Umsetzung von asynchronen Prozessen oder asynchrone Nachrichtenübertragung wird mit Designpattern wie Publish-Subscribe oder Queues umgesetzt. Um eine skalierbare Lösung für asynchrone Nachrichtenübertragung in der Cloud abbilden zu können, können Dienste wie Amazon Simple Notification Service (SNS) oder Microsoft Azure Event Grid für eine Publish / Subscriber Architektur verwendet werden bzw. Amazon Simple Queue Service (SQS), Microsoft Azure Queue Storage oder Alibaba Cloud AlibabaMQ for Apache Kafka als Queue eingesetzt werden.

- Suchfunktion: Die meisten Anwendungen bieten eine Suche für die bereitgestellten Daten. Eine umfangreiche Datenmenge kann eine Suche von bestimmten Daten aufwendig machen und sehr zeitintensiv sein. Um dies zu verbessern werden Search-Engines in eine Applikation integriert die bestimmte Eigenschaften von Daten in einem Index abbilden. Dies ermöglicht umfangreiche Suchanfragen mit einer niedrigen Responsezeit. Ein Search-Engine, wie beispielsweise ElasticSearch, kann selbst bereitgestellt und verwaltet werden. Jedoch erfordert es wie bei einer Datenbank einen erhöhten Aufwand, um die Installation zu optimieren bzw. kann die Installation nur begrenzt erweitert werden. Cloudprovider bieten dazu Search-Engine als Service an wie beispielsweise Amazon Elasticsearch Service, Microsoft Azure Cognitive Search oder Alibaba Cloud Elasticsearch (Laszewski et al., 2018).

Sicherheitsdienste

Ein Softwaresystem in der Cloud zu betreiben bringt einige Herausforderungen für die Datensicherheit mit sich, wobei es ähnliche Herausforderungen bei einer On-Premises Lösung gibt. Der Zugriff auf Daten, Anwendungen oder Teile der Infrastruktur müssen auf unterschiedlichen Ebenen abgesichert und überwacht werden. Clouddienste haben meist bereits eigene Sicherheits-Features inkludiert, wie beispielsweise Firewall-Regeln in einem Load Balancer, zusätzlich bieten Cloudanbieter Ergänzungen für die Sicherheit.

- Datenverschlüsselung: Systeme, die Daten verschlüsseln, entschlüsseln oder signieren, benötigen kryptografische Schlüssel, die sicher abgelegt werden müssen. Cloudprovider bieten einerseits Dienste mit Hardware-Sicherheitsmodulen für die Speicherung der kryptografischen Schlüssel und für die Verschlüsselung von Daten. Schlüssel können in Diensten wie AWS Key Management Service (KMS), Microsoft Azure Key Vault oder Alibaba Cloud Key Management Service (KMS) abgelegt werden.

- Identity Access Management (IAM): Softwaresysteme aus mehreren Services und Daten für unterschiedliche Benutzerrollen verwenden Identity Access Management Systeme, um den Zugriff für bestimmte User einzuschränken. IAM ermöglicht ein zentrales System für die Zugriffskontrolle zu verwalten und die Berechtigungen für alle Services zu steuern. Außerdem wird es ermöglicht mit Single Sign-One einen Benutzer mit einem Login für alle Applikationen im System zu authentifizieren. Clouddienste wären beispielsweise AWS Identity and Access Management (IAM), Microsoft Azure Active Directory oder Alibaba Cloud Resource Access Management (RAM).

- DDoS Protection: Anwendungen, die öffentlich zugänglich sind, können durch Cyberangriffe, wie Distributed Denial of Service (DDoS) Angriffe, betroffen sein. Dies führt dazu, dass die Anwendung aufgrund von einer hohen Anzahl an Anfragen von Angreifern nur mehr beschränkt nutzbar wird und die Last am System oder Latenzzeiten stark erhöht werden. Damit ein System in der Cloud vor Ausfällen und erhebliche Zusatzkosten geschützt wird, bieten Cloudprovider einen Dienst für einen DDoS Schutz an, wie beispielsweise AWS Shield, Microsoft Azure DDoS Protection oder Alibaba Cloud Anti-DDoS.

- Threat Protection: Softwaresysteme werden meist stetig weiterentwickelt. Bei großen Systemen kann es dazu führen, dass Sicherheitslücken auftreten oder ungeplante Aktivitäten am System ausgeführt werden. Cloudprovider bieten dazu intelligente Dienste, die automatisiert mögliche Bedrohungen erkennen und das System kontinuierlich überwachen. Fehler in einem System oder Angriffe auf einem System können zu einem unerwarteten Verhalten eines Systems führen. Um dies zu verhindern können Dienste wie Amazon GuardDuty, Microsoft Azure Advanced Threat Protection oder Alibaba Cloud Security Center eingesetzt werden (Bai, 2019).

System-Integrationsdienste

Die Betreibung eines Softwaresystems in der Cloud erleichtert die Umsetzung indem bereitgestellte Dienste genutzt werden können, anstatt Funktionen neu zu implementieren. Clouddienste können für die Betreibung von Ressourcen verwendet werden, wie für die Überwachung, Bereitstellung oder Konfiguration oder auch Dienste für komplexe Funktionalitäten wie Maschine Learning, Text oder Spracherkennung können ohne Fachwissen in den verschiedenen Domains genutzt werden.

- Monitoring: Ein System aus vielen verschiedenen Konzepten erschwert die Instandhaltung der Software, da die Metriken und Logs von den jeweiligen Komponenten separat entstehen. Monitoring Tools ermöglichen die benötigten Daten zentral zu sammeln und zu analysieren, damit gegebenenfalls bei Fehlern

oder Lastgrenzen, die Verfügbarkeit oder die Leistung erhöht wird. Cloudprovider bieten mit einem Log Center und der Kombination von Maschine Learning Diensten, die Möglichkeit automatisiert Unregelmäßigkeiten des Systems zu erkennen und somit die Wartung des Systems zu unterstützen. Dienste wie Amazon CloudWatch, Microsoft Azure Monitor oder Alibaba Cloud CloudMonitor ermöglichen es Metriken von Applikationen oder anderen Ressourcen in Echtzeit anzuzeigen und Events bei Thresholds auszuführen.

- Datenanalyse: Anwendungen im Geschäftsumfeld bieten meist die Möglichkeit, die erfassten Daten eins Systems auszuwerten. Datenanalysen erfordern aggregierte Daten von verschiedenen Services, die geladen, transformiert und in einem Data Lake oder Data Warehouse abgelegt werden. Analytics Clouddienste ermöglichen eine uneingeschränkte Kapazität und eine Automation für die Einbindung und Transformation von Daten von benutzten Clouddiensten. Als Data Warehouse kann beispielsweise Amazon Redshift, Microsoft Azure Synapse Analytics oder Alibaba Cloud MaxCompute genutzt werden.

- Softwarebereitstellung: Die Softwareentwicklung verwendet Continuous Integration (CI) und Continuous Delivery (CD) Prozesse um den Source-Code zu kompilieren, testen und bereitzustellen. Dieselbe Automatisierung der Prozess kann mit Clouddiensten umgesetzt werden, indem Dienste des Cloudanbieters verwendet werden, wie beispielsweise AWS CodePipeline oder Microsoft Azure Pipelines. Die Integration zu den verwendeten Ressourcen wird somit erleichtert.

- Maschine Learning: Der Einsatz von Künstlicher Intelligenz und Maschine Learning ist eine aufstrebende Funktionalität, die bei Softwaresystemen immer häufiger eingesetzt wird. Die Implementierung dieser Dienste umfasst eine hohe Komplexität und hohen Verwaltungsaufwand. Um dies zu erleichtern, bieten Cloudprovider die Möglichkeit bereitgestellte Dienste zu verwenden, ohne ein Expertenwissen im Bereich künstlicher Intelligenz im Unternehmen zu benötigen. Die Dienste ermöglichen, Modelle zu erstellen, diese zu trainieren und auszuwerten. Verwaltete Services für Maschine Learning sind beispielsweise Amazon SageMaker, Microsofft Azure Machine Learning oder Alibaba Cloud Machine Learning Platform for AI.

- Kognitive Dienste: Systeme mit unterschiedlichen Eingabemöglichkeiten mit Bildern, Text oder Sprache benötigen Funktionen, die die unstrukturierten Daten auswerten und in strukturierte Daten oder ausführbare Aktionen transformiert. Die Transformation benötigt kognitive Fähigkeiten, um die Formate bewerten zu können, dies wird mit künstlicher Intelligenz gelöst. Dienste können beispielsweise für die Bilderkennung, Spracherkennung, Übersetzungen oder Textanalysen genutzt werden. Cloudprovider bieten eine umfangreiche Auswahl an Diensten für die verschiedenen Anwendungsfälle, wie beispielsweise bei AWS, Microsoft Azure oder Alibaba Cloud.

- Migration: Eine Umstellung einer Infrastruktur von On-Premises zu einer Cloudlösung oder die Migration von einem Anbieter zu einem anderen Cloudprovider, kann eine Schwierigkeiten mit sich bringen. Um den Schritt in die Cloud zu erleichtern oder einen Lock-in-Effekt eines Cloudproviders zu entgehen, werden Clouddienste für den Umzug eines Systems angeboten. Die Migration kann in verschiedenen Ausprägungen erfolgen, wie „Lift and Shift“, bei der die bestehende Architektur unverändert bleibt. Bei Re-Plattform wird die Architektur an die Cloud-Infrastruktur adaptiert oder auch ein Refactoring des Systems. Die unterschiedlichen Ausprägungen werden mit Diensten von Cloudanbietern unterstützt, wie beispielsweise AWS Migration Hub, Microsoft Azure-Migrationscenter oder Alibaba Cloud Data Transport (Klaffenbach et al., 2019).

Marketplace

Eine weitere Möglichkeit Dienste in der Cloud zu nutzen ist die Nutzung eines Marketplace von Cloudanbietern. Applikationen aus dem Marketplace erleichtern den Einsatz von Third-Party Diensten. Mit einem Klick können SaaS Dienste von unabhängigen Anbietern in der Cloud benutzt werden. Die Auswahl umfasst eine große Variation von unterschiedlichen Diensten, wie SQL-Datenbanken, Firewall-Dienste, Maschine Learning oder auch unterschiedliche Business Applikationen. Wie bereits erwähnt, kann der Marketplace eine Erleichterung für die Betreibung eines Softwaresystems sein, ebenfalls kann der Marketplace Softwareanbieter eine zusätzliche Verkaufsmöglichkeit deren System bieten, wenn die Anwendung als Software-as-a-Service im Marketplace angeboten wird (Laszewski et al., 2018).

3.2.1.2 Zahlungsmodelle

In diesem Abschnitt werden die Zahlungsmodelle der Cloudprovider Amazons Web Services, Microsoft Azure und Alibaba Cloud verglichen. Die Verrechnung erfolgt bei den unterschiedlichen Anbietern auf eine ähnliche Weise, wie aus den folgenden Übersichten hervorgeht. Die Berechnung der Kosten variiert zwischen den verschiedenen Diensten, ob beispielsweise die Verwendung pro Sekunde, Stunde oder anderen Einheiten abgerechnet wird. Zusätzlich gibt es noch eine Unterscheidung wie die Ressourcen einem Kunden zugewiesen werden, um die Dienste nutzen zu können. Dabei werden die Ressourcen in die folgenden Kategorien eingeteilt:

- On-Demand: Die unabhängigste Möglichkeit ist einen Dienst mit einer On-Demand Option zu verwenden. Der Dienst kann ohne Einschränkung und ohne zeitlich gebundenen Vertrag genutzt werden. Die Verrechnung erfolgt nur bei der Benutzung der Dienste in den verschiedenen Einheiten, wie pro Sekunde oder Stunde. Nach Beendigung eines Dienstes fallen keine weiteren Kosten

mehr an bzw. nur die Kosten der verwendeten Einheit, wie beispielsweise eine gesamte Stunde verrechnet wird sobald der Dienst verwendet wurde oder mehr als eine Einheit bereits gelaufen ist. AWS nennt dieses Preismodell, wie auch die Kategorie genannt wurde, On-Demand. Bei Microsoft Azure findet man diese Variante als nutzungsbasierte Bezahlung und bei Alibaba Cloud als Pay-As-You-Go.

- Reserved: Eine weitere Möglichkeit Clouddienste zu nutzen, kann mit sogenannten reservierten Ressourcen erfolgen. Im Vergleich zu einer On-Demand Bezahlung bieten reservierte Ressourcen erhebliche Vergünstigungen, da der Cloudprovider die benötigten Ressourcen genauer planen kann und der Kunde sich für einen längeren Zeitrahmen an das Produkt bindet. Die Bezahlung kann monatlich und als Vorausbezahlung bzw. als teilweise Vorausbezahlung erfolgen. Die meisten Anbieter ermöglichen, dass das reservierte Kontingent in verschiedenen Instanz-Größen während der Laufzeit zu adaptieren, wie beispielsweise vier kleine Instanzen werden zu einer großen Instanz kombiniert. Bei Amazon Web Services und Microsoft Azure findet man diese Option als reservierte Instanz und bei Alibaba Cloud als Subscription.

- Spot-Instanzen: Die kostengünstigste Variante von Clouddiensten ist eine Spot-Instanz, wobei diese Möglichkeit meist nur bei virtuellen Maschinen (VMs) oder Container verfügbar ist. Eine Spot-Instanz nutzt nicht verwendete Compute-Kapazitäten von Rechenzentren für die Bereitstellung der Dienste. Aus diesem Grund gibt es hohe Rabatte bei Spot-Instanzen. Jedoch kann es dazu führen, dass die Instanz nur für eine kurze Zeit verfügbar ist, wenn der Cloudprovider die Ressourcen für andere On-Demand Dienste benötigt oder wenn der Preis die definierte Grenze überschreitet. Die Verrechnung erfolgt auf Basis der genutzten Ressourcen. Der Preis für die einzelnen Einheiten kann je nach Verfügbarkeit variieren. Aus diesem Grund bieten die Cloudanbieter die Möglichkeit eine Preisgrenze zu definieren, ist der Preis unter dieser Grenze kann die Instanz genutzt werden, ansonsten wird die Instanz gestoppt. AWS und Microsoft Azure verwenden den Begriff Spot-Instanz, bei Alibaba Cloud findet man es als „preemptible“ Instanzen (Salam et al., 2015).

Die Nutzung von Clouddiensten wird nach unterschiedlichen Einheiten verrechnet, die Einheit variiert meist je nach Art des Dienstes, wobei sich die Kosten aus mehreren Einheiten zusammenstellen können. Serverless-Dienste werden meist für die Benutzung von der Kapazität (CPU und Arbeitsspeicher) sowie für die Laufzeit verrechnet. Die folgende Auflistung gibt einen Überblick über mögliche Einheiten:

- Sekunde, Stunde, Monatliche Gebühr: Die Verrechnung auf Basis der Zeit in verschiedenen Einheiten erfolgt meist bei Diensten, die Ressourcen für die Datenverarbeitung über einen Zeitperiode benötigen, wie beispielsweise VMs, Container oder Dienste mit künstlicher Intelligenz.

- GB, Monthly Active User: Dienste, die nur eine geringe Rechenleistung benötigen und nur über die Anzahl der Einheiten mehr Kosten erzeugen, werden auch in der jeweiligen Einheit abgerechnet, wie beispielsweise Gigabyte oder Monthly Active User. Die Abrechnung erfolgt pro Einheit oder auch stufenweise, dass ein Kontingent bezahlt werden muss, wie beispielsweise das Speicher schrittweise gekauft werden kann.

- Anforderungen bzw. Abfragen: Dienste, die für bestimmte Funktionen zur Verfügung stehen oder jede Ausführung dieselben Ressourcen benötigt, werden pro Anfrage bzw. Abfrage abgerechnet. Die Abrechnung erfolgt jedoch meist in größeren Schritten, wie beispielsweise pro einer Mio. Anfragen. Ein Einsatzgebiet sind beispielsweise Serverless-Dienste, die für jeden Aufruf verrechnet werden, jedoch wird meist zusätzlich eine Gebühr auf Basis der Laufzeit verrechnet.

- Kalkulatorische Einheiten: Cloudprovider verwenden kalkulatorische Einheiten bei Diensten, die stark variieren bzw. umfangreich eingesetzt werden können. Die Berechnung einer Einheit erfolgt auf Basis von mehreren Kriterien, wie beispielsweise AWS die Einheit Load Balancer Capacity Unit für die Verrechnung eines Load-Balancer verwendet. Diese betrachtet für die Berechnung die Anzahl an neuen Verbindungen, aktiver Verbindungen, verarbeitete Bytes und die Anzahl an ausgewerteten Regeln.

Als Beispiel für die unterschiedlichen Zahlungsmodelle ist in der Abbildung 3.7 ein Vergleich der Cloudprovider Amazon Web Services, Microsoft Azure und Alibaba Cloud für die Verwendung einer virtuellen Maschine mit zwei vCPUs und acht GB RAM ersichtlich, die in der Region Deutschland / Frankfurt läuft. Die Kosten wurden in Dollar pro Stunde ($/h) angegeben. In der Übersicht ist ersichtlich, dass die Kosten bei einer On-Demand Verrechnung am höchsten sind und die Kosten bei reservierten oder Spot-Instanz abnehmen. Wobei die Kosten für die Spot-Instanz nicht angegeben werden konnten, da die Preise bei AWS und Alibaba Cloud variabel sind.

Linux VM mit 2 vCPU, 8GB RAM in Frankfurt	Amazon EC2	Microsoft Azure Virtuelle Computer	Alibaba Cloud Elastic Compute
On-Demand	0,08 $/h	0,112 $/h	0,114 $/h
Reserved für 1 Jahr	0,052 $/h	0,074 $/h	0,087 $/h
Spot-Instanz	Preis je nach Verfügbarkeit	0,0357 $/h	Preis je nach Verfügbarkeit

Abbildung 3.7: Kosten für eine virtuelle Maschine in Frankfurt (Quelle: eigene Darstellung)

3.2.2 Cloud Plattform Bereitstellung

Applikationen in der Cloud bereitzustellen ermöglicht eine Nutzung von hochverfügbaren IT-Ressourcen. Clouddienste sind konzipiert, um uneingeschränkt Ressour-

cen zu skalieren bzw. Ressourcen zu reduzieren je nach Auslastung und Anforderungen. Die Verwendung von Clouddiensten reicht jedoch nicht aus, um eine Applikation variabel zu betreiben, dazu muss die Architektur des Softwaresystems darauf ausgerichtet sein. Wie bereits im Abschnitt 2.2 erläutert, hat sich die Softwarearchitektur von einem monolithischen Ansatz mit mehreren Servern oder VMs zu einer Cloud Native Architektur mit Microservices und Containern entwickelt bzw. bis hin zu einer funktionalen Programmierung und einer Serverless-Infrastruktur. Aus diesem Grund wird sich die verwendete Architektur für die Bereitstellung von digitalen Plattformen in der Cloud auf diese drei Architektur Prinzipien fokussieren.

3.2.2.1 Monolithische Architektur

Eine monolithische Softwarearchitektur kombiniert die Funktionen einer Softwareanwendung in einer Codebasis, die als eine Gesamtapplikation auf einem Server ausgeführt wird. Diese Architektur wurde in früheren Implementierungen sehr häufig eingesetzt, wobei auch heutzutage noch viele monolithische Applikationen im Einsatz sind. Es gibt Einsatzszenarien, in denen diese Architektur zu bevorzugen ist, wie beispielsweise bei kleinen Softwareanwendungen mit einer hohen Abhängigkeit zwischen den Funktionen. Die einfache Handhabung durch die Verwendung einer Softwarekomponente für das gesamte System und somit keine Verteilung der Applikation. Dies bringt jedoch Herausforderungen für die Betreibung in der Cloud. Den Prinzipien einer Cloud für eine hochverfügbare und performanten Applikation sowie die Skalierbarkeit können nur bedingt auf eine monolithische Architektur übertragen werden. Die Bereitstellung kann durch virtuelle Maschinen als Clouddienst erfolgen. Cloudprovider stellen eine VM bereit, auf der die Applikation separat installiert werden muss. Die Applikation selbst ist somit nicht abgesichert, treten Fehler in der Applikation auf, bedeutet dass, das System nicht mehr erreichbar ist. Um die Ausfallsicherheit und die Performance zu erhöhen können mehrere Instanzen der Applikation betrieben werden, wobei dies auch nur mit der gesamten Applikation möglich ist. Es müssen somit mehrere virtuelle Maschinen mit einer jeweiligen Instanz der Applikation laufen, um eine erhöhte Ausfallsicherheit garantieren zu können. Die Skalierung kann nicht granular für die unterschiedlichen Funktionen durchgeführt werden. Abhängigkeiten zu anderen Funktionen, wie beispielsweise ein Identity Access Management System oder eine Datenbank, können mit Clouddiensten ersetzt werden, wenn es die Architektur zulässt. Eine Datenbank, die als Clouddienst angeboten wird, erleichtert die Verwaltung des gesamten Systems, in dem nur die Softwareapplikation gewartet werden muss. Der Vorteil bei einer monolithischen Architektur ist, dass die Entwicklungs- und Deploymentprozesse übersichtlich sind, durch die einheitliche Code- und Applikationsbasis. Die gesamtheitliche Systemkomponente erleichtert zudem die Verwaltung bzw. den Support der Applikation, da die Log-Informationen nur von einer Applikation stammen. Als Beispiel für die Bereitstellung einer monolithischen Architektur in der Cloud wurde in der Abbildung 3.8 eine Applikation mit der Verwendung von virtuellen Maschinen eines Cloudproviders zweimal be-

reitgestellt und mit einem Load-Balancer des Cloudproviders werden die Anfragen aufgeteilt. Die Daten der Applikation werden synchron in einer relationalen Cloud-Datenbank abgelegt. Wie bereits erwähnt, müsste eine weitere Instanz der virtuellen Maschine genutzt werden, wenn die Applikation weiter skaliert werden müsste oder die zugewiesenen Ressourcen könnten für die virtuellen Maschinen erhöht werden (Ingeno, 2018).

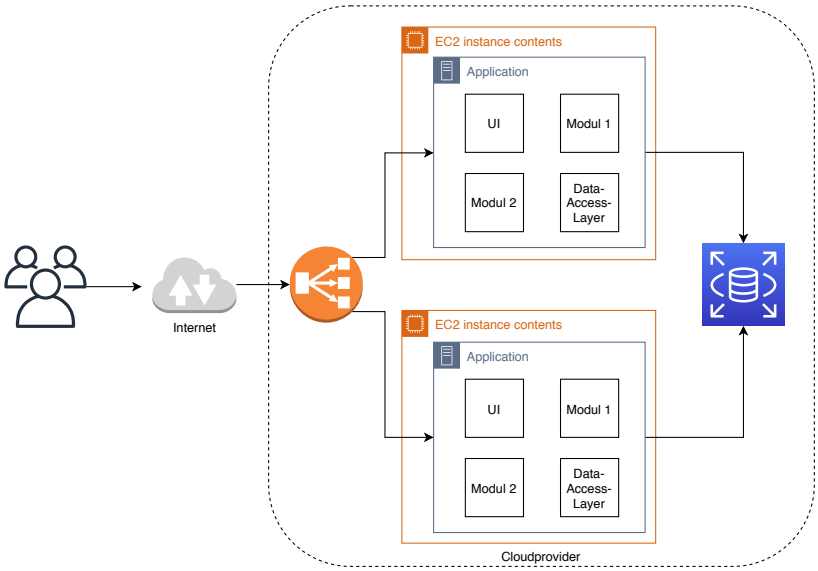

Abbildung 3.8: Monolithische Applikation in der Cloud (in Anlehnung an Ingeno, 2018)

3.2.2.2 Microservice Architektur

Eine Architektur, die mit Cloud Computing besser einhergeht, ist eine Microservice Architektur. Microservices ermöglichen ein verteiltes System, indem eine Software Anwendung in mehrere einzelne unabhängige Services aufgeteilt wird. Ein Service kann unabhängig von anderen Services betrieben werden und kommuniziert über standardisierte Protokolle wie REST oder stellt die Funktionen über definierte Schnittstellen bereit. Microservices werden in verschiedene Funktionsbereiche aufgeteilt, wobei die Komplexität einzelner Funktionen im Service implementiert wird und für die Anwender, Userinterface oder anderes Service, als einfachen Funktionsaufruf zur Verfügung stehen sollte. Funktionsabläufe ergeben sich durch Aufrufen von mehreren Schnittstellen von unterschiedlichen Services. Im Vergleich zur monolithischen Architektur wird durch die Aufteilung der Applikation eine feingranulare Anpassungsmöglichkeit und Skalierungsmöglichkeit geschaffen. Die Services können unabhängig voneinander upgedatet werden. Bei Anpassungen von Funktionsabläufen sollten neue Schnittstellen angeboten werden, um mögliche Integrationen von anderen Services nicht zu unterbrechen. Die Skalierung kann sich für jedes Service unterscheiden, um somit zu ermöglichen, dass Funktionen mit einer hohen Last, höher

skaliert werden im Vergleich zu unkritischen Funktionen. Die Unabhängigkeit zwischen den Microservices ermöglicht den Einsatz von unterschiedlichen Programmiersprachen und Frameworks bei jedem Service, um die Technologien an die Funktionen anzupassen. Um beispielsweise die Performance für komplexe Berechnungen durch den Einsatz von einer datenoptimierten Programmiersprache oder einer anderen Datenbank zu optimieren. Außerdem bieten die unabhängigen Services die Vorteile der Nutzung von Clouddiensten, in dem mehrere Instanzen der jeweiligen Services mit wenig Ressourcen betrieben werden können. Dadurch werden potenzielle Fehler auf einzelne Services eingeschränkt und betreffen nicht das gesamte System. Cloud Native verwendet die Microservice Architektur mit der Bereitstellung durch Container und deren Orchestrierung. Die Abbildung 3.9 zeigt eine exemplarische Bereitstellung einer Applikation mit Microservices in der Cloud. Die Applikation besteht aus zwei Services die in einem Kubernetes Cluster als Container orchestriert werden, wobei das Service 1 auf zwei Instanzen skaliert wurde. Beide Services verwenden eine eigene Clouddatenbank mit unterschiedlichen Technologien. Das Webinterface wird über einen eigenen Datendienst bereitgestellt, damit User ein Webinterface benutzen können, das über einen Load-Balancer oder einen API-Gateway auf die Schnittstellen der Microservices zugreift (Laszewski et al., 2018).

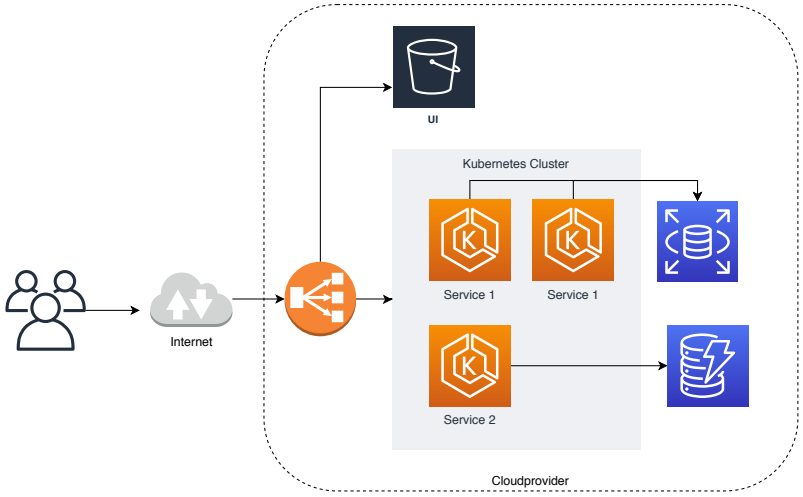

Abbildung 3.9: Microservice Applikation in der Cloud (in Anlehnung an Ingeno, 2018)

3.2.2.3 Serverless-Architektur

Eine Architektur ohne die Verwaltung einer Infrastruktur ermöglicht die Serverless-Architektur. Die Serverless-Architektur bietet eine direkte Bereitstellung einzelner Funktionen bei der die Server oder Container nicht gewartet werden müssen, sondern ein Clouddienst führt einen bereitgestellten Quellcode bei Bedarf aus. Serverless-Architekturen sind auch als Function-as-a-Service (FaaS) bekannt, da einzelne Funktionen einer Softwareapplikation als unabhängige Functions implementiert werden

und separat gewartet sowie ausgeführt werden. Die Functions sind intern im System verfügbar oder werden als API-Endpunkte bereitgestellt. Es sollte darauf geachtet werden, dass die Ausführung keine langandauernden Berechnungen abwickelt. Cloud-Functions wurden für die Ausführung von kurzen Aktionen konzipiert, wie beispielsweise Event-Workflows, Datentransformation oder Generieren von Dokumenten. Ein Softwaresystem mit komplexen und langandauernden Aktionen kann mit Microservices und Container kombiniert werden, da diese zeitlich nicht limitiert sind. Die Serverless-Functions sollten so konzipiert sein, dass eine Anforderung abgedeckt wird, durch eine vordefinierte Schnittstelle angesprochen wird und einen definierten Rückgabewert zurückgibt. Functions werden über einen API-Gateway bereitgestellt bzw. werden Functions von anderen Functions aufgerufen. Die Ausführung kann dabei synchron oder asynchron durch Events oder Queues ausgeführt werden, wobei darauf geachtet werden sollte, dass verschiedene Dienste mit den Cloud-Functions von Cloudprovidern kompatibel sind. Die Skalierung ist durch die nicht vordefinierten Serverressourcen nicht eingeschränkt bzw. wird nur definiert, wie viele Ressourcen (CPU und Arbeitsspeicher) für jede Ausführung vorhanden sind und die Last bei hoher Anzahl an Ausführungen wird vom Cloudprovider verwaltet. Aus diesem Grund werden keine Ressourcen verwendet, die nicht benötigt werden, aber es schränkt die Nutzung auch nicht ein. Eine einfache Verwendung von einer Serverless-Architektur zeigt das Beispiel in der Abbildung 3.10. Zur Veranschaulichung bietet die Applikation nur drei Functions an, einen Eintrag zu erstellen und zu laden. Der Eintrag wird dabei in einer relationalen Datenbank des Cloudproviders abgelegt. Eine weitere Cloud-Function stellt Daten einer weiteren Datenbank zur Verfügung. Die einzelnen Functions werden mit Hilfe eines API-Gateways einen User zur Verfügung gestellt, der mit einem Userinterface, das von einem Storage bereitgestellt wird, nutzen kann (Bangera, 2018).

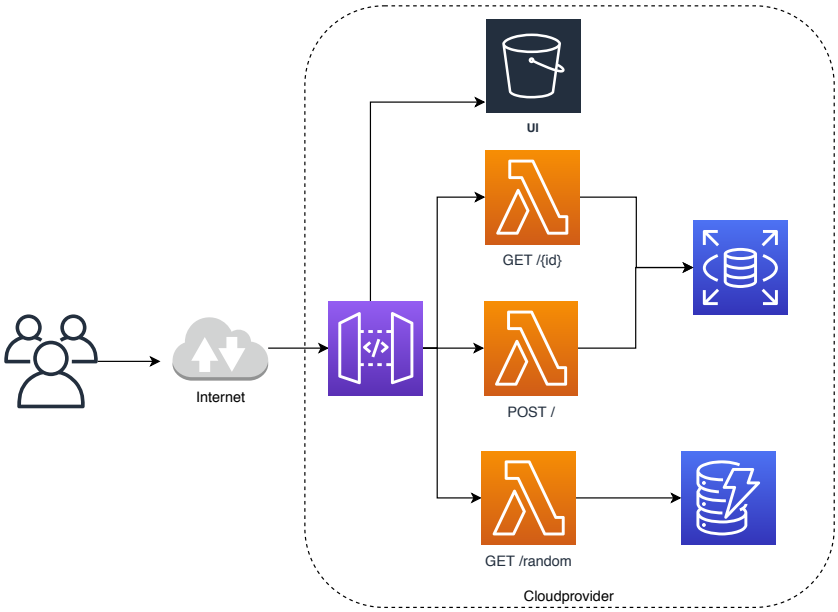

Abbildung 3.10: Serverless Applikation in der Cloud (in Anlehnung an Smith, 2020)

3.2.2.4 Integrationsmöglichkeiten

Eine Softwaresystem, das als Software-as-a-Service bereitgestellt wird, kann unterschiedliche Anforderungen für jeden Kunden auf das System haben. Aus diesem Grund gibt es verschiedene Vorgehensweisen, diese Anforderungen umzusetzen, wie bereits im Abschnitt 3.1.1.3 erläutert. Eine Möglichkeit ist für jeden Kunden eine eigene Codebasis zu verwalten, um den Wartungsaufwand von vielen unterschiedlichen Versionen der Applikation zu vermeiden. Ein System kann Integrationsmöglichkeiten bereitstellen, um die unterschiedlichen Anforderungen in einem System zu lösen. Einfache Adaptionsmöglichkeiten reduzieren nicht nur den zeitlichen Aufwand, sondern reduzieren die Fehlerwahrscheinlichkeit bei Anpassungen. Die Anpassungen können mit einer Architektur erfolgen, die bereits Anpassungsmöglichkeiten in abgebildeten Prozessen der Applikation abbildet. Einerseits ist es möglich, Prozesse auf Basis von kundenspezifischen Kriterien verschieden abzuarbeiten, dies kann beispielsweise mit einer Business Rule Engine als Teil der Applikation erfolgen. Andererseits kann es notwendig sein, dass der gesamte Prozess erweitert werden muss oder kundenspezifische Funktionen in die Applikation aufgenommen werden. Beide Möglichkeiten basieren auf einem ähnlichen Konzept, auf Basis von einer Aktion im System, sollen Daten bewertet werden und eine Aktion im System ausführen, wie beispielsweise mit einem Event-Driven Ansatz. Eine Business Rule Engine wird dazu genutzt, um bei der Ausführung von bestimmten Aktionen eine kundenspezifische oder mandantenspezifische Regel auszuführen, die anschließend auf Basis von Kriterien eine Aktion durchführt. Dieses Konzept wurde vorwiegend bei einer monolithischen Architektur eingesetzt, da bei einer Microservice Architektur der Verwaltungsaufwand von Rules für unterschiedliche Services, eventuell auch noch in unterschiedlichen Programmiersprachen, eine Herausforderung bieten kann (Ivanov, Sinderen, Leymann & Shan, 2013).

Anstelle einer Business Rule Engine kann eine Event-Driven Architektur verwendet werden. Diese kann in mehreren Ausprägungen erfolgen, wie beispielsweise das Events bei verschiedenen Aktionen im System erstellt werden, auf denen Integrationen sich registrieren können oder eine gesamte Event-Driven Architektur, bei der die gesamte Applikation mit Event-Streams abgebildet wird. Events können als Blocking oder als Bereitstellung von Informationen angeboten werden. Ein Blocking-Event unterbricht den Funktionsablauf bis eine Antwort für diesen Event geliefert wurde. Der Event kann ein externes System oder eine mandantenspezifische Funktion aufrufen, die beispielsweise den Ablauf erweitert und anschließend den Standardprozess abschließt, sobald die Daten verarbeitet wurden. Ein informativer Event stellt nur die Information zur Verfügung, dass eine bestimmte unabhängige Aktion der Applikation ausgeführt wird. Eine mögliche Anforderung bei einer E-Commerce Plattform könnte sein, dass eine Nachricht versendet wird sobald eine Bestellung aufgegeben wurde. In diesem Fall kann die Nachricht asynchron versendet werden und unterbricht den weiteren Ablauf der Bestellung im System nicht. Microservice oder Serverless-Architekturen verwenden eine Event-Driven Architektur außerdem, um das gesamte System bzw. die einzelnen Services miteinander zu verbinden. Funktio-

nen werden von den einzelnen Services abstrahiert und durch die Konfiguration von Events zusammengefügt. Dieser Ansatz ermöglicht, dass Funktionen in unterschiedlichen Szenarien wiederverwendet werden können und, dass bestimmte Events für Tenants angepasst werden können. Ein Event kann angepasst werden indem eine mandantenspezifische Implementierung aufgerufen wird, um eine Out-of-the-Box Funktionalität zu ersetzen oder eine Funktion zu erweitern. Erweiterungen können synchron oder asynchron zu den bestehenden Abläufen ausgeführt werden. Die Herausforderung bei dieser Architektur ist, dass die mandantenspezifischen Anpassungen mit der Out-of-the-Box Funktionalität kompatibel sind und systemkritische Abläufe nicht angepasst werden können (Bellemare, 2020).

4 Entwurf

In den vorherigen Kapiteln wurden verschiedene Ausprägungen von digitalen Plattformen, Architekturmustern und Möglichkeiten für die Bereitstellung von Plattformen mit Clouddiensten erläutert. In diesem Kapitel werden nun die erarbeiteten Informationen in Beispielen angewandt, um die unterschiedlichen Ausprägungen auf Basis von technischen und ökonomischen Kriterien zu bewerten. Die erläuterten Cloudprovider, Amazon Web Services, Microsoft Azure und Alibaba Cloud, zeigen in der Übersicht, dass die meisten Clouddienste von den meisten Anbietern in einer ähnlichen Weise angeboten werden. Die Preisgestaltung der Cloudprovider ähnelt sich sehr bzw. weicht nur teilweise ab. Aus diesen Gründen wird für den Entwurf dieser Arbeit der größte Cloudprovider, AWS, als exemplarischer repräsentativer Cloudprovider verwendet. AWS hatte im Jahr 2019 mit 45% den größten Marktanteil. Als Beispiele für den Einsatz von digitalen Plattformen wird die Branche E-Commerce verwendet. Durch den Einsatz einer digitalen Plattform und eines Plattform Business Modells, hat Amzaon sich zur umsatzstärksten Marke 2019 gemacht und und das enorme Potenzial einer Plattform gezeigt.

4.1 Bewertungskriterien

Die Bewertung einer Architektur, die in der Cloud angeboten wird, benötigt unterschiedliche Betrachtungsweisen. Die Auswahl einer Lösung für eine digitale Plattform kann nicht nur aufgrund von technischen Kriterien getroffen werden. Da verschiedene Möglichkeiten geboten werden um ein System in der Cloud bereitzustellen, aber es auch verschiedene Möglichkeiten für die Konzeption von Systemen gibt, wurde bereits erläutert. Hinzu kommt, dass ein Plattform Geschäftsmodell, einige strategische Herausforderungen mit sich bringt, die in das System mit einfließen müssen, wie beispielsweise die Anzahl an unterschiedlichen Kundengruppen, die eine Plattform nutzen. Aus diesem Grund erfolgt die Bewertung auf Basis von drei verschiedenen Aspekten, nicht-funktionale, ökonomische und technologische Aspekte.

4.1.1 Nicht-Funktionale Aspekte

Nicht-Funktionale Aspekte repräsentieren Qualitätsaspekte und Eigenschaften eines Systems, unabhängig von spezifischen eingesetzten Technologien. Die verschiedenen Eigenschaften können durch die Verwendung von verschiedenen Architekturmustern oder Technologien erfüllt werden. Somit können verschiedene Bereitstellungsarten der Cloud, die gleichen Eigenschaften in unterschiedlichen Vorgehensweisen erfüllen. Dies ist ein Grund für die Verwendung von unterschiedlichen Kriterien für die Bewertung von Cloudbereitstellungsmethoden. Die folgenden nicht-funktionale Aspekte werden in der Bewertung betrachtet:

- Skalierbarkeit: Ein wichtiger Faktor bei der Nutzung von Clouddiensten ist die mögliche Skalierbarkeit der Dienste. Der Vorteil bei der Nutzung einer Cloudplattform soll sein, dass ein Softwaresystem ohne Einschränkung skaliert werden kann. Ein eigenes Datacenter würde nur eine begrenzte Skalierung zulassen, da die Hardwareressourcen nur begrenzt verfügbar sind. Es sollte möglich sein, die Infrastruktur auf die gegebene Last dynamisch anzupassen. Die Skalierung kann horizontal oder vertikal erfolgen. Eine horizontale Skalierung bedeutet, dass mehre Instanzen des Dienstes genutzt werden und die Last auf diese Instanzen aufgeteilt wird, um die erforderte Leistung zu erfüllen. Eine vertikale Skalierung erfolgt, indem die verwendeten Ressourcen einer Instanz erhöht werden, um beispielsweise durch einen größeren Arbeitsspeicher, Daten performanter bearbeitet werden können. Der wichtigste Faktor der Skalierung von Clouddiensten ist, dass die Ressourcen schnell hoch skaliert oder verringert werden können. Außerdem sollte die Skalierung feingranular erfolgen können, um Dienste an den erforderten Ressourcen für eine bestimmte Auslastung genau anzupassen und somit Ressourcen nicht unnötig genutzt werden. Zudem sollte diese Anpassung dynamisch und automatisiert erfolgen. Die Bewertung erfolgt mit einer Skala von eins bis fünf. Wobei eins bedeutet, dass die genannten Faktoren nicht erfüllt werden und fünf bedeutet, dass alle Faktoren erfüllt werden.

- Zuverlässigkeit (Reliability): Bei der Verwendung von Systemen mit unterschiedlichen Daten ist es wichtig, dass den bereitgestellten Daten vertraut werden kann und die Daten korrekt abgearbeitet werden. Zuverlässigkeit eines Clouddienstes ist die Möglichkeit, Vorgänge konsistent ohne Unterbrechungen ausführen zu können. Fehler sollten beispielsweise zu keinem Datenverlust oder Ausfall der Ausführung führen. Zuverlässigkeit und Verfügbarkeit haben Abhängigkeiten, wobei sich die Verfügbarkeit auf die Korrektheit des Systems bezieht und Verfügbarkeit ist die Möglichkeit einen Vorgang auszuführen. Die Zuverlässigkeit kann durch redundante Diensten und dementsprechende Fehlerbehandlung erfolgen, um ein Fehlverhalten zu vermeiden. Die Bewertung erfolgt mit einer Skala von eins bis fünf. Eins bedeutet eine kaum verfügbare Zuverlässigkeit und fünf ist ein hoch zuverlässiges System.

- Erweiterbarkeit / Integrierbarkeit: Ein Softwaresystem kann unterschiedliche Anwendungsszenarien haben und wird ständig weiterentwickelt. Die Bereitstellung eines System in der Cloud sollte es ermöglichen, schnell und einfach auf Anpassungen reagieren zu können. Die Erweiterbarkeit unterscheidet sich somit von der Skalierbarkeit, dass neue Funktionen hinzugefügt oder Funktionen adaptiert werden können, anstatt die mögliche Last auf ein bestehendes System zu erhöhen. Anpassungen können einerseits Anpassungen des gesamten Systems oder der genutzten Infrastruktur sein aber auch Erweiterungen der Applikation. Die Erweiterbarkeit bezieht sich auf die Möglichkeit ein Standardprodukt anzupassen, wohin gegen die Integrierbarkeit, die Möglichkeit das Standardprodukt für einen Kunden bzw. einer Kundengruppe zu erweitern. Die Bewertung dieser Eigenschaft erfolgt zwischen eins und fünf. Eins bedeutet, dass Anpassungen am System oder Integrationen am System mit viel Aufwand verbunden sind und das gesamte System betroffen ist. Funktionsanpassungen können nicht unabhängig erfolgen. Fünf ist ein System, wenn einzelne Funktionen unabhängig von anderen Funktionen adaptiert oder erstellt werden können. Die Integration erfolgt getrennt vom Standardprodukt und hat keine direkten Rückwirkungen auf das bestehende System.

- Verfügbarkeit (Availability): Ein bereitgestelltes System sollte sich nicht nur konsistent verhalten, es sollte zudem dauerhaft verfügbar sein und zu den definierten Zeitpunkten zur Verfügung stehen. Die Verfügbarkeit von Clouddiensten bietet den Vorteil, dass die Verfügbarkeit nicht von einer gewarteten Hardware abhängt, sondern auf eine großenAuswahl an Hardware des Cloudproviders verteilt wird. Die Verfügbarkeit von Diensten kann durch Replikation von Daten und Load-Balancing erfolgen. Daten oder Services sind dadurch mehrfach verfügbar und können bei Ausfällen einzelner Instanzen durch weitere Instanzen überbrückt werden, bis die gewünschte Anzahl an Instanzen wieder verfügbar ist. Die Verfügbarkeit wird mit einer Skala von eins bis fünf bewertet, wobei eins keine Verfügbarkeit garantieren kann und fünf das System als Hochverfügbarkeit bereitgestellt wird.

- Upgradefähigkeit: Anpassungen bei deiner Software oder deren Infrastruktur müssen laufend ausgerollt werden. Die Upgradefähigkeit beschreibt die Möglichkeit, die Änderungen einfach und mit niedrigem Risiko zu installieren. Um das Risiko von Anpassungen zu reduzieren sollte es möglich sein, nur Teile des Systems unabhängig von anderen zu aktualisieren. Der Ausfall des Systems während des Updates sollte dabei so gering wie möglich sein bzw. nicht vorhanden sein. Die Bewertung erfolgt mit einer Skala von eins bis fünf. Ein System, dass nur gesamtheitlich und mit Down-Time aktualisiert werden kann wird mit eins bewertet, im Gegensatz dazu wird ein System ohne Ausfall des Systems und Aktualisierungen von einzelnen Funktionen mit fünf bewertet (Schubert, Jeffery & Neidecker-Lutz, 2010; Siegel & Perdue, 2012).

4.1.2 Ökonomische Aspekte

Die ökonomische Betrachtung der Verwendung von Clouddiensten ist ein wichtiger Entscheidungsträger, wenn man neue Clouddienste einführt. Eine Infrastrukturumstellung wird meistens durchgeführt, um die Kosten des Systems zu optimieren, Reduzierung des Verwaltungsaufwand einer Infrastruktur sowie schnelle Umsetzung von IT unterstützten Geschäftsmodellen und Prozessen.

- Kosten: Die Optimierung der Kosten ist ein essentielles Einflusskriterium bei der Auswahl von Clouddiensten. Die Relevanz der Kosten für die Betreibung und Verwaltung einer Infrastruktur kann, je nach Geschäftsmodell, variieren und muss mit weiteren Kriterien zusammen betrachtet werden. Ein Beispiel dazu ist, dass bei Geschäftsfeldern, die viele Systeme On-Premises anbieten, nicht zwingend eine kostenoptimierteste Infrastruktur benötigt wird, sondern eine hohe Portabilität des gesamten Systems geboten werden muss, um nicht nur Cloudprovider proprietäre Dienste nutzen zu können. Die Kosten werden für die benötigte Infrastruktur, um ein Softwaresystem bei einem Cloudprovider bereitzustellen berechnet. Ein weiterer Faktor sind die Kosten, die sich für die Verwendung des Systems bei der Nutzung von verschiedenen Anzahlen an Usern ergeben. Je nach Auslastung eines Systems können dadurch unterschiedliche Konzepte kostengünstiger bzw. kostenintensiver ausfallen.

- Fixkosten zu variablen Kosten (Capex into Opex): Die Verwendung von Clouddiensten soll ermöglichen, dass die Fixkosten, die für die Betreibung eines Softwaresystems anfallen weitestgehend reduziert werden. Die Kosten sollen als variable Kosten eingepreist werden, um die Kosten auf Basis der Nutzerzahl zu adaptieren oder dass Kosten nur für die benötigten Ressourcen anfallen. Kaum vorhandene Fixkosten bedeuten somit, dass ein Unternehmen eine Softwareanwendung in einer Testphase mit sehr geringen Kosten bereitstellen kann und wenn das System nicht genutzt wird, keine Kosten anfallen. Die Bewertung erfolgt mit einer Skala von eins bis fünf, wobei eins bedeutet, dass ein großer Teil der anfallenden Kosten Fixkosten sind und fünf eine Infrastruktur ohne anfallende Fixkosten ist.

- Time-to-Market: Digitale Geschäftsmodelle erfordern meist, dass ein Unternehmen schnell auf Kundenbedürfnisse eingehen und Softwaresysteme bereitstellen kann. Aus diesem Grund ist ein wichtiger Faktor einer Cloudarchitektur, dass der Entwicklungs- und Verwaltungsaufwand für neue Systeme gering ist, um Implementation in kürzester Zeit mit wenig zusätzlichen Ressourcen bereitgestellt werden kann. Time-to-Market hat meist Abhängigkeiten zu weiteren technologischen Aspekten, wie beispielsweise die Anzahl der benötigten Tools. Die Bewertung erfolgt auf Basis einer Skala von eins bis fünf. Eine lange Time-to-Market ist eins und eine Umsetzung in kürzester Zeit ist eine Bewertung von fünf (Schubert et al., 2010; Siegel & Perdue, 2012).

4.1.3 Technologische Aspekte

Technologischen Aspekte und Herausforderungen gehen häufig einher mit den nichtfunktionalen und ökonomischen Aspekten. Nicht-Funktionale Anforderungen, wie beispielsweise eine Hochverfügbarkeit, kann technologische Auswirkungen haben, die sich wiederum in den Kosten widerspiegeln können. Die technologische Betrachtung eines Cloudsystems erläutert technische Herausforderungen, die sich bei den unterschiedlichen Architekturen ergeben können.

- Sicherheit: Die Sicherheit von Softwareapplikationen ist ein wichtiges Kriterium bei der Verarbeitung von sensitiven Daten und im Allgemeinen bei der Übertragung von unterschiedlichsten Applikationsdaten. Sicherheit umfasst die Datensicherheit, Privatsphäre sowie Compliance Richtlinien. Die Bewertung erfolgt in welchen Ausmaß Sicherheit des Systems gewährt werden kann, von eins mit einer geringen Sicherheit bis fünf mit einer hohen Sicherheit.

- Messbarkeit / Monitoring: Clouddienste werden häufig als Pay-per-use Lösung bereitgestellt. Dabei ist wichtig, dass die Verwendung eines Systems genau nachverfolgt werden kann. Die Kosten fallen je nach Verwendung einzelner Funktionen an, dazu ist es wichtig, dass eine detaillierte Auswertung des Systems erfolgen kann. Zudem muss ein Softwaresystem laufend überwacht werden, um bei eventuellen Fehlern reagieren zu können. Deshalb erfolgt die Bewertung auf Basis dessen, in welchem Ausmaß ein System gemessen oder überwacht werden kann, dazu wird eine Skala von eins bis fünf verwendet. Eins bietet nur eine sehr grobe Überwachungsmöglichkeit, wohin gehen fünf eine detaillierte Aufschlüsselung ermöglicht.

- Benötigte Tools: Die Bereitstellung einer Softwareapplikation mit der Verwendung von Clouddiensten kann je nach Architektur und Service eine Vielzahl an zusätzlichen Tools benötigen. Die Tools werden für die Entwicklung und für die Verwaltung der Clouddienste benötigt. Eine geringe Anzahl von zusätzlichen Tools bedeutet eine Bewertung von fünf bis zu eins, bei der eine hohe Anzahl an zusätzlichen Tools, für die Umsetzung benötigt werden.

- Portabilität: Die Zielgruppe von einer Softwareapplikation kann stark variieren, je nach Geschäftsmodell, kann somit eine Flexibilität der Bereitstellung benötigt werden. Die Architektur und die Infrastruktur sollte zu verschiedenen Cloudprovidern oder auch On-Premises Lösungen kompatibel sein. Die Portabilität definiert wie sehr die Infrastruktur zu proprietären Diensten gebunden ist bzw. ob sich Lock-in Effekte ergeben. Die Bewertung erfolgt mit einer Skala von eins bis fünf, wobei eins bedeutet, dass ohne einer Anpassung der Applikation der Anbieter nicht gewechselt werden kann und fünf, eine Lösung ist, die ohne Einschränkung auf unterschiedlichen Anbietern bereitgestellt werden kann (Schubert et al., 2010; Siegel & Perdue, 2012).

4.2 Plattform Architekturen

Die Analyse der Bereitstellungsmöglichkeiten von digitalen Plattformen erfolgt aufgrund der durchgeführten Analyse. Die Ausprägungen basieren auf den drei gängigsten Architekturen, monolithische, Microservice und Serverless-Architektur. Die Bereitstellungsmethoden werden für die jeweiligen Architekturen gewählt, die jeweils am häufigsten eigesetzt wird. Aus diesem Grund werden Monolithen mit virtuellen Maschinen, Microservices mit Container und Serverless mit Cloudfunctions bereitgestellt. Ein Konzept für mögliche Integrationen wird nicht erarbeitet. Es wird nur auf die Integrationsmöglichkeiten eingegangen, um kundenspezifische Anforderungen bei der Analyse nicht verfälschend darzustellen.

Die erläuterten Ausprägungen von digitalen Plattformen werden als Grundlage für die Architektur verwendet. Die strategische Entscheidung ein Geschäftsmodell auf eines der vier erarbeiten Ausprägungen der Plattformen aufzubauen, kann genutzt werden, um sich in einem weiteren Schritt für eine Architektur der Plattform und deren Umsetzung zu entscheiden. Auf die Architektur für die Nutzung der Plattform durch mehrere Kunden wird in der Analyse nicht eingegangen, da es für die Solution Architektur keinen direkten Einfluss hat. Multi-Tenancy oder ein dementsprechendes Datenmodell wird auf Applikationsebene umgesetzt, dass sich die auf Komponentenebene der Solution Architektur nicht auswirkt. Es wird von allen drei Architekturen eine zentrale Datenbank bzw. Datenablage unterstützt. Um ein vergleichbares Ergebnis erzielen zu können, wird die Datenablage in der Analyse nicht betrachtet. Aus diesem Grund werden in den unterschiedlichen Modellen dieselben Datendienste bei den jeweiligen Ausprägungen genutzt. Wie bereits am Beginn des Kapitels erläutert, werden die verschiedenen Ausprägungen auf Beispielen aus dem E-Commerce erstellt. Es werden folgende Ausprägungen von digitalen Plattformen erarbeitet:

- One-Sided Plattform ohne Netzeffekte: Eine Plattform wird zur Verfügung gestellt, die von einem Unternehmen eingesetzt wird und einen Wert für dessen Kunden erstellt. Im E-Commerce ist es ein einfacher Onlineshop, bei dem Produkte von einem Unternehmen verwaltet werden und bereitgestellt werden. Kunden können die angebotenen Produkte kaufen, jedoch ohne Interaktion zwischen den einzelnen Kunden, dadurch entstehen keine Netzeffekte. Beispielsweise wenn keine Bewertungen im Onlineshop möglich sind, bekommt ein Kunde keinen Mehrwert bei steigender Kundenanzahl.

- One-Sided Plattform mit direkten Netzeffekten: Eine Plattform bzw. ein Onlineshop, der von einem Unternehmen betrieben wird, ist One-Sided. Durch Funktionen die Interaktionen zwischen den Kunden erlauben entstehen direkte Netzeffekte. Eine höhere Kundenanzahl des Onlineshops bietet die Möglichkeit, dass Kunden Produkte aufgrund von einer hohen Anzahl an Bewertungen auswählen.

- Multi-Sided Plattform mit indirekten Netzeffekten: Bei einer Plattform mit mehreren Kundengruppen, wie im E-Commerce mit Kunden und Verkäufern, gibt es zwei Seiten. Der Verkäufer möchte Produkte im Onlineshop bereitstellen und der Kunde ist auf der Suche nach Produkten. Je mehr Kunden auf der Plattform sind, desto attraktiver wird die Plattform indirekt für Anbieter. Wobei einerseits mehr Anbieter negativ für einzelne Anbieter sein können, anderseits bedeuten viele Anbieter ein umfangreiches Produktportfolio, dass die Plattform für Kunden wiederum attraktiver macht. Dadurch profitieren Anbieter indirekt von anderen Anbietern, da der Wert für den Kunden durch eine vergrößerte Auswahl steigt. Kunden profitieren durch eine größere Kundenanzahl auch nur indirekt von den bereits genannten Kriterien.

- Multi-Sided Plattform mit direkten und indirekten Netzeffekten: Eine weitere Möglichkeit ist, dass eine Plattform mehrere Kundengruppen hat und diese untereinander interagieren können. Eine Plattform, auf der jeder Benutzer Produkte zum Verkaufen anbieten kann oder versteigern kann, profitiert davon, dass mehr User die Plattform nutzen, da das Produkt von mehr Personen gesehen wird. Benutzer können durch die Nutzung der Plattform aber auch günstig zu Produkten von anderen Benutzern kommen. Ein weiterer Mehrwert für die Käufer und auch Verkäufer der angebotenen Produkten ist, wenn mehr Benutzer von einem Verkäufer etwas gekauft haben, dass Verkäufer durch Bewertungen vertrauensvoller werden und somit die Sicherheit für den Käufer steigt, aber auch dass der Verkäufer eine höhere Wahrscheinlichkeit hat, weitere Produkte zu verkaufen.

4.2.1 One-Sided Plattform ohne Netzeffekte

Als One-Sided Plattform ohne Netzeffekte wird ein Onlineshop für den Verkauf von physischen Produkten auf Basis von Referenz-Architekturen konzipiert. Der Onlineshop ermöglicht einem Unternehmen einen Produktkatalog in der Plattform zu konfigurieren, Kunden zu verwalten und das Aufgeben von Kundenbestellungen sowie die Verfolgung von diesen. Die Architektur in den drei Varianten haben denselben Funktionsumfang und bieten die einzelnen Funktionen als REST-API an, sowie ein statisches Webinterface für die Kundeninteraktion mit dem System. Die folgenden Features müssen in dem Onlineshop implementiert werden:

- Konto registrieren, anmelden und abmelden

- Administrator kann Produkte hinzufügen, bearbeiten und löschen

- Produkte könnten aufgelistet und nach Typ oder Marke gefiltert werden

- User kann Produkte in den Warenkorb legen, die Anzahl der Produkte anpassen oder entfernen
- User kann die Produkte des Warenkorbs bestellen
- User kann die Aufträge überprüfen
- Bei abgegeben Bestellungen kann ein externer Bezahlungsdienst eingebunden werden

4.2.1.1 Monolithische Architektur

Die zuvor spezifizierten Anforderungen wurden auf Basis einer Referenz Architektur von Microsoft in eine monolithische Applikation abgebildet. Die Applikation besteht aus vier Modulen für den User, den Produktkatalog, der Warenkorb-Funktionalität und dem Bestellvorgang (Nish Anil, 2020). Die Applikation bietet ein Webinterface für die Userinteraktion, dass die Funktionen mit REST-APIs anbinden. Die Abbildung 4.1 zeigt die Übersicht des Softwaresystems und die Bereitstellung durch Clouddienste. Eine virtuelle Maschine mit Linux wird für den Betrieb der Applikation genutzt, in diesem Fall ist es eine Instanz von Amazon EC2. Die Daten werden in einer relationalen Datenbank abgelegt, die als Clouddienst bereitgestellt wird. Ein Load-Balancer wird verwendet, um die Möglichkeit zu bieten, dass die Last auf mehreren Instanzen der Applikation aufgeteilt wird und um die Verfügbarkeit des Systems zu erhöhen. Als Integration bietet die Applikation eine Möglichkeit einen externen Bezahlungsdienst zu konfigurieren. Die Funktionen werden durch 21 API-Endpunkten bereitgestellt. Die Userverwaltung wurde mit vier APIs, der Katalog mit sechs APIs, der Warenkorb mit fünf APIs und der Bestellvorgang mit sechs APIs abgebildet.

4.2.1.2 Microservice Architektur

Der Onlineshop wurde mit einer Microservice Referenz-Architektur von Microsoft abgebildet. Die Applikation umfasst dieselben Module wie auch der Monolith, jedoch wurden die Funktionen in eigene Microservices aufgetrennt (Nish Anil, 2020). Die Bereitstellung erfolgt durch die Verwendung von Containern und deren Orchestrierung durch Kubernetes. Alle Services werden durch die Verwendung von managed Services von Amazon Web Services bereitgestellt. Die Datenbank ist eine relationale Datenbank, die für die Abstraktion der Bereitstellungmethode gemeinsam von allen Microservices genutzt wird. Die Container werden in AWS Fargate ausgeführt, die durch Amazon Elastic Kubernetes Service (Amazon EKS) orchestriert werden. Kubernetes verwaltet die einzelnen Instanzen und die gesamten Cluster (Amazon

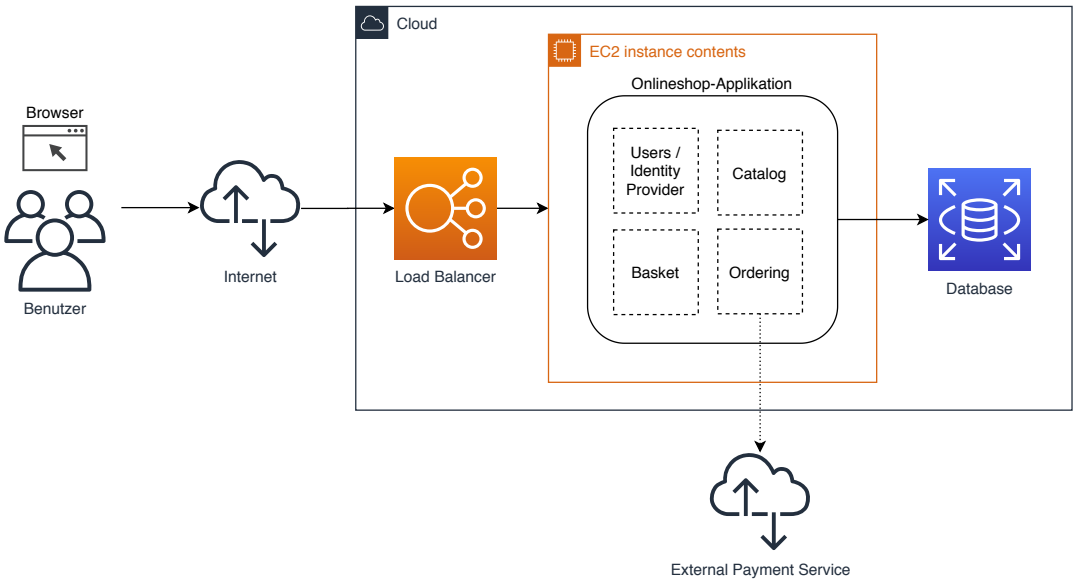

Abbildung 4.1: Monolithische Architektur eines einfachen Onlineshop, der in der Cloud bereitgestellt wird (Quelle: eigene Darstellung)

Web Services, 2019). Die einzelnen Services und APIs werden durch eine Application-Load-Balancer gruppiert und bereitgestellt. Das gesamte System besteht aus 21 APIs, die von vier Microservices bereitgestellt werden. Die einzelnen Service bieten APIs für die Funktionen an und ein monolithisches Webinterface wird über Amazon S3 als statische Webseite bereitgestellt. Die Inhaltsbereitstellung erfolgt mit Amazon CloudFront. Eine gesamte Übersicht dieser Architektur ist in der Abbildung 4.2 ersichtlich.

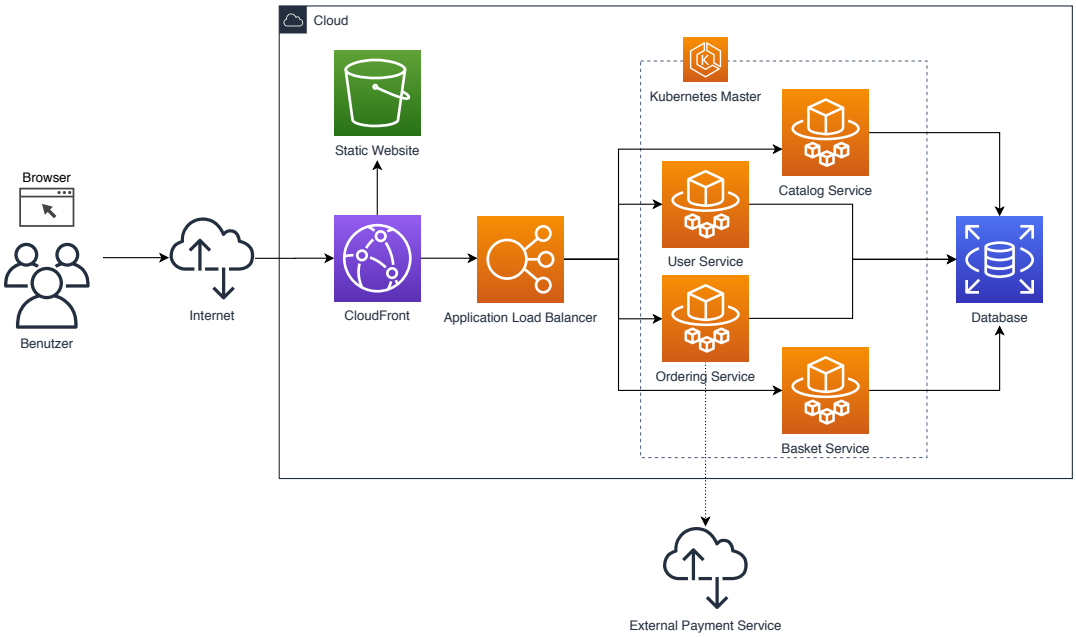

Abbildung 4.2: Microservice Architektur eines einfachen Onlineshop, der in der Cloud bereitgestellt wird (Quelle: eigene Darstellung)

4.2.1.3 Serverless-Architektur

Das Konzept für einen Onlineshop auf Basis von Serverless-Technologien wurde auf einer Referenz-Architektur einer E-Commerce Plattform von AWS erstellt. Das System nutzt Amazon S3 und Amazon CloudFront für die Bereitstellung einer statischen Website sowie einen Application-Load-Balancer für die Gruppierung der einzelnen Services, die mit einem API-Gateway sowie AWS Lambda abgebildet werden. Das System wurde mit einer Event-Driven Architektur abgebildet, bei der die Daten in einer Amazon DynamoDB abgebildet werden und die Events werden in Amazon EventBridge, einem Event Bus registriert. Anstelle einer eigenen Implementierung für die Userverwaltung, wird Amazon Cognito als Userverwaltung eingesetzt. Aktionen wie Registrierung eines neuen Users oder Login erzeugt einen Event, der über eine Amazon Lambda an den Event Bus gesendet wird. Der Event Bus wird zentral verwendet, bei dem einzelne Services auf neue Daten reagieren können oder eine Integration für bestimmte Events erfolgen kann, ohne eine weitere Implementierung in der Applikation vorsehen zu müssen. Die weiteren Funktionen wie der Produktkatalog, Warenkorb und der Bestellprozess bieten dieselben APIs wie mit einer Microservice oder monolithischen Architektur über einen API-Gateway an. Der API-Gateway ruft für die einzelnen Anfragen Functions auf, die die Daten aus der Datenbank lesen und schreiben. Anpassungen von Daten registrieren mit automatischer Hilfe einer weiteren Function einen Event im Event Bus. Das System stellt 17 APIs zur Verfügung, die aus 21 AWS Lambda Functions (Eine Function für Userevents, sieben Functions für den Produktkatalog, fünf Functions für den Warenkorb und acht Functions für den Bestellvorgang) bestehen. Die Architektur ist in der Abbildung 4.3 ersichtlich, die Functions wurden in einem Element als AWS Lambda gruppiert (Amazon Web Services, 2020).

4.2.2 One-Sided Plattform mit direkten Netzeffekten

Ein Onlineshop bildet direkte Netzeffekte ab, wenn ein Kunde davon profitiert je mehr User die Plattform nutzen. Direkte Netzeffekte werden durch ein Bewertungssystem der Produkte abgebildet, denn je mehr Nutzer über den Onlineshop bestellen, desto höher ist die Anzahl der Bewertungen, was wiederum den Onlineshop attraktiver macht, da Kunden, Informationen über die Qualität von Produkten von anderen Kunden bekommen. Die zuvor definierten Anforderungen von Abschnitt 4.2.1 werden damit ergänzt, dass ein Kunde nach der Bestellung eines Produktes die bestellten Produkte mit einer Wertung von 1-5 bewerten kann und dazu eine Rezession schreiben kann. Die Funktionen werden als REST-API bereitgestellt und der Onlineshop kann über ein Webinterface genutzt werden. Der Onlineshop erfordert die folgenden Anforderungen:

- Konto registrieren, anmelden und abmelden

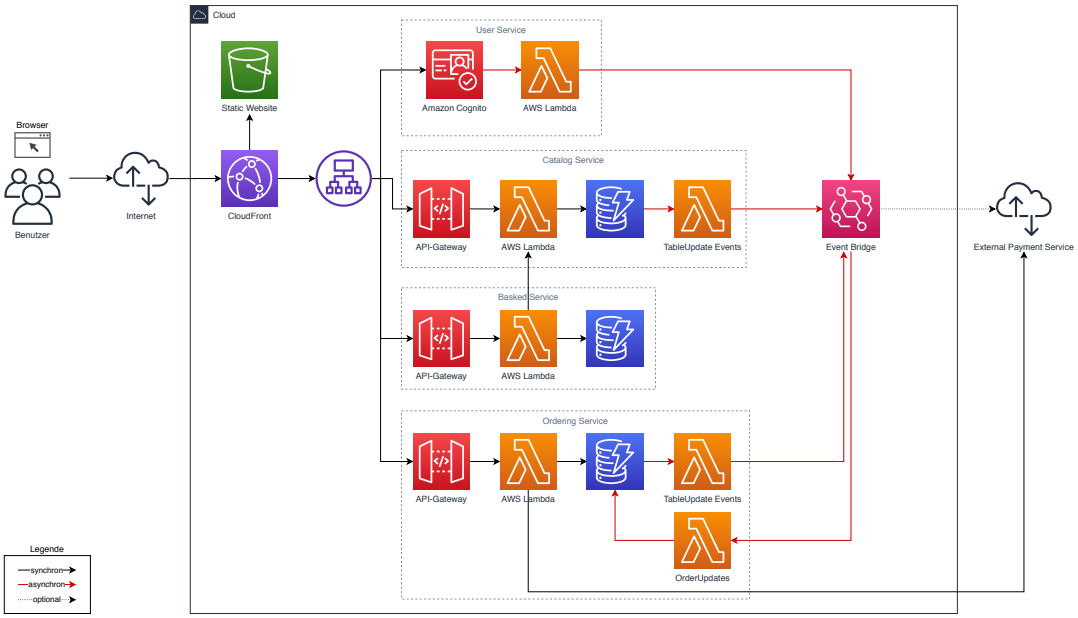

Abbildung 4.3: Serverless-Architektur eines einfachen Onlineshop, der in der Cloud bereitgestellt wird (Quelle: eigene Darstellung)

- Administrator kann Produkte hinzufügen, bearbeiten und löschen
- Produkte könnten aufgelistet und nach Typ oder Marke gefiltert werden
- User kann Produkte in den Warenkorb legen, die Anzahl der Produkte anpassen oder entfernen
- User kann die Produkte des Warenkorbs bestellen
- Produkte können nach der Bestellung bewerten werden
- Die Bewertung von Produkten wird allen Usern bereitgestellt
- User kann die Aufträge überprüfen
- Bei abgegeben Bestellungen kann ein externer Bezahlungsdienst eingebunden werden

4.2.2.1 Monolithische Architektur

Die Erweiterung des Onlineshops mit einer Bewertungsmöglichkeit von Produkten, die nach einer Bestellung ermöglicht wird, hat keine Auswirkung auf den Lösungs-

ansatz. Aus diesem Grund ist die ursprüngliche Architektur in Abschnitt 4.2.1.2 ersichtlich. Der Unterschied zur initialen Architektur ist, dass eine weitere API für Bewertung erstellt wurde.

4.2.2.2 Microservice Architektur

Produktbewertungen werden als Teil eines Produkts im Produktkatalog gespeichert. Die initiale Architektur aus dem Abschnitt 4.2.1.2 nutzte bereits ein separates Service für die Verwaltung des Produktkataloges. Die Funktionalität für die Bewertung wird aus diesem Grund mit einer weiteren API im Produktkatalog-Service implementiert. Die Anpassung hat somit keinen Einfluss auf die gesamte Architektur und deren Bereitstellung.

4.2.2.3 Serverless-Architektur

Die Serverless-Architektur mit einem Event-Driven Design verwendet einen zentralen Event-Bus für den Informationsaustausch von verschiedenen Aktionen im System. Eine Bewertung kann in unterschiedlichen Funktionsbereichen unterschiedlich genutzt werden, daher wird die Bewertung als neues Service implementiert. Wie in der Abbildung 4.4 ersichtlich ist, können Bewertung über einen weiteren API registriert werden, die im Event-Bus veröffentlicht werden und das Katalog-Service auf diesen Event registriert ist um die Bewertung bei dem jeweiligen bewerteten Produkt zu hinterlegen. Das System wurde mit zwei weiteren AWS Lambda Functions erweitert.

4.2.3 Multi-Sided Plattform mit indirekten Netzeffekten

Indirekte Netzeffekte ergeben sich bei einer Multi-Sided Plattform durch die gemeinsame Nutzung einer Plattform von zwei oder mehreren Kundengruppen. Ein Onlineshop, der Unternehmen anbietet deren Produkte zu verkaufen, generiert indirekte Netzeffekte. Durch ein umfangreicheres Produktportfolio wird der Onlineshop für Kunden attraktiver und für Unternehmen wird der Onlineshop attraktiver je mehr Kunden den Shop nutzen. Die bereits genannten Anforderungen werden weiter genutzt. Die Ergänzung des Onlineshops ist, dass Unternehmen eigene Produkte verwalten können und diese dem Kunden zur Bestellung verfügbar gemacht werden können. Eine Bestellung wird bei der Durchführung auf die jeweiligen Verkäufer aufgeteilt. Die folgenden Anforderungen müssen mit den bereits genannten Anforderungen aus Abschnitt 4.2.2 erfüllt werden:

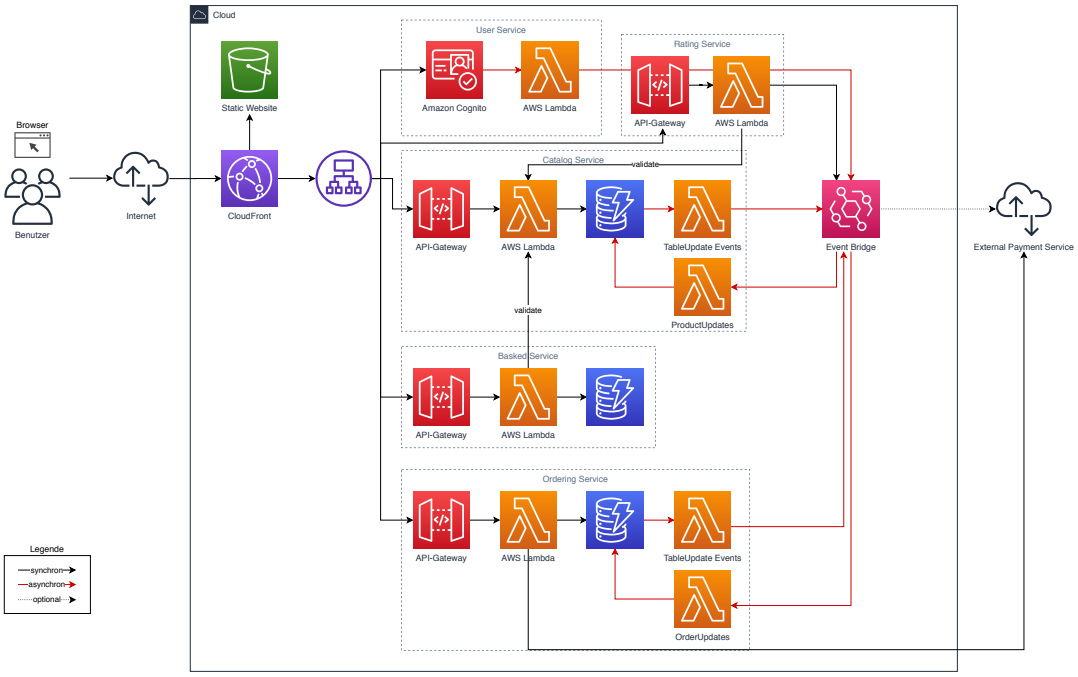

Abbildung 4.4: Serverless-Architektur für einen Onlineshop mit direkten Netzeffekten (Quelle: eigene Darstellung)

- User kann sich als Käufer oder Verkäufer registrieren
- Verkäufer haben nur Zugriff auf deren Produkte
- Verkäufer können neue Produkte registrieren, anpassen und entfernen
- Die Verfügbarkeit kann für jedes Produkt individuell erfasst werden
- Kunden können zwischen Produkten aller Lieferanten auswählen
- Verkäufer werden über die Bestellungen derer Produkte informiert
- Verkäufer können deren externes Bestellsystem an den Onlineshop anbinden

4.2.3.1 Monolithische Architektur

Die Anforderung eine weitere Zugangsmöglichkeit für Verkäufer zu ermöglichen, hat auf die monolithische Architektur nur geringe Auswirkungen. Zumindest auf einer höheren Abstraktionsebene wie in der Abbildung 4.1 gezeigt wird, hat die Erweiterung keine Auswirkung. Die Applikation wurde mit einem neuen Modul für die Verwaltung der Waren von Verkäufern erweitert. Die Anpassung stellt weitere APIs

zur Verfügung und das Userinterface wurde adaptiert, jedoch hat dies keine Auswirkung auf die Bereitstellung. Eine weitere Integration wurde ins Warehouse eingebaut, damit Verkäufer deren eigenes Bestellsystem integrieren können.

4.2.3.2 Microservice Architektur

Ein Two-Sided Marketplace erfordert eine Einschränkung auf einen Kontext für jedes Unternehmen, damit jedes Unternehmen nur deren Produkte und Bestellungen verwalten kann. Diese Einschränkung erfolgt auf Applikationsebene und ist somit in der Übersicht nicht ersichtlich. Die Verwaltung von unterschiedlichen Produkten pro Unternehmen sowie deren Warenstand und Bestellvorgang erfordert zusätzliche Logik. Um diese vom allgemeinen Produktkatalog und Bestellvorgang zu trennen wird ein weiteres Microservice für ein Warenhaus erstellt. Dieses Service bietet die Schnittstelle für die Verwaltung des Lagerbestands des Produktkataloges sowie die Abwicklung der unternehmensspezifischen Bestellungen. Das Service bietet zudem eine Möglichkeit, die Bestellung an ein externes System weiterzuleiten. Die Abbildung 4.5 zeigt die Erweiterung eines Microservices, welches mit einem weiteren Container bereitgestellt wird. Das Microservice stellt vier neue REST-Schnittstellen bereit.

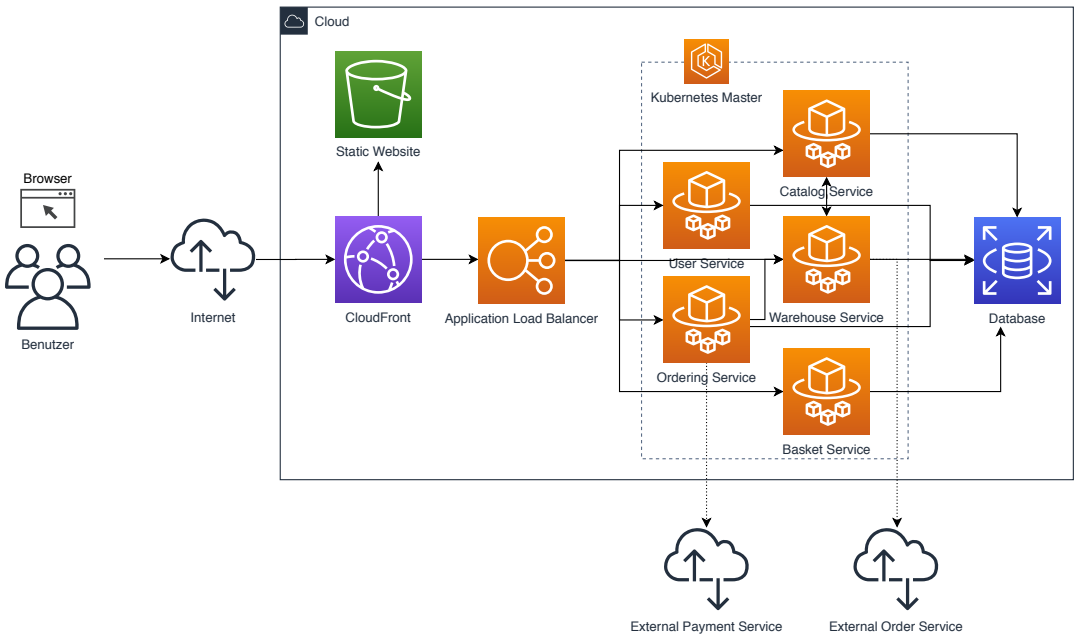

Abbildung 4.5: Two-Sided Marketplace mit einer Microservice Architektur (Quelle: eigene Darstellung)

4.2.3.3 Serverless-Architektur

Funktionsbereiche bei der Serverless-Architektur sind in einzelne Services gruppiert worden, bei der die einzelnen Functions einen unabhängigen Sourcecode haben und mithilfe eines API-Gateways veröffentlicht werden. Eine neue Funktionalität, wie Funktionalität für ein Warenhause werden somit separat entwickelt. Die Systemlandschaft wird für die Verwaltung des Lagerstands und der einzelnen Bestellungen für einen Verkäufer mit sechs weiteren AWS Lambda Functions und vier neuen APIs abgebildet. Die Abbildung 4.6 zeigt die gesamte Systemlandschaft. Die Anbindungen zwischen den einzelnen Funktionsbereichen wie dem Warenhaus, Bestellvorgang und Produktkatalog erfolgt über den zentralen Event-Bus. Der Event-Bus ermöglicht außerdem eine einfache Integration eines externen Bestellsystem, in dem auf den Event eine neue Bestellung für einen bestimmten Verkäufer registriert wird. Die Zugriffseinschränkung bzw. Events erfolgt auf Applikationsebene, dadurch hat die zusätzliche Komplexität keine direkte Auswirkung auf die gesamte abstrahierte Lösung.

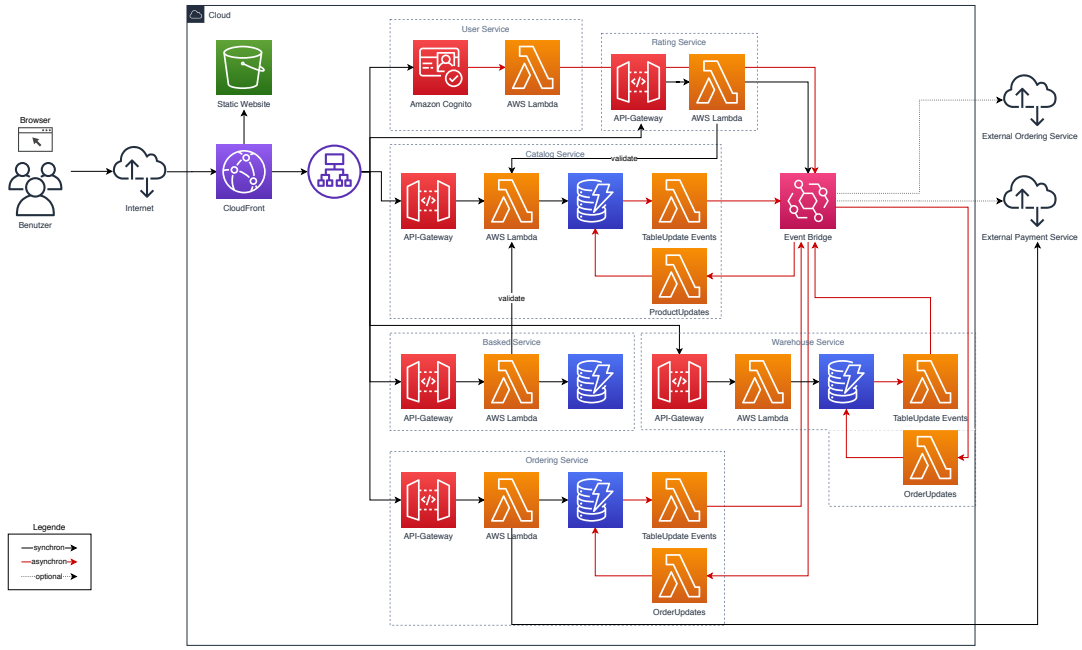

Abbildung 4.6: Two-Sided Marketplace mit einer Serverless-Architektur (Quelle: eigene Darstellung)

4.2.4 Multi-Sided Plattform mit direkten und indirekten Netzeffekten

Eine E-Commerce Plattform mit direkten und indirekten Netzeffekten wird bei einem Multi-Sided Onlineshop oder Auktionsplattform genutzt. Ein Onlineshop bei

dem einerseits Unternehmen Produkte verkaufen können und andererseits Privatkunden gebrauchte Produkte versteigern können. Je mehr User die Plattform nutzen, desto mehr steigt der Wert für den Kunden, da mehr User die angebotenen Produkte sehen und die Wahrscheinlichkeit, dass ein Produkt erfolgreich versteigert wird, erhöht sich. Die Plattform nutzt die bereits genannten Funktionalitäten eines herkömmlichen Onlineshops und erweitert diese mit einer Aktionsmöglichkeit. Die Funktion für eine Auktion ermöglicht eine Interaktion zwischen Usern. Der Funktionsumfang des Onlineshops mit indirekten Netzeffekten wird mit den folgenden Anforderungen ergänzt:

- Kunde kann eigene Produkte konfigurieren und bereitstellen
- Eine Auktion kann für ein Produkt gestartet, angepasst oder beendet werden
- Die vergangenen Auktionen können angesehen werden
- Kunden können Fragen zu einem Produkt stellen
- Der Bestellverlauf von ersteigerten Produkten kann vom Verkäufer gewartet werden

4.2.4.1 Monolithische Architektur

Die Erweiterung des Onlineshops mit einer Auktionsfunktion hat, wie bereits bei den vorherigen Anpassungen, keine Auswirkung auf die gesamte Solution und deren Bereitstellung. Die Funktionalität, um ein Produkt zur Auktion anbieten zu können oder für ein Produkt bieten zu können und dem Verkäufer Fragen zu stellen, wird über weitere API-Endpunkte bereitgestellt. Anpassungen einer monolithischen Architektur haben auf die Bereitstellung kaum Auswirkungen. Die Bereitstellung muss weitere API-Endpunkte erlauben und gegebenenfalls die benötigten Ressourcen für die Ausführung der Applikation erhöht werden.

4.2.4.2 Microservice Architektur

Die Microservice Architektur für den Onlineshop wurde konzipiert, damit neue Funktionen einfach erweitert werden können. Die Auktionsfunktionalität wird als ein neues Microservice entwickelt, um die bereits bestehende Funktionalität nicht anpassen zu müssen und damit die Functions getrennt voneinander bei Bedarf skaliert werden können. Auktion und Anfragen werden über sechs weitere APIs bereitgestellt. Das System umfasst somit nun 32 APIs die durch sechs Microservices in Container in einem Kubernetes Cluster bereitgestellt werden.

4.2.4.3 Serverless-Architektur

Eine Event-Driven Architektur ermöglicht eine sehr flexible Erweiterung des bestehenden Systems durch die Unabhängigkeit der einzelnen Komponenten. Die Abbildung 4.7 zeigt die Übersicht der gesamten Auktionsplattform. Die Informationen über die Auktion werden getrennt von den Bestellungen und dem Produktkatalog abgelegt. Bei neuen Auktionen oder abgeschlossenen Auktionen können die bestehenden Services auf Events, die durch die Auktion oder Fragen ausgelöst werden, reagieren. Um die Anforderungen erfüllen zu können, werden sechs neue APIs angeboten, die zehn neue AWS Lambda Functions für die Verarbeitung der Daten und Events benötigen. Die gesamte Serverless-Architektur stellt somit 28 APIs bereit, sowie die Userverwaltung durch Amazon Cognito und 39 AWS Lambda Functions für die Datenverarbeitung, die über einen Event-Bus kombiniert werden.

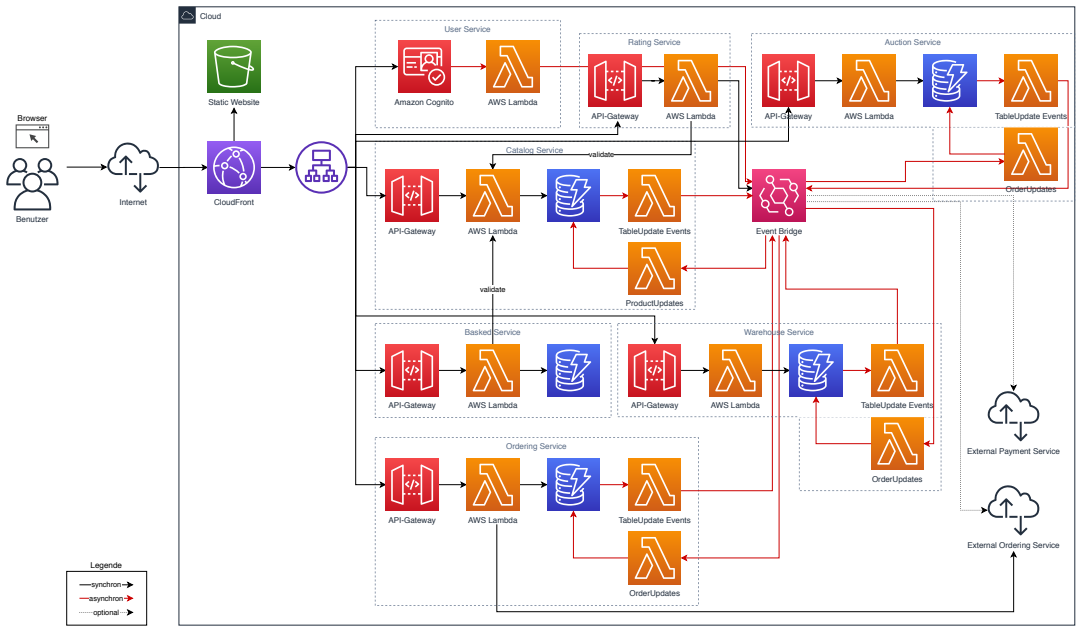

Abbildung 4.7: Auktionsplattform mit einer Serverless-Architektur (Quelle: eigene Darstellung)

4.3 Bereitstellungsmethoden

Monolithische, Microservice und Serverless-Architekturen unterstützen die gängigsten Programmiersprachen wie Java, Go, C#, Python, Node.js oder einige weitere. Der Unterschied ist, dass eine Monolithische Anwendung nur mit einer Programmiersprache entwickelt werden kann. Einzelne Microservices können mit unterschiedlichen Technologien entwickelt werden und bei Serverless-Functions könnte jede Function eine unterschiedliche Technologie nutzen. Die genannten Bereitstellungsmetho-

den im Abschnitt 4.2 können auch variiert werden, wie beispielsweise eine monolithische Applikationen kann in einer virtuellen Maschine oder einem Container betrieben werden, dasselbe gilt auch für Microservices. Container können in der Cloud einerseits als Managed Service genutzt werden, wie bei Amazon Web Services mit AWS Fargate, oder Container werden auf einer eigenen Instanz, wie Amazon EC2, gewartet. Eine Applikation aus Serverless-Function wird vorwiegend auf Serverless-Ressourcen, wie AWS Lambda, ausgeführt. Als Alternative zu einem Clouddienst werden auch Applikationen angeboten, in denen die Functions ausgeführt werden können, wie beispielsweise OpenFaas. OpenFaas bietet die Möglichkeit Serverless-Functions in einen Kubernetes Cluster bzw. Container auszuführen (Laszewski et al., 2018).

Die erstellten Architekturen aus dem Abschnitt 4.2 verwenden Clouddienste von Amazon Web Services. Eingesetzt wurden für die monolithische Architektur die Dienste Amazon Elastic Load Balancer, Amazon Elastic Compute Cloud und Amazon RDS für PostgreSQL. Die Microservice Architektur verwendet Amazon CloudFront, Amazon S3, Amazon Elastic Load Balancer, AWS EKS, AWS Fargate und Amazon RDS für PostgreSQL. Für die Bereitstellung durch Serverless-Technologien wurden Amazon CloudFront, Amazon S3, Amazon Elastic Load Balancer, AWS Lambda, Amazon API Gateway, Amazon EventBridge, Amazon Cognito und Amazon DynamoDB eingesetzt. Die Dimensionierung der einzelnen Dienste erfolgt für die Auswertung auf einem Vergleich der Architekturen der Los Andes University von 2017. Es wurde ein System mit denselben Funktionalitäten mit den drei Bereitstellungsmethoden ausgeführt, die Responsezeiten waren bei einem Monolithen und Serverless-Functions fast ident. Die Anfragen bei Microservices hatten eine etwas längere Ausführung (Villamizar et al., 2017). Aus diesem Grund werden für die Auswertung der Architektur folgenden Ressourcen verwendet:

- monolithische Architektur: 3x Amazon Elastic Compute Cloud (EC2) Instanzen vom Type c6g.xlarge (4 vCPUs, 8GiB RAM)

- Microservices: Services mit vielen Berechnungen werden auf einer Instanz mit zwei vCPUs und vier GB RAM betrieben, Services mit einem geringeren Aufwand auf einer Instanz mit einer vCPU und zwei GB RAM. Es muss beachtet werden, dass AWS Fargate bei einer CPU mindestens zwei GB RAM und bei zwei CPUs mindestens vier GB RAM erfordert.

- Serverless-Function: Jeder Function werden 512MB RAM zugewiesen.

5 Auswertung

Dieses Kapitel wertet die erstellten Architekturen von Kapitel 3 auf die definierten Kriterien von Kapitel 4 aus. Die vier Ausprägungen von digitalen Plattformen werden auf unterschiedlichen Cloudtechnologien bereitgestellt und die daraus folgenden Kosten werden für ein exemplarisches Szenario berechnet. Die Auswertung betrachtet ökonomische wie auch technische Kriterien, die am Ende zu einer Entscheidungsmatrix zusammengeführt werden.

5.1 Ergebnisse

Die erarbeiteten Szenarien von digitalen Plattformen wurden bereits im Abschnitt 4.2 erläutert und die Anforderungen in den verschiedenen Ausbaustufen mit einer monolithischen, Microservice oder Serverless-Architektur durch die Verwendung von Clouddiensten erläutert. Zudem wurden technische und ökonomische Bewertungskriterien im Abschnitt 4.1 definiert. Die unterschiedlichen Szenarien und Ausprägungen werden mit diesen Bewertungskriterien in diesem Abschnitt ausgewertet und betrachtet. Mögliche Gemeinsamkeiten oder Übereinstimmungen des Ergebnisses werden anschließend gruppiert.

5.1.1 Bewertung

Am Beginn werden die drei Architekturen in den vier Ausprägungen bewertet und verglichen. Der Vergleich der Kosten für die Bereitstellung wird anschließend im nächsten Abschnitt 5.1.2 erarbeitet. Die nachfolgende Bewertung betrachtet nichtfunktionale, ökonomische und technologische Aspekte für One-Sided und Multi-Sided Plattformen mit unterschiedlichen Netzeffekten.

Der Vergleich der unterschiedlichen Architekturen und Plattform Ausprägungen zeigte, dass sich die Bewertungskriterien, außer den Kosten, zwischen den unterschiedlichen Plattformen nicht unterscheiden. Aus diesem Grund wurde eine Bewertung der Aspekte im Allgemeinen für die Nutzung von einer monolithischen, Microservice und Serverless-Architektur für digitale Plattformen durchgeführt. In der Abbildung

5.1 ist das Ergebnis der drei Architekturen ersichtlich. Das Ergebnis ohne Berücksichtigung der Kosten und einer gleichen Gewichtung jedes Kriteriums, zeigt, dass die gesamte Bewertung für Serverless am besten abschneidet, gefolgt von Microservices und Monolithen. Es ist ersichtlich, dass sich die technologischen Aspekte in Summe fast ausgleichen. Beispielsweise eine Anwendung mit Serverless-Diensten benötigt zwar wenige zusätzliche Tools, ist jedoch sehr abhängig vom Cloud Provider und eine monolithische Applikation, benötigt mehrere Tools zur Betreibung der Anwendung, jedoch kann die Anwendung auch unabhängig von der Infrastruktur betrieben werden. Eine genauere Erläuterung zur Bewertung der jeweiligen Architektur folgt in den nächsten Absätzen.

	Bewertung		
Kriterium	**Monolithisch**	**Microservice**	**Serverless**
Nicht-Funktionale Aspekte			
Skalierbarkeit	2	4	5
Zuverlässigkeit(Reliability)	3	3	3
Erweiterbarkeit / Integrierbarkeit	2	3	4
Verfügbarkeit(Availability)	3	4	5
Upgradefähigkeit	3	4	4
Summe	**13**	**18**	**21**
Ökonomische Aspekte			
Fixkosten	2	3	4
Time-to-Market	2	3	4
Summe	**4**	**6**	**8**
Technologische Aspekte			
Sicherheit	3	4	4
Messbarkeit/Monitoring	3	3	4
Benötigte Tools	3	2	4
Portabilität	5	4	2
Summe	**14**	**13**	**14**
Gesamt	**31**	**37**	**43**

Abbildung 5.1: Bewertung der monolithischen, Microservice und Serverless-Architektur (Quelle: eigene Darstellung)

5.1.1.1 Monolithische Architektur

Die monolithische Architektur zeichnet sich durch den gesamtheitlichen Aufbau aus, in der alle Funktionalitäten der Applikation kombiniert werden. Die Bereitstellung der Applikation erfolgt in diesem Beispiel mit einer virtuellen Maschine. Der monolithische Aufbau bringt einerseits Vorteil bei der Interaktion der Funktionalitäten aber auch Nachteile beim Gesamtsystem. Die nachfolgende Auflistung erläutert die Bewertung aller Kriterien, die auch in der Abbildung 5.1 übersichtlich dargestellt werden:

- Skalierbarkeit: Der monolithische Aufbau schränkt die Skalierungsmöglichkeiten ein. Es kann nur die gesamte Applikation dupliziert werden und die Last auf den Instanzen verteilt werden (Horizontale Skalierung) oder die Ressourcen für eine Instanz werden erhöht (Vertikale Skalierung). Hat beispielsweise nur eine Teilfunktion der Applikation eine hohe Auslastung, kann die Skalierung nicht nur für diese Funktion erfolgen oder nur begrenzt, indem die Implementierung eine erhöhte Parallelisierung ermöglicht im Vergleich zu anderen Funktionen (De Santis, Florez, Nguyen, Rosa & Redbooks, 2016).

- Zuverlässigkeit (Reliability): Die Zuverlässigkeit von einer monolithischen Applikation hängt von der Implementierung ab. Einerseits können durch die gesamtheitliche Umsetzung in einem Service, Transaktionen einfach über die gesamte Ausführung einer Aktion verwaltetet werden und im Fehlerfall zurückgerollt werden. Andererseits kann ein Fehler in der Applikation, die gesamte Applikation zum Absturz bringen und der gesamte Funktionsbereich einer Anwendung von einer Instanz fällt aus (Ingeno, 2018).

- Erweiterbarkeit / Integrierbarkeit: Eine Anpassung oder Erweiterung des bestehenden Systems erfordert Anpassungen am gesamten System. Anpassungen für eine Funktion kann somit Auswirkungen auf andere Funktionen verursachen, die prinzipiell keinen Zusammenhang zur Anpassung haben. Dies erhöht das Risiko von Änderungen bei einem Monolithen und erhöht die Komplexität bei den Testausführungen. Die Integrationsmöglichkeiten müssen bei einem Monolithen genau definiert werden, wie beispielsweise durch die Verwendung von Business Rule Engines bei Prozessen, die anpassbar sein sollen oder durch die Bereitstellung von Events bei verschiedenen Operationen (Tiwana, 2013).

- Verfügbarkeit (Availability): Die Verfügbarkeit korreliert mit der Skalierbarkeit und Zuverlässigkeit. Wie bereits bei der Skalierbarkeit und Zuverlässigkeit erläutert, erschwert die monolithische Architektur eine redundante Auslegung der Infrastruktur. Eine hohe Verfügbarkeit benötigt eine große Anzahl an Ressourcen, da die gesamte Applikation dupliziert werden muss (De Santis et al., 2016).

- Upgradefähigkeit: Ein Upgrade kann ein Technologie Upgrade der Applikation sein, wie beispielsweise Framework Updates oder die Applikation wird mit einer neuen Version ausgerollt. Ein Update der Technologien kann bei einem Monolithen nur gesamtheitlich erfolgen, da dieselben Technologien in der gesamten Applikation genutzt werden. Die Abhängigkeit kann einen großen Mehraufwand bedeuten, da bei größeren Framework Upgrades, Anpassungen in mehreren Modulen der Applikation erforderlich sein können. Anpassungen an den Applikationen, sei es eine Fehlerkorrektur, Funktionserweiterung oder Technologie Update, kann zudem nur als ein gesamtes Release ausgeliefert werden. Eine umfangreiche Applikation hat ein höheres Fehlerpotential bei Anpassungen aber auch eine längere Deploymentzeit. Ein Upgrade der Applikation

ist somit unabhängig von der Größe der Anpassung, eine kleine Korrektur in einer Funktion bedeutet den gleichen Aufwand für ein Deployment wie eine große Erweiterung (Carneiro & Schmelmer, 2016).

- Fixkosten: Die Betreibung einer Infrastruktur mit virtuellen Maschinen erfordert für eine dauerhafte Bereitstellung der Applikation, eine Instanz die durchgehend läuft. Die Infrastruktur kann bei einem Leerlauf auf die Mindestanforderungen reduziert werden, jedoch entstehen auch für diese Ressource Kosten.

- Time-to-Market: Um eine Anpassung von einer monolithischen Applikation ausliefern zu können, ist ein hoher Aufwand und Komplexität notwendig, wie bereits bei der Erweiterbarkeit und Upgradefähigkeit erläutert. Die Abhängigkeit zwischen den Funktionen und der Umfang eines Monolithen machen es schwierig, eine Anwendung bereitzustellen. Es muss nur ein Service im Vergleich zu einer Microservice Architektur angepasst werden, jedoch ist die Anpassung komplexer und erschwert mehrere Änderungen parallel durchzuführen. Eine Anpassung kann die Umsetzung einer anderen Änderung erheblich beeinflussen. Der erhöhte Aufwand bei der Implementierung, beim Testen und der Bereitstellung wirkt sich auf eine lange Time-to-Market aus (Carneiro & Schmelmer, 2016).

- Sicherheit: Die Sicherheitsmaßnahmen können, wie auch bei allen anderen Kriterien, nur auf die gesamte Applikation angewendet werden. Die Einschränkungen bei der Applikation sind für die gesamte Applikation und können nicht auf einzelne Komponenten eingeschränkt werden, die Aufteilung erfolgt durch Autorisierung. Bei einer Sicherheitslücke im System kann dies dazu führen, dass ein Zugriff auf das gesamte System gewährt wird. Beispielsweise war ein Angriff wie SQL-Injektion oder ähnliche Attacken erfolgreich, könnten Daten des gesamten Systems zurückgegeben werden. Betrachtet man die Sicherheitslösungen für monolithische Systeme sind diese einfacher umzusetzen, da der Datenaustausch nicht über mehrere Netzwerkverbindung erfolgt, sondern nur innerhalb der Applikation und dem Client (Ingeno, 2018).

- Messbarkeit/Monitoring: Einen Vorteil bietet das Monitoring von einer monolithischen Applikation, da es nur ein Service gibt, erfolgt das Logging an einer zentralen Stelle. Funktionsabläufe erfolgen innerhalb einer Applikation, dies ermöglicht eine einfache Nachvollziehbarkeit von Logs und Funktionsaufrufen. Es wird jedoch erschwert, die jeweiligen Daten von einzelnen Funktionen zu trennen, wie beispielsweise benötigte Ressourcen oder die Last, die durch die einzelnen Funktionen entstehen.

- Benötigte Tools: Durch die Verwendung von virtuellen Maschinen zur Betreibung der Applikation erfordert die Ausführung der Applikation eine Laufzeitumgebung die manuell gewartet werden muss. Je nach eingesetzten Technologien, die bei der Umsetzung eingesetzt werden, müssen zusätzliche Tool ein-

gesetzt werden. Der einheitliche Aufbau der Applikation schränkt den Umfang des zusätzlichen Tools jedoch ein (De Santis et al., 2016).

- Portabilität: Die Ausführung eines Monolithen in einer virtuellen Maschine erfolgt nativ auf einem System, dies ermöglicht, dass die Anwendung auf jeglichen Systemen ohne große Einschränkungen ausgeführt werden kann. Meist gibt es nur eine Einschränkung auf die Mindestanforderungen, die für die Installation und die Ausführung gegeben sind (Klaffenbach et al., 2019).

5.1.1.2 Microservice Architektur

Eine Microservice Architektur ermöglicht durch die Aufteilung des Softwaresystems auf mehreren Services, eine Betreibung als verteiltes System. Im Vergleich zu einem monolithischen Aufbau bietet diese Architektur einige Vorteile, vor allem bei den nicht-funktionalen Anforderungen wie auch in der Übersicht in der Abbildung 5.1 ersichtlich ist. In dieser Bewertung wurden Microservices mit Container bereitgestellt auf Basis der Architektur für digitale Plattformen, die im Abschnitt 4.2 erläutert wurde.

- Skalierbarkeit: Die Aufteilung des gesamten Systems in mehreren unabhängigen Services ermöglicht eine feingranulare Skalierung. Die Skalierung kann für die einzelnen Funktionsbereiche bzw. Microservice je nach Bedarf erfolgen. Services mit einer hohen Auslastung können vertikal und horizontal skaliert werden.

- Zuverlässigkeit (Reliability): Im Vergleich zu einem Monolithen wirken sich Fehler im System nur auf Teilbereiche aus, wie beispielsweise einem Service. Jedoch erschwert der verteilte Aufbau des Systems die Umsetzung von Transaktionen oder die Handhabung von Aktionsausführungen im Fehlerfall. Die Architektur ermöglicht die Fehlerbehandlung unabhängig von der Implementierung des Service, wie beispielsweise, dass Requests, die fehlgeschlagen sind, erneut versucht werden.

- Erweiterbarkeit / Integrierbarkeit: Anpassungen eines Microservice Systems können unabhängig von anderen Services durchgeführt werden. Einzelne Services sollten aufgebaut werden, indem die Implementierung abwärtskompatibel ist. Anpassungen oder Erweiterungen des Systems können somit eingegrenzt umgesetzt werden, dies reduziert den Testaufwand und das Fehlerrisiko. Die Integrationsmöglichkeiten können ähnlich zu einem Monolithen umgesetzt werden, wobei die Trennung von Funktionsbereichen eine Prozessanpassung erleichtert. Anfragen zu anderen Services können mit einem Event-Driven-Design erfolgen oder über konfigurierbare Interfaces pro Mandanten erfolgen. Somit könnten sich die Ausführungen je nach Anwendern unterschiedlich verhalten, ohne die gesamte Applikation zu adaptieren.

- Verfügbarkeit (Availability): Die Verfügbarkeit wird durch Aufteilung auf mehreren Services und deren redundanten Aufteilung ermöglicht. Zudem unterstützt ein Traffic Management den Ablauf von Requests, in dem die Anfragen auf mehrere Services aufgeteilt werden können oder im Fehlerfall automatisch innerhalb des Systems erneut versucht werden.

- Upgradefähigkeit: Die Upgradefähigkeit hängt mit der Verfügbarkeit und der Erweiterbarkeit zusammen. Die Aufteilung des Systems und die Kompatibilität zu früheren Versionen ermöglicht, das Upgrades für einzelne Services durchgeführt werden. Dieses Vorgehen reduziert das Risiko von potentiellen Fehlern bei neuen Versionen. Die Kompatibilität von mehreren Versionen und dem Einsatz eines Traffic Managements ermöglichen zudem eine schrittweise Umstellung, in dem nur ein Teil der Requests auf die neue Version weitergeleitet werden. Im Fehlerfall wird auf die alte Version zurück gewechselt. Ein weiterer Vorteil ist die technologische Unabhängigkeit zwischen den Services, um Technologien unabhängig voneinander einzusetzen und Updates durchzuführen.

- Fixkosten: Eine Microservice Architektur kann durch die Aufteilung der Funktionalität auf mehreren Service bei einem Leerlauf die Instanzen auf die Mindestanforderungen reduzieren. Eine asynchrone Kommunikation zwischen den Services, wie beispielsweise mit Queues, ermöglicht, dass konsumierende Dienste teilweise komplett gestoppt werden können und erst bei Bedarf gestartet werden, um die Kosten zu reduzieren.

- Time-to-Market: Die Implementierung von neuen Funktionen kann unabhängig und parallel zu anderen Services erfolgen. Vordefinierte Schnittstellen zwischen Teilbereichen reduzieren somit die Durchlaufzeit der Umsetzung und darauf folgend die Time-to-Market. Außerdem kann das Deployment schrittweise erfolgen, wie bereits bei der Upgradefähigkeit erläutert wurde. Der Testaufwand und Deploymentaufwand hängt vom jeweiligen Service ab, das angepasst wurde. Im Vergleich zu einer monolithischen Architektur muss anstatt des gesamten Systems, bei einer Anpassung von nur einem Service, nur dieses Service bereitgestellt werden.

- Sicherheit: Ein Sicherheitssystem bei Microservices umzusetzen hat einen Mehraufwand im Vergleich zu einem Monolithen, durch die Umsetzung für mehrere Services. Die Aufteilung auf mehreren Services teilt das potenzielle Risiko auf die Services auf. Wird eine Sicherheitslücke bei einem Service gefunden, hat diese möglicherweise nur Auswirkungen auf dieses Service und nicht auf die gesamte Datenmenge. Die Sicherheitsmaßnahmen können zudem vom Service abstrahiert werden, wie beispielsweise bei einer Umsetzung bei der Netzwerkübertragung der Daten zwischen den Services. Je nach Umsetzung könnte beispielsweise ein Identitätsmanagement System den Zugriff auf einzelne API-Endpunkte bereits unterbinden bevor die Anfrage an das jeweilige Microservice gesendet wird.

- Messbarkeit/Monitoring: Die feingranulare Aufteilung der Funktionen über mehrere Services ermöglicht eine genaue Nachvollverfolgbarkeit und die benötigten Ressourcen für die jeweiligen Funktionsbereiche. Die Aufteilung erschwert jedoch das Monitoring der Services, da das Logging für jeden Service separat erfolgt und in irgendeiner Weise zentral zusammengeführt werden muss. Für die Nachvollziehbarkeit von Funktionsabläufen sollten Konzepte eingesetzt werden, um die Funktionsaufrufe von mehreren Services kombinieren zu können, wie beispielsweise durch die Verwendung einer Transaktions- oder Request-ID.

- Benötigte Tools: Die Verwendung von Containern erleichtert die Ausführung der Applikation, indem die Laufzeitumgebung bereits im Container vordefiniert wird und nicht am jeweiligen Zielgerät installiert werden muss. Die umfangreiche Architektur mit mehreren Services erfordert dennoch einen Mehraufwand, da die einzelnen Container orchestriert werden müssen und die Funktionen über mehrere Services vereint werden müssen. Zudem kann durch unterschiedliche Technologien bei verschiedenen Services zusätzliche Tools für die Entwicklung des jeweiligen Microservice benötigt werden.

- Portabilität: Container ermöglichen eine betriebssystem-unabhängige Betreibung der Applikationen. Die Kombination der einzelnen Services erhöht jedoch den Aufwand für die Betreibung des Softwaresystems (De Santis et al., 2016).

5.1.1.3 Serverless-Architektur

Die Serverless-Architektur verwendet ein Event-Driven-Design, wie auch bei den Beispielen im Kapitel 4. Es wurden Cloudfunctions und ein Event-Bus für die Kommunikation zwischen den Functions verwendet. Die Erläuterung zur Bewertung aus der Abbildung 5.1 ist in der folgenden Auflistung ersichtlich:

- Skalierbarkeit: Die Skalierung von einer Serverless-Architektur erfolgt durch den Cloudprovider. Es wird die Function definiert und der Zugriff verwaltet. Die Auslastung für die jeweiligen Functions wird automatisiert skaliert. Bei der Umsetzung muss somit auf die Skalierung nicht separat eingegangen werden. Die Last auf das System wird unbegrenzt automatisch ausgeglichen.

- Zuverlässigkeit (Reliability): Die Zuverlässigkeit des Systems und die Korrektheit der Daten werden bei einer Microservice Architektur getrennt von der jeweiligen Function betrachtet. Der Aufruf von einzelnen Functions muss in einem Fehlerfall den Aufruf erneut durchführen oder die ausgeführten Aktionen zurückrollen. Die konsumierten Events müssen bei kritischen Anforderungen überprüft werden, dass sie erfolgreich verarbeitet wurden.

- Erweiterbarkeit / Integrierbarkeit: Das gesamte System besteht aus einer Vielzahl an unabhängige Functions, die über direkte Aufrufe oder Events aufgerufen werden. Diese Aufteilung ermöglicht eine einfache Erweiterung des Systems, indem nur neue Function registriert werden oder bei Aufrufen eingebunden werden müssen. Anpassungen von einzelnen Functions, bei denen der Input und Output nicht verändert wird, können unabhängig vom gesamten System durchgeführt werden. Ein zentraler Event-Bus ermöglicht eine einfache Integration des Systems. Jegliche Änderung durch eine Function wird beim Event-Bus bekannt gegeben und externe Dienste oder Integrationen können auf diese Events reagieren. Zudem kann auf konfigurierbaren Regeln der Ablauf von Aktionen für Kunden adaptiert und erweitert werden.

- Verfügbarkeit (Availability): Die Verfügbarkeit wird wie bereits bei der Skalierung vom jeweiligen Cloudprovider übernommen. Die Verfügbarkeit wird in Service-Level-Agreements (SLA) definiert. Clouddienste wie auch Cloudfunctions haben, ohne eine zusätzliche Implementierung vornehmen zu müssen, eine hohe Verfügbarkeit.

- Upgradefähigkeit: Upgrades können bei einer Serverless-Architektur für jede Function unabhängig erfolgen. Die Architektur muss festlegen, welche Schnittstellen und Kommunikationswege zwischen Functions verwendet werden und auf Grund dieser Definition, sollten Abhängigkeiten der Functions ersichtlich werden. Wie bereits bei der Erweiterbarkeit erwähnt, können neue Anforderungen unabhängig von dem bestehenden System erfolgen, wenn beispielsweise nur neue Events entstehen oder eine neue Function auf einem bestehenden Event reagieren soll. Bei Anpassung einer Cloudfunction bei gleichbleibenden Input und Output Werten ist ein Update nur für diese eine Function notwendig.

- Fixkosten: Bei Cloudfunctions fallen die Kosten nur bei einer Ausführung an, somit gibt es keine Fixkosten für den Computing Dienst, wie auch bereits bei der Skalierbarkeit und Verfügbarkeit erwähnt. Datenspeicher und Datenbanken werden teilweise für die Datenablage von großen Dateien auch ohne Aufrufe verrechnet.

- Time-to-Market: Die Verwendung von bereitgestellten Diensten eines Cloudproviders ermöglicht eine rasche Umsetzung von Anforderungen, da keine eigene Infrastruktur gewartet werden muss. Die Implementierung fokussiert sich nur auf die Umsetzung der Anforderungen. Eine Trennung der Anforderungen und vordefinierten Schnittstellen und Events ermöglicht zudem eine Entwicklung, die gut parallelisiert werden kann, um die Time-to-Market noch weiter zu verringern.

- Sicherheit: Die Sicherheitsmaßnahmen für Serverless Softwaresysteme sind vergleichbar mit einer Microservice Architektur, da die Maßnahmen über mehrere Funktionen oder Services erfolgen müssen. Der Vorteil einer Serverless-

Architektur ist, dass die Cloudprovider meist Dienste für die Unterstützung der Sicherheit bereitstellen, um die Authentifizierung und Autorisierung zu erleichtern. Die Function muss auf herkömmliche Angriffsmöglichkeiten achten, wie beispielsweise Code-Injektion oder SQL-Injektion.

- Messbarkeit/Monitoring: Die Überwachung der Functions und anderen Diensten erfolgt durch den Cloudprovider. Die Informationen von den Diensten werden zentral abgelegt, jedoch wird die Übersicht der Daten durch die große Anzahl an unterschiedlichen Diensten erschwert. Funktionen sollten auf Nachverfolgbarkeit bei Loginformation achten, um bei einem Fehlerfall den Ablauf der Functions nachverfolgen zu können.

- Benötigte Tools: Für die Ausführung und Entwicklung von Serverless Applikationen stellen die meisten Cloudprovider ein einheitliches Tool zur Verfügung. Optional können weitere Tools eingesetzt werden, um die Entwicklung zu vereinfachen.

- Portabilität: Die Implementierung von Cloudfunctions ist meist für die Betreibung eines spezifischen Anbieters, somit ist die Portabilität eingeschränkt. Der Aufbau von Cloudfunctions ist jedoch bei den meisten Anbietern vergleichbar. Zudem gibt es Alternativen wie OpenFaaS, wenn kein proprietärer Dienst eingesetzt werden soll, wobei dadurch die Vorteile durch die uneingeschränkte Verwendung und Skalierung verloren gehen, die meisten alternativen Services müssen manuell gewartet werden, wie in diesem Fall mit Kubernetes (Bass, 2019).

5.1.2 Kostenübersicht

Die Kostenberechnung erfolgt für die vier Ausprägungen von digitalen Plattformen und jeweils als Monolith, Microservice und Serverless-Architektur. Die Bereitstellung wird mit Amazon Web Services in der Region Europa/Frankfurt durchgeführt. Die Systeme werden mit einer Last von 1000 aktiven Usern angenommen, die pro Minute 500 Requests oder 900.000 Requests pro Monat absenden. Die durchschnittliche Request Größe ist 20KB. Aus diesem Grund wird, wie bereits im Abschnitt 4.3 erläutert, für die Bereitstellung der monolithischen Architektur EC2 Instanzen verwendet, Microservices werden mit dem Serverless-Container-Dienst AWS Fargate bereitgestellt sowie die Serverless-Architektur verwendet AWS Lambda für die Cloudfunctions. Die angegebenen Preise enthalten keine Mehrwertsteuer und wurden am 18.11.2020 von aws.amazon.com abgerufen.

One-Sided Plattform ohne Netzeffekte

Die Architektur für eine One-Sided Plattform ohne Netzeffekte wurde im Abschnitt 4.2.1 erläutert. Diese Ausprägung definierte die Grundlage für die Erweiterungen der Plattform und die Verwendung einer Plattform für unterschiedliche Geschäftsmodelle. Die Bereitstellung einer einfachen Plattform mit der Monolithen Architektur und einer redundanten Ausführung mit drei AWS EC2 (c6g.xlarge) Instanzen ergab einen monatlichen Preis von € 418,61. Die Details für die Berechnung sind in der Abbildung 5.2 ersichtlich. Die Kosten entstehen bei einer Vollauslastung, aber auch wenn das System nicht genutzt wird. Somit sind die Fixkosten für die Applikation relativ hoch. Das System könnte optimiert werden indem beispielsweise bei geringer Auslastung eine der drei Instanzen gestoppt wird und nur genutzt wird, wenn es die Last erfordert. Wie bereits bei der Bewertung im Abschnitt 5.1.1 erläutert, ist der Verwaltungsaufwand für die Orchestrierung der virtuellen Maschine relativ hoch.

Monolith

Anzahl	Service	Notiz	Preis	Einheit	Gesamt	Gesamt / Monat
3	EC2	c6g.xlarge(4vCPUs, 8GiB RAM)	0,1552	USD/h	0,4656	335,23
	Elastic Load Balancing					
1		Application Load Balancer-Stund	0,027	USD/h	0,027	19,44
0,6		LCU-Stunde	0,008	USD/h	0,0048	3,46
1	Amazon RDS	db.t3.medium	0,084	USD/h	0,084	60,48
					0,5814	**418,61**

Abbildung 5.2: Kostenberechnung für die Bereitstellung einer digitalen Plattform mit einer monolithischen Architektur durch die Verwendung von AWS Services (Quelle: eigene Darstellung)

Dieselben Anforderungen wurden in einer Microservice Architektur umgesetzt und mit Serverless Container von AWS Fargate bereitgestellt. Alternativ könnten die Container manuell auf einer EC2 Instanz betrieben werden, wobei dies einen erhöhten Verwaltungsaufwand bedeutet. Die Verwendung von Amazon EKS in Kombination mit AWS Fargate ermöglicht eine einfache Ausführung und uneingeschränkte Skalierung des Systems. Ein vergleichbare Microservice Infrastruktur zu der monolithischen Architektur würde monatliche Kosten von € 698,28 ergeben. Alle Microservices wurden redundant ausgelegt. In diesem Beispiel wurde das Service für den Produktkatalog und den Warenkorb mit Instanzen mit jeweils zwei vCPUs und vier GB RAM betreiben, sowie das User- und Ordering-Service mit einer vCPU und zwei GB RAM. Für die Berechnung der Kosten für das Webinterface wurde angenommen, dass die statische Webseite eine Größe von 500KB hat und die Webseite 500-mal pro Stunde geladen wird. Dies ergibt gesamt 12 vCPUs und 24 GB RAM für die Computing Instanzen. Eine genaue Aufschlüsselung der Kosten ist in der Abbildung 5.3 ersichtlich. Wie bereits bei den EC2 Instanzen, entstehen bei der Microservice Architektur die Kosten für die Instanzen auch wenn kein Benutzer das System verwendet. Die Kosten können reduziert werden, in dem die zugewiesenen Ressourcen reduziert werden oder die Replizierung von den Containern in Kubernetes reduziert wird.

Die kostengünstigste Variante mit der exemplarischen Auslastung von 500 Requests pro Minute bietet die Serverless-Architektur, mit monatlichen Kosten von € 307,96.

Microservice

Anzahl	Service	Notiz	Preis	Einheit	Gesamt	Gesamt / Monat
	AWS Fargate					
12	4x2, 4x1 vCPU	pro vCPU pro Stunde	0,04656	USD/h	0,55872	402,28
24	4x4, 4x2 GB RAM	pro GB pro Stunde	0,00511	USD/h	0,12264	88,30
1	Amazon EKS		0,1	USD/h	0,1	72,00
	Elastic Load Balancing					
1		Application Load Balancer-Stund	0,027	USD/h	0,027	19,44
0,6		LCU-Stunde	0,008	USD/h	0,0048	3,46
1	Amazon RDS	db.t3.medium	0,084	USD/h	0,084	60,48
0,85	CloudFront		0,085	USD/GB/h	0,07225	52,02
1	S3	GET pro 1 000 Anforderungen	0,00043	USD/h	0,00043	0,31
					0,96984	**698,28**

Abbildung 5.3: Kostenberechnung für die Bereitstellung einer digitalen Plattform mit einer Microservice Architektur durch die Verwendung von AWS Services (Quelle: eigene Darstellung)

Ein weiterer Vorteil dieser Architektur ist, dass es keine Fixkosten gibt. Somit Fallen keine Kosten an, wenn das System nicht genutzt wird. Die Berechnung verwendet eine durchschnittliche Ausführzeit pro AWS Lambda von 200ms und einen zugewiesenen Arbeitsspeicher von 512 MB. Jeder Request führt im Durchschnitt drei Serverless-Functions aus. Die Kosten könnten noch weiter optimiert werden, indem überprüft wird wie viel Arbeitsspeicher pro Function notwendig ist, wie beispielsweise eine Function zur Registrierung eines Events wird nicht so viel Arbeitsspeicher benötigen wie eine Function für die Berechnung des Warenkorbs.

Serverless

Anzahl	Service	Notiz	Preis	Einheit	Gesamt / h	Gesamt / Monat
1000	Amazon Cognito		0,0055	USD/Monat	5,5	5,50
	AWS Lambda					
0,09		pro 1 Mio. Anforderungen	0,2	USD/h	0,018	12,96
9000		für jede GB-Sekunde	1,66667E-05	USD/h	0,1500003	108,00
	Elastic Load Balancing					
1		Application Load Balancer-Stund	0,027	USD/h	0,027	19,44
0,6		LCU-Stunde	0,008	USD/h	0,0048	3,46
0,03	Amazon API Gateway	pro 1 Mio. Anfragen	3,7	USD/h	0,111	79,92
0	Amazon EventBridge	pro 1 Mio. Anfragen nicht AWS S	1	USD/h	0	0,00
	Amazon DynamoDB					
0		ab 25 GB	0,306	USD / GB / M	0	0,00
0,006		pro Million Schreibanforderungs	1,525	USD/h	0,00915	6,59
0,09		pro Million Leseanforderungsein	0,305	USD/h	0,02745	19,76
0,85	CloudFront		0,085	USD/GB/h	0,07225	52,02
1	S3	GET pro 1 000 Anforderungen	0,00043	USD/h	0,00043	0,31
					$ 0,4201	**$ 307,96**

Abbildung 5.4: Kostenberechnung für die Bereitstellung einer digitalen Plattform mit einer Serverless-Architektur durch die Verwendung von AWS Services (Quelle: eigene Darstellung)

One-Sided Plattform mit direkten Netzeffekte

Die Architektur zwischen den One-Sided Plattformen weicht nur leicht ab. Die Erweiterungen bei dem Monolithen und Microservices wirken sich auf die Bereitstellung

nicht aus. Die neue Funktion wird über weitere APIs bereitgestellt. Es wird angenommen, dass das Userverhalten auf die Funktionen adaptiert wird und somit die Requests pro Minute unverändert bleiben. Die Serverless-Architektur wird mit weiteren Functions ergänzt und durch die gleichbleibende Anzahl an Requests und da keine Fixkosten anfallen, gibt es auch bei der Serverless Lösung keine Anpassungen der Berechnung. Dabei ist zu beachten, dass die verwendeten Ressourcen nicht auf die erhöhte Komplexität und dem erweitertem Funktionsumfang der monolithischen und Microservice Applikation abgeglichen wurden. Eine Erweiterung von bestehenden Services kann bei umfangreichen Anpassungen somit Auswirkungen auf die benötigte Infrastruktur haben.

Multi-Sided Plattform mit indirekten Netzeffekte

Die zwei Sichten der Plattform erhöhen die Komplexität der Architektur, dadurch hat dies auch Auswirkungen auf den Aufbau der Microservices und der Functions. Der monolithische Aufbau bleibt beim Grundaufbau bei allen Ausprägungen unverändert. Die Erläuterung der Architektur im Abschnitt 4.2 zeigt, dass bei der Microservice Architektur ein weiteres Service hinzugefügt wurde. Das neue Warehouse Service wird redundant mit einem Container mit einer vCPU und zwei GB RAM betrieben. Aus diesem Grund erhöhen sich die monatlichen Kosten für die Microservice Architektur auf € 780,05 bei denselben Aufrufen pro Minute. Die Serverless-Architektur erfordert für die Ergänzung einer weiteren Sicht, sechs neue AWS Lambda Functions und durch die Komplexität des Systems wird die durchschnittliche Anzahl an Funktionsaufrufen pro Anfrage auf vier erhöht. Die Anpassungen ergeben monatlichen Kosten von € 354,87. Die zunehmende Komplexität zeigt beim Vergleich der Kosten, dass der Abstand zwischen unterschiedlichen Methoden abnimmt, wobei die zunehmende Komplexität bei der Bereitstellung des Monolithen nicht eingepreist wurde.

Multi-Sided Plattform mit direkten und indirekten Netzeffekte

Wie bereits bei den beiden Ausprägungen der One-Sided Plattform, hat der Unterschied zwischen beiden Multi-Sided Plattformen keine Auswirkung auf die Kosten der ausgeführten Applikation. Die Bereitstellung verändert sich nur für die Serverless Umsetzung durch weiteren Functions, wobei die Anzahl der aufgerufenen Functions pro Request nicht verändert wird. Eine umfangreiche Erweiterung der Plattform durch die Interaktion von mehreren Kundengruppen kann den Ressourcenbedarf einer Applikation mit gleichbleibender Anzahl an Requests erhöhen. Der komplexere Aufbau der Applikation kann dazu führen, dass die jeweiligen Anfragen eine größere Datenmenge verarbeiten oder überprüfen müssen.

Kostenvergleich

Ein Vergleich der Kosten von One-Sided Plattformen zeigt, dass bei einer gleichbleibenden Infrastruktur und der Annahme, dass die Kosten für eine monolithischen Architektur und einer Serverless-Architektur dieselben sind, der Break-Even bei 698 Requests pro Minute liegt. Die Kosten für die Betreibung einer Plattform, die mit 698 Requests pro Minute genutzt wird und die zuvor definierten Ressourcen verwendet, liegen bei € 419,91 pro Monat. Dies vergleicht hypothetisch die Kosten für eine Serverless-Architektur und einem Monolithen mit virtuellen Maschinen. Die erhöhte Last auf das System müsste weiter analysiert werden ob die Anzahl der Requests tatsächlich mit der genutzten Infrastruktur ermöglicht werden kann. Derselbe Vergleich mit Cloud-Functions und Microservices würde einen Break-Even bei 1358 Requests pro Minute ergeben und zu monatlichen Kosten von € 793,- für die beiden Architekturen führen.

Ein weiterer Vergleich der Multi-Sided Plattformen zeigt, dass der Break-Even bei einer monolithischen und Serverless-Architektur bereits bei 598 Requests pro Minute erreicht wird. Bei einem Vergleich der Microservices und Cloud-Functions liegt der Break-Even bei 1275 Requests pro Minute und bei monatlichen Kosten von € 860,41. Eine genauere Aufschlüsselung der Kosten können vom Anhang entnommen werden. Wie bereits zuvor erwähnt, kann die erhöhte Nutzung der Plattform bei einem Monolithen und bei Microservices dazu führen, dass die Infrastruktur skaliert werden muss, um die Anfragen verarbeiten zu können. Der Vergleich zeigt, dass eine hohe Auslastung des Systems bei einer Serverless-Architektur zu höheren Kosten führt wie beispielsweise durch die Verwendung von Container oder virtuelle Maschinen.

5.2 Auswahlmodell

Der folgende Abschnitt kombiniert die erstellte Bewertung und die entscheidenden Ausprägungen von digitalen Plattformen. Die erarbeiten Information werden in Form einer Matrix kombiniert, um eine Auswahl für eine bestimmtes Vorgehensmodell für die Bereitstellung einer digitalen Plattform in der Cloud zu erleichtern.

Die Bewertung der Architekturen zeigte, dass sich die unterschiedlichen Architekturen nicht direkt auf die Plattform Ausprägungen auswirken. Die Unterschiede ergeben sich durch die Anpassung der Gewichtung der jeweiligen Kriterien. Die Gewichtung müsste auf die Anforderungen des Geschäftsmodell angepasst werden. Die Kostenbetrachtung zeigte, dass ein Vorteil der Serverless-Architektur ist, dass bei geringer Anzahl an Anfragen die Kosten sehr gering gehalten werden, da es keine Fixkosten für die Betreibung der Infrastruktur gibt. Ein weiterer Faktor, der bei der Kostenanalyse ersichtlich wurde, ist die zunehmende Komplexität eines Softwaresystem mit erhöhten Kosten bei der Serverless-Architektur. Die Aufteilung der Funktionen

in mehreren Services kann je nach Skalierung der einzelnen Services und geringer Auslastung zu Mehrkosten führen.

Aufgrund der Analyse aus dem Kapitel 3 wird zu den bereits bewerteten Kriterien die Multi-Tenancy und das SaaS Reifegradmodell betrachtet. Die Whitelabeling Möglichkeit ist bei der Beantwortung der Forschungsfrage entscheidend, aus diesem Grund hat die Integrationsmöglichkeit einen hohen Einfluss auf die Auswahlmöglichkeiten. Zudem haben die Geschäftsmodelle einen erheblichen Einfluss auf die Plattformen. Ein Geschäftsmodell das Einnahmen indirekt erzeugt, benötigt eine hohe Useranzahl auf der Plattform was wiederum entscheidend für die Kosten der Clouddienste ist. Die Kosten sollten nicht entscheidend für die Auswahl einer Architektur sein, da nicht nur die berechneten Kosten relevant sind, sondern auch die dazu benötigten Ressourcen für die Verwaltung einer Lösung, wie viel Aufwand es ist die Infrastruktur zu betreiben oder wie flexible die Architektur über einen längeren Zeitraum ist, betrachtet werden. Aus diesen Gründen wurde in der Auswahlmatrix in der Abbildung 5.5, die Kosten entnommen und mit den folgen Kriterien ersetzt. Die Kriterien basieren auf den zuvor genannten Auswirkungen auf die Plattformen und unterscheiden ob die Kriterien zutreffen oder nicht.

- Indirekte Finanzierung oder hohe Useranzahl: Erfolgt die Finanzierung der Plattform indirekt, benötigt sie eine hohe Anzahl an Usern, die die Plattform verwenden, um die Plattform gewinnbringend betreiben zu können. Niedrige Einnahmen pro User können auch dazu führen, dass die Architektur für eine große Anzahl an Usern konzipiert werden muss. Die große Last kann sich je nach eingesetzter Bereitstellungsmethode auf die Kosten für die Bereitstellung auswirken. Eine Microservice Architektur bietet Vorteile bei einer Vielzahl an Anfragen. Cloud-Functions bieten einen größeren Vorteil bei niedriger Auslastung, durch die nicht vorhandenen Fixkosten, wie bereits bei der Bewertung erläutert.

- Gleichmäßige Aufteilung der Last über die Berechnungsperiode: Ein System, dass über das gesamte Monat oder eine bestimmte Betrachtungsperiode, eine gleichmäßige Verteilung der Anfragen hat, erleichtert das Sizing der Infrastruktur im Vergleich zu einem System, das nur unregelmäßig Lastspitzen verursacht. Ein System mit gleichbleibenden Anforderungen kann vor allem bei einer großen Anzahl an Anfragen einen erheblichen Kostenvorteil bringen, wenn eine vordefinierte Instanz einer virtuellen Maschine oder Container benutzt werden, da die Kosten für eine Computing Instanz meist günstiger sind als eine große Anzahl an Ausführungen von Cloudfunctions. Ein System mit stark unterscheidender Last oder Lastspitzen für kurze Zeiträume, bietet Vorteile bei der Verwendung von Serverless-Diensten. Die Kosten fallen nur bei der Ausführung an. Im Vergleich dazu müssten Instanzen mit vielen zugewiesen Ressourcen dauerhaft betrieben werden obwohl die zusätzlichen Ressourcen nur für kurze Zeit benötigt werden.

- Datenseparierung oder einzelne Instanzen pro Mandanten: Plattformen mit Regulierungen können es erfordern, dass die Plattform physikalisch von anderen Kunden getrennt werden muss. White-Label Plattformen ohne Mandantenfähigkeit (SaaS Reifegrad 1) erfordern eine Bereitstellung des gesamten Systems unabhängig von anderen Kunden. Die beiden genannten Anforderungen ergeben je nach Größe des Systems und Architektur erhebliche Kostenunterschiede. Aus diesen Gründen wurde auch dieses Kriterium aufgeteilt, für eine Plattform mit der Verwendung von mehreren Mandanten (Bewertung für „nein“) in einer Installation oder einer getrennten Installation (Bewertung „ja“).

		Bewertung			
Kriterium		**Monolithisch**	**Microservice**	**Serverless**	**Gewichtung**
Nicht-Funktionale Aspekte					
Skalierbarkeit		2	4	5	(1 - 3)
Zuverlässigkeit(Reliability)		3	3	3	(1 - 3)
Erweiterbarkeit / Integrierbarkeit		2	3	4	(1 - 3)
Verfügbarkeit(Availability)		3	4	5	(1 - 3)
Upgradefähigkeit		3	4	4	(1 - 3)
Ökonomische Aspekte					
Fixkosten		2	3	4	(1 - 3)
Time-to-Market		2	3	4	(1 - 3)
Kosten*					
Technologische Aspekte					
Sicherheit		3	4	4	(1 - 3)
Messbarkeit/Monitoring		3	3	4	(1 - 3)
Benötigte Tools		3	2	4	(1 - 3)
Portabilität		5	4	2	(1 - 3)
***Kosten**					
Indirekte Finanzierung oder hohe Useranzahl	ja	3	4	2	(1 - 3)
	nein	3	2	4	
Gleichmäßige Aufteilung der Last über die Berechnungsperiode	ja	4	4	4	(1 - 3)
	nein	2	3	4	
Datenseparierung oder einzelne Instanzen pro Mandanten	ja	2	3	4	(1 - 3)
	nein	3	3	3	

Abbildung 5.5: Matrix für die Architekturauswahl (Quelle: eigene Darstellung)

Die Abbildung 5.5 stellt das Ergebnis der Auswertung da. Wie bereits am Beginn dieses Abschnitts erläutert wurde, ist die Gewichtung der Kriterien relevant für die Entscheidungsfindung anhand der erstellten Entscheidungsmatrix. Die Spalte „Gewichtung“ kann dazu genutzt werden, um die Gewichtung der Kriterien festzulegen. Exemplarisch wurde eine Gewichtung verwendet, bei der die Kriterien mit 1 bis 3 gewichtet werden. Eins bedeutet eine niedrige Relevanz, zwei ist neutral und drei bedeutet eine hohe Relevanz. Die Entscheidungsgrundlage ergibt sich indem die Bewertung mit der Gewichtung multipliziert wird und anschließend aufsummiert wird (Broy et al., 2013). Die Gewichtung kann auch auf anderen Vorgehensweisen adaptiert werden.

6 Fazit und Ausblick

Das Ziel dieser Arbeit war, mit Hilfe einer Analyse von Architekturen für digitale Plattformen als White-Label Lösung und deren Bereitstellung durch Clouddiensten, empfehlenswerte Auswahlmöglichkeiten für unterschiedliche Einsatzgebiete durch eine ökonomische und technische Perspektive zu untersuchen. Dies konnte durch eine Literaturrecherche und Auswertungen von Referenz-Architekturen erarbeitet und verifiziert werden.

Zu Beginn der Arbeit wurden relevante Definition und Grundlagen erarbeitet. Es wurden Grundlagen und Konzepte für die Softwarebereitstellung erläutert und auf die Vorgehensweisen der Softwareentwicklung eingegangen und wie sich diese über die letzten Jahre weiterentwickelt hat. Im Kapitel 3 wurde die Basis für den Entwurf von digitalen White-Label Plattformen erarbeitet. Zu Beginn wurden mögliche Ausprägungen der Plattform analysiert, die Aufteilung erfolgt zwischen verschiedenen Kundengruppen, Netzeffekten, die durch die Plattform Nutzung ermöglicht werden und eine Aufteilung nach Anpassungsmöglichkeiten. Die zweite Betrachtung der White-Label Lösungen erfolgt mit einem technischen Fokus. Es wurden Architektur Auswirkungen und Umsetzungsmöglichkeiten wie Mandantenfähigkeit, SaaS Datenmodelle oder ein SaaS Reifegradmodell analysiert. Ein weiterer Abschnitt der Analyse beschäftigte sich mit den Grundlagen von Clouddiensten. Eine Übersicht von möglichen Clouddiensten und Beispiele der drei umsatzstärksten Anbieter wurden erläutert. Die Bereitstellung von Plattformen in der Cloud wurde für Monolithische, Microservice und Serverless-Architekturen betrachtet. Die erarbeiten Informationen wurden im darauffolgenden Kapitel 4 für die unterschiedlichen Plattform Ausprägungen eingearbeitet. Es wurden relevante Kriterien für die Auswahl von Clouddiensten für die spätere Auswertung spezifiziert. Für die Auswertung wurden Beispiele für die unterschiedlichen Plattform Ausprägungen im Bereich von E-Commerce erarbeitet und eine Architektur in drei unterschiedlichen Architekturen konzipiert, die mit Clouddiensten von Amazon Web Services bereitgestellt werden können. Die erarbeiteten Kriterien und Konzepte wurden im darauffolgenden Kapitel 5 genauer betrachtet und ausgewertet. Die Auswertung betrachtete die technischen und ökonomischen Kriterien, um in einer erstellten Bewertungsmatrix mit einer Gewichtung der unterschiedlichen Kriterien die Entscheidung für eine Architektur und deren Bereitstellung mit Clouddiensten zu erleichtern. Die Auswertung zeigte, dass vor allem bei einer gelegentlichen oder variierenden Benutzung, Serverless-Dienste Vorteile bei der Betreibung von digitalen Plattformen und deren Integrationsmöglichkeiten bieten. Ein System mit einer kontinuierlichen hohen Auslastung kann

durch die Verwendung von Containern oder virtueller Maschinen günstiger betrieben werden. Die Auswahlentscheidung umfasst jedoch eine Vielzahl an unterschiedlichen Kriterien, die je nach Geschäftsmodell oder Einsatzgebiet entscheidend sein können. Aus diesem Grund muss die Gewichtung der Kriterien dementsprechend adaptiert werden, um eine aufschlussreiche Unterstützung für die Entscheidung bieten zu können. Das Ergebnis dieser Arbeit bietet diese Flexibilität und kann somit für unterschiedliche Plattform Konzepte adaptiert werden.

Diese Arbeit hat sich auf den Einsatz von Clouddiensten bei digitalen Plattformen auf Basis einer Literaturrecherche und bewerte Referenz-Architekturen bezogen. Eine weitere Evaluierung der erstellten Konzepte für die Skalierung und Performance bei unterschiedlichen Auslastungen der Plattformen kann mit Testungen von vorhandenen Plattformen unterstützt werden. Performancetests können beispielsweise Gewissheit bei den Auslastungsgrenzen und benötigten Ressourcen genutzt werden, um das Ergebnis zu evaluieren und potenzielle Optimierungen der Ressourcen zu erkennen. Eine Analyse der Integrationsmöglichkeiten könnte in weiterführenden Arbeiten betrachtet werden, zum Beispiel wie sich Clouddienste in der Integration verhalten und welche Kriterien für die Auswahl von Clouddiensten bei der Integration einer Software-as-a-Service Plattform relevant sind.

7 Anhang

7.1 Berechnung

7.1.1 One-Sided Plattform

AWS Dienste

Region	Europa (Frankfurt)	
Setup	1000	Aktive User
	500	Requests pro Minute
	900.000	Requests pro Monat
	20	Request Größe in KB

Monolith

Anzahl	Service	Notiz	Preis	Einheit	Gesamt	Gesamt / Monat
3	EC2	c6g.xlarge(4vCPUs, 8GiB RAM)	0,1552	USD/h	0,4656	335,23
	Elastic Load Balancing					
1		Application Load Balancer-Stund	0,027	USD/h	0,027	19,44
0,6		LCU-Stunde	0,008	USD/h	0,0048	3,46
1	Amazon RDS	db.t3.medium	0,084	USD/h	0,084	60,48
					0,5814	**418,61**

Microservice

Anzahl	Service	Notiz	Preis	Einheit	Gesamt	Gesamt / Monat
	AWS Fargate					
12	4x2, 4x1 vCPU	pro vCPU pro Stunde	0,04656	USD/h	0,55872	402,28
24	4x4, 4x2 GB RAM	pro GB pro Stunde	0,00511	USD/h	0,12264	88,30
1	Amazon EKS		0,1	USD/h	0,1	72,00
	Elastic Load Balancing					
1		Application Load Balancer-Stund	0,027	USD/h	0,027	19,44
0,6		LCU-Stunde	0,008	USD/h	0,0048	3,46
1	Amazon RDS	db.t3.medium	0,084	USD/h	0,084	60,48
0,85	CloudFront		0,085	USD/GB/h	0,07225	52,02
1	S3	GET pro 1 000 Anforderungen	0,00043	USD/h	0,00043	0,31
					0,96984	**698,28**

Serverless

Anzahl	Service	Notiz	Preis	Einheit	Gesamt / h	Gesamt / Monat
1000	Amazon Cognito		0,0055	USD/Monat	5,5	5,50
	AWS Lambda					
0,09		pro 1 Mio. Anforderungen	0,2	USD/h	0,018	12,96
9000		für jede GB-Sekunde	1,66667E-05	USD/h	0,1500003	108,00
	Elastic Load Balancing					
1		Application Load Balancer-Stund	0,027	USD/h	0,027	19,44
0,6		LCU-Stunde	0,008	USD/h	0,0048	3,46
0,03	Amazon API Gateway	pro 1 Mio. Anfragen	3,7	USD/h	0,111	79,92
0	Amazon EventBridge	pro 1 Mio. Anfragen nicht AWS S	1	USD/h	0	0,00
	Amazon DynamoDB					
0		ab 25 GB	0,306	USD / GB / M	0	0,00
0,006		pro Million Schreibanforderungs	1,525	USD/h	0,00915	6,59
0,09		pro Million Leseanforderungsein	0,305	USD/h	0,02745	19,76
0,85	CloudFront		0,085	USD/GB/h	0,07225	52,02
1	S3	GET pro 1 000 Anforderungen	0,00043	USD/h	0,00043	0,31
					$ 0,4201	**$ 307,96**

Details

Elastic Load Balancer

1 LCU	inkludierte Leistung:		
	25	neue Verbindungen pro Sekunde	
	3000	aktive Verbindungen pro Minute	
	1	GB pro Stunde für EC2-Instances	
	1000	Regelauswertungen pro Sekunde	

LCU

	21	APIs	
	1000	aktive Users	
0,08	2	Neue Verbindungen pro Sek	
0,16666667	500	aktive Verbindungen pro Minute	
0,6	0,6	GB pro h	=500 Verbindungen pro min * 60 * 20KB
0,056	56	Regelauswertungen pro Sekunde	

CloudFront

0,6	GB pro h - API	=500 Verbindungen pro min * 60 * 20KB
0,25	GB pro h - S3	= jeder User ladet in der Stunde ca. einmal die Seite mit 500KB
0,85		

S3	1	Einheiten pro 1000 Anfragen	=jeder User ladet die Seite einmal in der Stunde

Lambda 17 APIs zur Verfügung, die aus 21 AWS Lambda Funktionen bestehen

23	Lambda	
0,5	GB RAM	
200	ms durchschnittliche Ausführung	
0,09	pro 1 Mio. Anforderungen	=(500 Verbindungen pro Min *60 *3 Lambda pro Anfrage)/1000000
9000	für jede GB-Sekunde	

API-Gateway

0,03	1 Mio. Anfragen pro h	=500 Verbindungen pro Min * 60 / 1 Mio

Dynamo

0,006	pro Million Schreibanforderungs	=20% von 500 Verbindungen pro Minute * 60 / 1Mio
0,09	pro Million Leseanforderungsein	=500 Verbindungen pro Min * 60 * 3 Leseanfragen pro Anfrage / 1Mio

Vergleich Serverless und Monolith

AWS Dienste

Region	Europa (Frankfurt)	
Setup	1000	Aktive User
	698	Requests pro Minute
	1.256.400	Requests pro Monat
	20	Request Größe in KB

Monolith

Anzahl	Service	Notiz	Preis	Einheit	Gesamt	Gesamt / Monat
3	EC2	c6g.xlarge(4vCPUs, 8GiB RAM)	0,1552	USD/h	0,4656	335,23
	Elastic Load Balancing					
1		Application Load Balancer-Stund	0,027	USD/h	0,027	19,44
0,8376		LCU-Stunde	0,008	USD/h	0,0067008	4,82
1	Amazon RDS	db.t3.medium	0,084	USD/h	0,084	60,48
					0,5833008	**419,98**

Serverless

Anzahl	Service	Notiz	Preis	Einheit	Gesamt / h	Gesamt / Monat
1000	Amazon Cognito		0,0055	USD/Monat	5,5	5,50
	AWS Lambda					
0,12564		pro 1 Mio. Anforderungen	0,2	USD/h	0,025128	18,09
12564		für jede GB-Sekunde	1,66667E-05	USD/h	0,20940042	150,77
	Elastic Load Balancing					
1		Application Load Balancer-Stund	0,027	USD/h	0,027	19,44
0,8376		LCU-Stunde	0,008	USD/h	0,0067008	4,82
0,04188	Amazon API Gateway	pro 1 Mio. Anfragen	3,7	USD/h	0,154956	111,57
0	Amazon EventBridge	pro 1 Mio. Anfragen nicht AWS S	1	USD/h	0	0,00
	Amazon DynamoDB					
0		ab 25 GB	0,306	USD / GB / M	0	0,00
0,008376		pro Million Schreibanforderungs	1,525	USD/h	0,0127734	9,20
0,12564		pro Million Leseanforderungsein	0,305	USD/h	0,0383202	27,59
1,1866	CloudFront		0,085	USD/GB/h	0,100861	72,62
1	S3	GET pro 1 000 Anforderungen	0,00043	USD/h	0,00043	0,31
					$ 0,5756	**$ 419,91**

Details

Elastic Load Balancer

1 LCU	inkludierte Leistung:	
	25	neue Verbindungen pro Sekunde
	3000	aktive Verbindungen pro Minute
	1	GB pro Stunde für EC2-Instances
	1000	Regelauswertungen pro Sekunde

LCU

	21	APIs	
	1000	aktive Users	
0,08	2	Neue Verbindungen pro Sek	
0,16666667	698	aktive Verbindungen pro Minute	
0,8376	0,8376	GB pro h	=500 Verbindungen pro min * 60 * 20KB
0,056	56	Regelauswertungen pro Sekunde	

CloudFront

0,8376	GB pro h - API	=500 Verbindungen pro min * 60 * 20KB
0,349	GB pro h - S3	= jeder User ladet in der Stunde ca. einmal die Seite mit 500KB
1,1866		

S3	1	Einheiten pro 1000 Anfragen	=jeder User ladet die Seite einmal in der Stunde

Lambda 17 APIs zur Verfügung, die aus 21 AWS Lambda Funktionen bestehen

23	Lambda	
0,5	GB RAM	
200	ms durchschnittliche Ausführung	
0,12564	pro 1 Mio. Anforderungen	=(500 Verbindungen pro Min *60 *3 Lambda pro Anfrage)/1000000
12564	für jede GB-Sekunde	

API-Gateway

0,04188	1 Mio. Anfragen pro h	=500 Verbindungen pro Min * 60 / 1 Mio

Dynamo

0,008376	pro Million Schreibanforderungs	=20% von 500 Verbindungen pro Minute * 60 / 1Mio
0,12564	pro Million Leseanforderungsein	=500 Verbindungen pro Min * 60 * 3 Leseanfragen pro Anfrage / 1Mio

Vergleich Serverless und Microservices

AWS Dienste
Region Europa (Frankfurt)
Setup 1000 Aktive User
1358 Requests pro Minute
2.444.400 Requests pro Monat
20 Request Größe in KB

Microservice

Anzahl	Service	Notiz	Preis	Einheit	Gesamt	Gesamt / Monat
	AWS Fargate					
12	4x2, 4x1 vCPU	pro vCPU pro Stunde	0,04656	USD/h	0,55872	402,28
24	4x4, 4x2 GB RAM	pro GB pro Stunde	0,00511	USD/h	0,12264	88,30
1	Amazon EKS		0,1	USD/h	0,1	72,00
	Elastic Load Balancing					
1		Application Load Balancer-Stund	0,027	USD/h	0,027	19,44
1,6296		LCU-Stunde	0,008	USD/h	0,0130368	9,39
1	Amazon RDS	db.t3.medium	0,084	USD/h	0,084	60,48
2,3086	CloudFront		0,085	USD/GB/h	0,196231	141,29
1	S3	GET pro 1 000 Anforderungen	0,00043	USD/h	0,00043	0,31
					1,1020578	**793,48**

Serverless

Anzahl	Service	Notiz	Preis	Einheit	Gesamt / h	Gesamt / Monat
1000	Amazon Cognito		0,0055	USD/Monat	5,5	5,50
	AWS Lambda					
0,24444		pro 1 Mio. Anforderungen	0,2	USD/h	0,048888	35,20
24444		für jede GB-Sekunde	1,66667E-05	USD/h	0,40740081	293,33
	Elastic Load Balancing					
1		Application Load Balancer-Stund	0,027	USD/h	0,027	19,44
1,6296		LCU-Stunde	0,008	USD/h	0,0130368	9,39
0,08148	Amazon API Gateway	pro 1 Mio. Anfragen	3,7	USD/h	0,301476	217,06
0	Amazon EventBridge	pro 1 Mio. Anfragen nicht AWS S	1	USD/h	0	0,00
	Amazon DynamoDB					
0		ab 25 GB	0,306	USD / GB / M	0	0,00
0,016296		pro Million Schreibanforderungs	1,525	USD/h	0,0248514	17,89
0,24444		pro Million Leseanforderungsein	0,305	USD/h	0,0745542	53,68
2,3086	CloudFront		0,085	USD/GB/h	0,196231	141,29
1	S3	GET pro 1 000 Anforderungen	0,00043	USD/h	0,00043	0,31
					$ 1,0939	**$ 793,09**

Details

Elastic Load Balancer

1 LCU	inkludierte Leistung:		
	25	neue Verbindungen pro Sekunde	
	3000	aktive Verbindungen pro Minute	
	1	GB pro Stunde für EC2-Instances	
	1000	Regelauswertungen pro Sekunde	

LCU

	21	APIs	
	1000	aktive Users	
0,08	2	Neue Verbindungen pro Sek	
0,16666667	1358	aktive Verbindungen pro Minute	
1,6296	1,6296	GB pro h	=500 Verbindungen pro min * 60 * 20KB
0,056	56	Regelauswertungen pro Sekunde	

CloudFront

	1,6296	GB pro h - API	=500 Verbindungen pro min * 60 * 20KB
	0,679	GB pro h - S3	= jeder User ladet in der Stunde ca. einmal die Seite mit 500KB
	2,3086		

S3	1	Einheiten pro 1000 Anfragen	=jeder User ladet die Seite einmal in der Stunde

Lambda 17 APIs zur Verfügung, die aus 21 AWS Lambda Funktionen bestehen

	23	Lambda	
	0,5	GB RAM	
	200	ms durchschnittliche Ausführung	
	0,24444	pro 1 Mio. Anforderungen	=(500 Verbindungen pro Min *60 *3 Lambda pro Anfrage)/1000000
	24444	für jede GB-Sekunde	

API-Gateway

	0,08148	1 Mio. Anfragen pro h	=500 Verbindungen pro Min * 60 / 1 Mio

Dynamo

	0,016296	pro Million Schreibanforderungs	=20% von 500 Verbindungen pro Minute * 60 / 1Mio
	0,24444	pro Million Leseanforderungsein	=500 Verbindungen pro Min * 60 * 3 Leseanfragen pro Anfrage / 1Mio

7.1.2 Multi-Sided Plattform

AWS Dienste

Region	Europa (Frankfurt)	
Setup	1000	Aktive User
	500	Requests pro Minute
	900.000	Requests pro Monat
	20	Request Größe in KB

Monolith

Anzahl	Service	Notiz	Preis	Einheit	Gesamt	Gesamt / Monat
3	EC2	c6g.xlarge(4vCPUs, 8GiB RAM)	0,1552	USD/h	0,4656	335,23
	Elastic Load Balancing					
1		Application Load Balancer-Stund	0,027	USD/h	0,027	19,44
0,6		LCU-Stunde	0,008	USD/h	0,0048	3,46
1	Amazon RDS	db.t3.medium	0,084	USD/h	0,084	60,48
					0,5814	**418,61**

Microservice

Anzahl	Service	Notiz	Preis	Einheit	Gesamt	Gesamt / Monat
	AWS Fargate					
14	4x2, 6x1 vCPU	pro vCPU pro Stunde	0,04656	USD/h	0,65184	469,32
28	4x4, 6x2 GB RAM	pro GB pro Stunde	0,00511	USD/h	0,14308	103,02
1	Amazon EKS		0,1	USD/h	0,1	72,00
	Elastic Load Balancing					
1		Application Load Balancer-Stund	0,027	USD/h	0,027	19,44
0,6		LCU-Stunde	0,008	USD/h	0,0048	3,46
1	Amazon RDS	db.t3.medium	0,084	USD/h	0,084	60,48
0,85	CloudFront		0,085	USD/GB/h	0,07225	52,02
1	S3	GET pro 1 000 Anforderungen	0,00043	USD/h	0,00043	0,31
					1,0834	**780,05**

Serverless

Anzahl	Service	Notiz	Preis	Einheit	Gesamt / h	Gesamt / Monat
1000	Amazon Cognito		0,0055	USD/Monat	5,5	5,50
	AWS Lambda					
0,12		pro 1 Mio. Anforderungen	0,2	USD/h	0,024	17,28
12000		für jede GB-Sekunde	1,66667E-05	USD/h	0,2000004	144,00
	Elastic Load Balancing					
1		Application Load Balancer-Stund	0,027	USD/h	0,027	19,44
0,6		LCU-Stunde	0,008	USD/h	0,0048	3,46
0,03	Amazon API Gateway	pro 1 Mio. Anfragen	3,7	USD/h	0,111	79,92
0	Amazon EventBridge	pro 1 Mio. Anfragen nicht AWS S	1	USD/h	0	0,00
	Amazon DynamoDB					
0		ab 25 GB	0,306	USD / GB / M	0	0,00
0,006		pro Million Schreibanforderungs	1,525	USD/h	0,00915	6,59
0,12		pro Million Leseanforderungsein	0,305	USD/h	0,0366	26,35
0,85	CloudFront		0,085	USD/GB/h	0,07225	52,02
1	S3	GET pro 1 000 Anforderungen	0,00043	USD/h	0,00043	0,31
					$ 0,4852	**$ 354,87**

Details

Elastic Load Balancer

1 LCU	inkludierte Leistung:	
	25	neue Verbindungen pro Sekunde
	3000	aktive Verbindungen pro Minute
	1	GB pro Stunde für EC2-Instances
	1000	Regelauswertungen pro Sekunde

LCU

	21	APIs	
	1000	aktive Users	
0,08	2	Neue Verbindungen pro Sek	
0,16666667	500	aktive Verbindungen pro Minute	
0,6	0,6	GB pro h	=500 Verbindungen pro min * 60 * 20KB
0,056	56	Regelauswertungen pro Sekunde	

CloudFront

0,6	GB pro h - API	=500 Verbindungen pro min * 60 * 20KB
0,25	GB pro h - S3	= jeder User ladet in der Stunde ca. einmal die Seite mit 500KB
0,85		

S3	1	Einheiten pro 1000 Anfragen	=jeder User ladet die Seite einmal in der Stunde

Lambda 17 APIs zur Verfügung, die aus 21 AWS Lambda Funktionen bestehen

23	Lambda	
0,5	GB RAM	
200	ms durchschnittliche Ausführung	
0,12	pro 1 Mio. Anforderungen	=(500 Verbindungen pro Min *60 *4 Lambda pro Anfrage)/1000000
12000	für jede GB-Sekunde	

API-Gateway

0,03	1 Mio. Anfragen pro h	=500 Verbindungen pro Min * 60 / 1 Mio

Dynamo

0,006	pro Million Schreibanforderungs	=20% von 500 Verbindungen pro Minute * 60 / 1Mio
0,12	pro Million Leseanforderungsein	=500 Verbindungen pro Min * 60 * 3 Leseanfragen pro Anfrage / 1Mio

Vergleich Serverless und Monolith

AWS Dienste

Region	Europa (Frankfurt)	
Setup	1000	Aktive User
	1275	Requests pro Minute
	2.295.000	Requests pro Monat
	20	Request Größe in KB

Microservice

Anzahl	Service	Notiz	Preis	Einheit	Gesamt	Gesamt / Monat
	AWS Fargate					
14	4x2, 6x1 vCPU	pro vCPU pro Stunde	0,04656	USD/h	0,65184	469,32
28	4x4, 6x2 GB RAM	pro GB pro Stunde	0,00511	USD/h	0,14308	103,02
1	Amazon EKS		0,1	USD/h	0,1	72,00
	Elastic Load Balancing					
1		Application Load Balancer-Stund	0,027	USD/h	0,027	19,44
0,6		LCU-Stunde	0,008	USD/h	0,0048	3,46
1	Amazon RDS	db.t3.medium	0,084	USD/h	0,084	60,48
2,1675	CloudFront		0,085	USD/GB/h	0,1842375	132,65
1	S3	GET pro 1 000 Anforderungen	0,00043	USD/h	0,00043	0,31
					1,1953875	**860,68**

Serverless

Anzahl	Service	Notiz	Preis	Einheit	Gesamt / h	Gesamt / Monat
1000	Amazon Cognito		0,0055	USD/Monat	5,5	5,50
	AWS Lambda					
0,306		pro 1 Mio. Anforderungen	0,2	USD/h	0,0612	44,06
30600		für jede GB-Sekunde	1,66667E-05	USD/h	0,51000102	367,20
	Elastic Load Balancing					
1		Application Load Balancer-Stund	0,027	USD/h	0,027	19,44
0,6		LCU-Stunde	0,008	USD/h	0,0048	3,46
0,0765	Amazon API Gateway	pro 1 Mio. Anfragen	3,7	USD/h	0,28305	203,80
0	Amazon EventBridge	pro 1 Mio. Anfragen nicht AWS S	1	USD/h	0	0,00
	Amazon DynamoDB					
0		ab 25 GB	0,306	USD / GB / M	0	0,00
0,0153		pro Million Schreibanforderungs	1,525	USD/h	0,0233325	16,80
0,306		pro Million Leseanforderungsein	0,305	USD/h	0,09333	67,20
2,1675	CloudFront		0,085	USD/GB/h	0,1842375	132,65
1	S3	GET pro 1 000 Anforderungen	0,00043	USD/h	0,00043	0,31
					$ 1,1874	**$ 860,41**

Details

Elastic Load Balancer

1 LCU	inkludierte Leistung:		
	25	neue Verbindungen pro Sekunde	
	3000	aktive Verbindungen pro Minute	
	1	GB pro Stunde für EC2-Instances	
	1000	Regelauswertungen pro Sekunde	

LCU

	21	APIs	
	1000	aktive Users	
0,08	2	Neue Verbindungen pro Sek	
0,16666667	1275	aktive Verbindungen pro Minute	
0,6	1,53	GB pro h	=500 Verbindungen pro min * 60 * 20KB
0,056	56	Regelauswertungen pro Sekunde	

CloudFront

	1,53	GB pro h - API	=500 Verbindungen pro min * 60 * 20KB
	0,6375	GB pro h - S3	= jeder User ladet in der Stunde ca. einmal die Seite mit 500KB
	2,1675		

S3	1	Einheiten pro 1000 Anfragen	=jeder User ladet die Seite einmal in der Stunde

Lambda 17 APIs zur Verfügung, die aus 21 AWS Lambda Funktionen bestehen

	23	Lambda	
	0,5	GB RAM	
	200	ms durchschnittliche Ausführung	
	0,306	pro 1 Mio. Anforderungen	=(500 Verbindungen pro Min *60 *4 Lambda pro Anfrage)/1000000
	30600	für jede GB-Sekunde	

API-Gateway

	0,0765	1 Mio. Anfragen pro h	=500 Verbindungen pro Min * 60 / 1 Mio

Dynamo

	0,0153	pro Million Schreibanforderungs	=20% von 500 Verbindungen pro Minute * 60 / 1Mio
	0,306	pro Million Leseanforderungsein	=500 Verbindungen pro Min * 60 * 3 Leseanfragen pro Anfrage / 1Mio

Vergleich Serverless und Microservices

AWS Dienste

Region	Europa (Frankfurt)	
Setup	1000	Aktive User
	1275	Requests pro Minute
	2.295.000	Requests pro Monat
	20	Request Größe in KB

Microservice

Anzahl	Service	Notiz	Preis	Einheit	Gesamt	Gesamt / Monat
	AWS Fargate					
14	4x2, 6x1 vCPU	pro vCPU pro Stunde	0,04656	USD/h	0,65184	469,32
28	4x4, 6x2 GB RAM	pro GB pro Stunde	0,00511	USD/h	0,14308	103,02
1	Amazon EKS		0,1	USD/h	0,1	72,00
	Elastic Load Balancing					
1		Application Load Balancer-Stund	0,027	USD/h	0,027	19,44
0,6		LCU-Stunde	0,008	USD/h	0,0048	3,46
1	Amazon RDS	db.t3.medium	0,084	USD/h	0,084	60,48
2,1675	CloudFront		0,085	USD/GB/h	0,1842375	132,65
1	S3	GET pro 1 000 Anforderungen	0,00043	USD/h	0,00043	0,31
					1,1953875	**860,68**

Serverless

Anzahl	Service	Notiz	Preis	Einheit	Gesamt / h	Gesamt / Monat
1000	Amazon Cognito		0,0055	USD/Monat	5,5	5,50
	AWS Lambda					
0,306		pro 1 Mio. Anforderungen	0,2	USD/h	0,0612	44,06
30600		für jede GB-Sekunde	1,66667E-05	USD/h	0,51000102	367,20
	Elastic Load Balancing					
1		Application Load Balancer-Stund	0,027	USD/h	0,027	19,44
0,6		LCU-Stunde	0,008	USD/h	0,0048	3,46
0,0765	Amazon API Gateway	pro 1 Mio. Anfragen	3,7	USD/h	0,28305	203,80
0	Amazon EventBridge	pro 1 Mio. Anfragen nicht AWS S	1	USD/h	0	0,00
	Amazon DynamoDB					
0		ab 25 GB	0,306	USD / GB / M	0	0,00
0,0153		pro Million Schreibanforderungs	1,525	USD/h	0,0233325	16,80
0,306		pro Million Leseanforderungsein	0,305	USD/h	0,09333	67,20
2,1675	CloudFront		0,085	USD/GB/h	0,1842375	132,65
1	S3	GET pro 1 000 Anforderungen	0,00043	USD/h	0,00043	0,31
					$ 1,1874	**$ 860,41**

Details

Elastic Load Balancer

1 LCU	inkludierte Leistung:		
	25	neue Verbindungen pro Sekunde	
	3000	aktive Verbindungen pro Minute	
	1	GB pro Stunde für EC2-Instances	
	1000	Regelauswertungen pro Sekunde	

LCU

	21	APIs	
	1000	aktive Users	
0,08	2	Neue Verbindungen pro Sek	
0,16666667	1275	aktive Verbindungen pro Minute	
0,6	1,53	GB pro h	=500 Verbindungen pro min * 60 * 20KB
0,056	56	Regelauswertungen pro Sekunde	

CloudFront

	1,53	GB pro h - API	=500 Verbindungen pro min * 60 * 20KB
	0,6375	GB pro h - S3	= jeder User ladet in der Stunde ca. einmal die Seite mit 500KB
	2,1675		

S3	1	Einheiten pro 1000 Anfragen	=jeder User ladet die Seite einmal in der Stunde

Lambda 17 APIs zur Verfügung, die aus 21 AWS Lambda Funktionen bestehen

	23	Lambda	
	0,5	GB RAM	
	200	ms durchschnittliche Ausführung	
	0,306	pro 1 Mio. Anforderungen	=(500 Verbindungen pro Min *60 *4 Lambda pro Anfrage)/1000000
	30600	für jede GB-Sekunde	

API-Gateway

	0,0765	1 Mio. Anfragen pro h	=500 Verbindungen pro Min * 60 / 1 Mio

Dynamo

	0,0153	pro Million Schreibanforderungs	=20% von 500 Verbindungen pro Minute * 60 / 1Mio
	0,306	pro Million Leseanforderungsein	=500 Verbindungen pro Min * 60 * 3 Leseanfragen pro Anfrage / 1Mio

Abkürzungsverzeichnis

API	Application Programming Interface
AWS	Amazon Web Services
CD	Continuous Deployment
CI	Continuous Integration
CSMIC	Cloud Service Measurement Index Consortium
DBA	Database Administrator
DDoS	Distributed Denial of Service
EC2	Elastic Compute Cloud
FaaS	Function-as-a-Service
IaaS	Infrastructure-as-a-Service
IAM	Identity and Access Management
LCU	Load Balancer Capacity Units
PaaS	Platform-as-a-Service
RAM	Random-Access Memory
RDS	Relational Database Service
SaaS	Software-as-a-Service
SLA	Service Level Agreement
SQL	Structured Query Language
UI	Userinterface
vCPU	Virtual Central Processing Unit
VM	Virtuelle Maschine

Abbildungsverzeichnis

Literaturverzeichnis

Amazon Web Services. (2019, August). *Implementing microservices on aws.* Zugriff am 10. Nov. 2020 auf `https://d1.awsstatic.com/whitepapers/microservices-on-aws.pdf`

Amazon Web Services. (2020, Februar). *Aws serverless ecommerce platform.* Zugriff am 10. Nov. 2020 auf `https://github.com/aws-samples/aws-serverless-ecommerce-platform`

Bai, H. (2019). *Zen of cloud: Learning cloud computing by examples, second edition*. CRC Press.

Balzert, H. (2011). *Lehrbuch der softwaretechnik: Entwurf, implementierung, installation und betrieb*. Spektrum Akademischer Verlag.

Balzert, H. & Ebert, C. (2008). *Lehrbuch der softwaretechnik: Softwaremanagement*. Spektrum Akademischer Verlag.

Bangera, S. (2018). *Devops for serverless applications: Design, deploy, and monitor your serverless applications using devops practices*. Packt Publishing.

Bass, D. (2019). *Advanced serverless architectures with microsoft azure: Design complex serverless systems quickly with the scalability and benefits of azure*. Packt Publishing.

Bellemare, A. (2020). *Building event-driven microservices*. O'Reilly Media.

Broy, M., Pink, A., Kargl, E., Koßmann, H., Lagally, M. & Schimper, T. (2013). *Softwareentwicklung für kommunikationsnetze*. Springer Berlin Heidelberg.

Carneiro, C. & Schmelmer, T. (2016). *Microservices from day one: Build robust and scalable software from the start*. Apress.

Chong, F. & Carraro, G. (2006). Architecture strategies for catching the long tail.

De Santis, S., Florez, L., Nguyen, D., Rosa, E. & Redbooks, I. (2016). *Evolve the monolith to microservices with java and node*. IBM Redbooks.

Fowler, M. (2003). *Patterns für enterprise-application-architekturen*. mitp-Verlag.

García, V. & Biggs, J. (2019). *Building intelligent cloud applications: Develop scalable models using serverless architectures with azure*. O'Reilly Media, Incorporated.

Gawer, A. (2011). *Platforms, markets and innovation*. Edward Elgar.

Ingeno, J. (2018). *Software architect's handbook: Become a successful software architect by implementing effective architecture concepts*. Packt Publishing.

Ivanov, I., Sinderen, M., Leymann, F. & Shan, T. (2013). *Cloud computing and services science: Second international conference, closer 2012, porto, portugal, april 18-21, 2012. revised selected papers*. Springer International Publishing.

Kavis, M. (2014). *Architecting the cloud: Design decisions for cloud computing service models (saas, paas, and iaas)*. Wiley.

Kelly, A. (2012). *Business patterns for software developers*. Wiley.

Klaffenbach, F., Klein, M. & Sundaresan, S. (2019). *Multi-cloud for architects: Grow your it business by means of a multi-cloud strategy*. Packt Publishing.

Laszewski, T., Arora, K., Farr, E. & Zonooz, P. (2018). *Cloud native architectures: Design high-availability and cost-effective applications for the cloud*. Packt Publishing.

Mahmood, Z. & Hill, R. (2011). *Cloud computing for enterprise architectures*. Springer London.

Nish Anil. (2020). *Einführung in die eshoponcontainers-referenz-app*. Zugriff am 10. Nov. 2020 auf `https://docs.microsoft.com/de-de/dotnet/architecture/cloud-native/introduce-eshoponcontainers-reference-app`

Nooren, P., van Gorp, N., Eijk, N. & Fathaigh, R. (2018, 08). Should we regulate digital platforms? a new framework for evaluating policy options: Evaluating policy options for digital platforms. *Policy and Internet*, *10*.

OECD. (2019). *An introduction to online platforms and their role in the digital transformation*. OECD Publishing.

Pradeep, K. A., V., V. & Palaniswami, S. (2010). *Configurability in saas for an electronic contract management application*. WSEAS Press.

Prithviraj, S. (2015). *Architecting cloud saas software - solutions or products: Engineering multi-tenanted distributed architecture software*. Pearson.

Reznik, P., Dobson, J. & Gienow, M. (2019). *Cloud native transformation: Practical patterns for innovation*. O'Reilly Media.

Salam, A., Gilani, Z. & Haq, S. (2015). *Deploying and managing a cloud infrastructure: Real-world skills for the comptia cloud+ certification and beyond: Exam cv0-001*. Wiley.

Schubert, L., Jeffery, K. & Neidecker-Lutz, B. (2010). The future of cloud computing: Opportunities for european cloud computing beyond 2010. *Expert Group report, public version*, *1*.

Shrivastava, S., Srivastav, N. & Arora, K. (2020). *Solutions architect's handbook: Kickstart your solutions architect career by learning architecture design principles and strategies*. Packt Publishing.

Siegel, J. & Perdue, J. (2012). *Cloud services measures for global use: The service measurement index (smi)*.

Specker, A. (1998). *Kognitives software engineering: ein schema- und scriptbasierter ansatz*. vdf, Hochsch.-Verlag an der ETH.

Strader, T. (2010). *Digital product management, technology and practice: Interdisciplinary perspectives: Interdisciplinary perspectives*. Business Science Reference.

Tiwana, A. (2013). *Platform ecosystems: Aligning architecture, governance, and strategy*. Elsevier Science.

Villamizar, M., Garcés, O., Ochoa, L., Castro, H., Salamanca, L., Verano Merino, M., ... Lang, M. (2017, 04). *Cost comparison of running web applications in the cloud using monolithic, microservice, and aws lambda architectures* (Bd. 11). Zugriff am 14. Nov. 2020 auf `https://www.researchgate.net/publication/316532483_Cost_comparison_of_running_web_applications_in_the_cloud_using_monolithic_microservice_and_AWS_Lambda_architectures`

Wiegers, K. (2013). *Creating a software engineering culture*. Pearson Education.

Yablonsky, S. (2018). *Multi-sided platforms (msps) and sharing strategies in the digital economy: Emerging research and opportunities: Emerging research and opportunities*. IGI Global.