Analyse und Entwicklung cloudbasierter, asynchroner Multiplayer-Spielarchitekturen mit Amazon Web Services

Masterarbeit

für die Prüfung zum

Master of Arts

des Studiengangs Zeitabhängige Medien / Sound – Vision – Games

an der

Hochschule für Angewandte Wissenschaften Hamburg

von

Clemens Hübner

Abgabedatum 9. April 2025

Matrikelnummer [redacted]
Erstprüfer Prof. Dr. Eike Langbehn
Zweitprüferin M.A. Charlotte Knorr

Abstract

Cloud computing enables completely new architectural concepts for online games. This master's thesis empirically examines the use of Amazon Web Services for asynchronous multiplayer games and develops a guideline for optimal backend architectures based on analysis and practical examples. By comparing several architectural approaches, concrete recommendations are derived to help developers implement scalable and cost-efficient cloud backends for asynchronous multiplayer games.

Zusammenfassung

Cloud Computing ermöglicht völlig neue Architekturkonzepte für Online-Spiele. Die vorliegende Masterarbeit untersucht für asynchrone Multiplayer-Spiele empirisch den Einsatz von Amazon Web Services und entwickelt aus Analyse und Praxisbeispielen einen Leitfaden für optimale Backend-Architekturen. Durch den Vergleich mehrerer Architekturansätze werden konkrete Empfehlungen abgeleitet, die Entwicklern helfen sollen, skalierbare und kosteneffiziente Cloud-Backends für asynchrone Multiplayer-Spiele umzusetzen.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

A1 ..... Architektur 1: Klassische dreischichtige Architektur
A2 ..... Architektur 2: Serverlose Architektur
A3 ..... Architektur 3: Containerbasierte Architektur
A4 ..... Architektur 4: GameLift Architektur
ALB ..... Application Load Balancer
API ..... Application Programming Interface
AWS ..... Amazon Web Services
AZ ..... Availability Zone
BaaS ..... Backend as a Service
BSI ..... Bundesamt für Sicherheit in der Informationstechnik
CaaS ..... Container as a Service
CCU ..... Concurrent Users
CD ..... Continuous Delivery
CDN ..... Content Delivery Network
CI. ..... Continuous Integration
CLB ..... Classic Load Balancer
CPU ..... Central Processing Unit
DDD ..... Domain-Driven Design
DDoS ..... Distributed Denial-of-Service
DNS ..... Domain Name System
DTO ..... Data Transfer Object
EC2 ..... Elastic Compute Cloud
ECS ..... Elastic Container Service
EDA ..... Event-Driven Architecture

EKS ..... Elastic Kubernetes Service
ELB ..... Elastic Load Balancing
FaaS ..... Function as a Service
FIFO ..... First In, First Out
HPA ..... Horizontal Pod Autoscaler
HTTP ..... Hypertext Transfer Protocol
HTTPS ..... Hypertext Transfer Protocol Secure
IaaS ..... Infrastructure as a Service
IaC ..... Infrastructure as Code
IEEE ..... Institute of Electrical and Electronics Engineers
IoT ..... Internet of Things
IP. ..... Internet Protokoll
IT ..... Informationstechnologie
MOBA ..... Multiplayer Online Battle Arena
MQTT ..... Message Queuing Telemetry Transport
MSK ..... Managed Streaming für Apache Kafka
NAT ..... Network Address Translation
NLB ..... Network Load Balancer
OSI ..... Open Systems Interconnection
PaaS ..... Platform as a Service
RAM ..... Random Access Memory
RDS ..... Relational Database Service
REST ..... Representational State Transfer
SaaS ..... Software as a Service
SAD ..... Software Architecture Document
SDK ..... Software Development Kit
SMS ..... Short Message Service
SNS ..... Simple Notification Service
SPOT ..... Single Point of Truth
SQL ..... Structured Query Language
SQS ..... Simple Queue Service
SWOT ..... Strengths, Weaknesses, Opportunities, Threats
VPA ..... Vertical Pod Autoscaler
VPC ..... Virtual Private Cloud
WAF ..... Web Application Firewall

1 Einleitung

Die rasante Entwicklung und Popularität von Cloud-Technologien in Kombination mit Multiplayer-Spielen eröffnen neue, effiziente und kostengünstige Wege zur Gestaltung digitaler Spielerlebnisse. Immer mehr Entwickler, von kleinen Indie-Studios [1] bis hin zu großen Unternehmen, setzen auf cloudbasierte IT-Infrastrukturen, um ihre Online-Spielkonzepte schneller und flexibler umzusetzen. [2] In Kombination mit der Möglichkeit, Spieler durch asynchrone Mechaniken zeitlich zu entkoppeln, eröffnen sich viele interessante Ansätze, die mithilfe der vielfältigen Dienste des Cloud-Computing-Anbieters „Amazon Web Services“ (AWS) umgesetzt werden können.

Obwohl Cloud-Technologien und ihre Anwendung in der Spieleentwicklung kontinuierlich voranschreiten, fehlen bislang systematische Untersuchungen, die praxisnahe Handlungsempfehlungen für den Einsatz von AWS im Kontext asynchroner Multiplayer-Architekturen liefern.

Diese Arbeit leistet einen Beitrag, indem sie bestehende Ansätze empirisch evaluiert und daraus klare Architekturempfehlungen ableitet, die den verschiedenen Anforderungen asynchroner Multiplayer-Spiele gerecht werden.

[1] Indie-Studios: Kleine, unabhängige Entwicklerteams, die eigenständig und ohne Unterstützung großer Publisher Computerspiele realisieren. „Indie“ engl. Abk. für „Unabhängig“

[2] Vgl. Horan, Stephen: COVID-19 hat eine Gaming-Revolution gestartet und die Cloud wird den Prozess noch weiter beschleunigen: Von lokalen Spielen hin zu unendlicher Skalierbarkeit, in: Rackspace, 11.01.2021, [online] https://www.rackspace.com/de/blog/covid-kickstarted-gaming-revolution-and-now-cloud-will-accelerate-it (abgerufen am 18.12.2024).

Dabei werden nicht nur technische Parameter wie Skalierbarkeit betrachtet, sondern auch wirtschaftliche Aspekte und strategische Vorteile für Entwickler untersucht. Darüber hinaus bietet die Kombination aus theoretischer Analyse und praktischer Umsetzung einen Leitfaden, der als Orientierungshilfe für zukünftige Projekte dienen kann.

2 Zielsetzung

Die moderne Entwicklung von Online-Spielen steht vor der Herausforderung, den steigenden Anforderungen an Skalierbarkeit und Robustheit ihrer Architektur gerecht zu werden. Heutige Cloud-Technologien sind eine effiziente und kostengünstige Methode, diese Probleme zu adressieren.[3]

AWS, als führender Anbieter von Cloud-Infrastrukturen, bietet eine Vielzahl von Diensten, die es erlauben, effiziente Architekturen für asynchrone Multiplayer-Spiele zu entwickeln. Trotz des breiten Angebots an Technologien und Architekturmustern im AWS-Umfeld gibt es bislang wenige Anwendungs- und Optimierungsbeispiele dieser Dienste im Hinblick auf die speziellen Anforderungen asynchroner Multiplayer-Spiele.

Das Ziel dieser Arbeit ist es, die Potenziale und Herausforderungen der Nutzung von AWS im Kontext cloudbasierter, asynchroner Multiplayer-Spielarchitekturen systematisch zu untersuchen und daraus konkrete Empfehlungen für eine optimale Architekturentwicklung abzuleiten.

[3] Vgl. Lonis, Peter: The Power of Cloud Computing in the Gaming Industry, in: Leaseweb Blog, 29.06.2023, [online] https://blog.leaseweb.com/2023/06/29/the-power-of-cloud-computing-in-the-gaming-industry/ (abgerufen am 18.12.2024).

2.1 Vorgehen

Um dies zu erreichen, werden zunächst die Grundlagen und Konzepte von asynchronen Multiplayer-Spielen und Cloud Computing erklärt. Anschließend wird auf alle wichtigen AWS-Dienste eingegangen, die in den Architekturlösungen verwendet werden. Darauf folgt eine Auswahl und Erklärung von bestehenden Architekturen, die für asynchrone Multiplayer-Spiele empfohlen oder verwendet werden. Im Anschluss erfolgt ein umfangreicher Vergleich und eine Analyse jeder Architektur. Aus den Erkenntnissen dieses Vergleichs werden dann Empfehlungen abgeleitet, wann welche Architektur verwendet werden sollte.

Mit dieser Herangehensweise soll ein Beitrag zur wissenschaftlichen Diskussion im Bereich Cloud Computing und Multiplayer-Game-Design geleistet werden und ein praxisnaher Leitfaden für Entwickler entstehen, der bei der Realisierung zukünftiger Spieleprojekte als Orientierung dient.

3 Grundlagen

Dieses Kapitel bildet die Grundlage der in dieser Arbeit behandelten Themen. Es werden grundlegende Konzepte, AWS als Cloud-Computing-Anbieter sowie die primär genutzten AWS-Dienste detailliert erläutert, um einen einheitlichen Bezugsrahmen zu schaffen, auf dessen Basis die weiteren Analysen und Vergleiche aufbauen.

3.1 Asynchrone Multiplayer-Spiele

Asynchrone Multiplayer-Spiele sind Mehrspieler-Spiele, bei denen die Teilnehmer nicht gleichzeitig online sein müssen, um miteinander zu interagieren. Die Aktionen der Spieler erfolgen zeitversetzt. Ein Spieler kann einen Spielzug oder eine Aktion ausführen und das Spiel speichert diesen Zustand, sodass ein anderer Spieler ihn später abrufen und darauf reagieren kann. Ein Server verwaltet den fortlaufenden Spielzustand und koordiniert die Züge der Spieler, ohne dass alle gleichzeitig online sein müssen. Ein gutes Beispiel hierfür ist das Brettspiel Schach, hier ist ein zeitlicher Versatz zwischen den Spielerzügen möglich.[4]

Die Entwicklung asynchroner Multiplayer-Spiele stellt Entwickler vor eine technische Herausforderung. Da die Spieler nicht gleichzeitig verbunden sind, muss die Spielwelt oder der Spielzustand dauerhaft verfügbar sein. Der Spielzustand muss auf einem Server gespeichert werden, der nach jedem Zug die relevanten Daten in eine Datenbank schreibt, damit sie von anderen Spielern später abgerufen werden können. Dieser serverseitig gespeicherte Zustand bildet den SPOT[5] der Partie und muss konsistent bleiben, selbst wenn Spieler nacheinander und mit Unterbrechungen spielen. Eine robuste Datenbank oder ein dedizierter Spielserver ist also unerlässlich, um Züge zwischenzulagern und an die Spieler zu verteilen.

[4] Vgl. Secchi, Marco.: Multiplayer Game Development with Unreal Engine 5, Packt Publishing, 06.10.2023, S. 13.
[5] „Single Point of Truth“ (SPOT) - engl. für „einziger Punkt der Wahrheit“.

3.1.1 Client-Server Modell

Bei asynchronen Multiplayer-Spielen ist die Netzwerkarchitektur oft deutlich weniger komplex als bei Echtzeitspielen. Anstatt einer permanenten Live-Verbindung mit Fokus auf niedriger Latenz werden hier überwiegend HTTP-Anfragen genutzt, ähnlich wie in klassischen Webanwendungen. Sobald ein Spieler einen Zug ausführt oder den aktuellen Spielstand anfordert, wird eine entsprechende Anfrage an einen Server gesendet. Dieses Modell, worin Spieler ausschließlich über einen zentralen Server miteinander kommunizieren, wird Client-Server Modell genannt.[6]

Ein wesentlicher Vorteil dieses Modells liegt in der hohen Sicherheit, da alle kritischen Berechnungen und Statusänderungen zentral auf dem Server stattfinden. Dadurch wird der unbefugten Manipulation von Spieler-Daten effektiv entgegengewirkt. Allerdings muss der Server ständig verfügbar sein und ist bei hoher Nutzerzahl anfälliger für Engpässe, da viele gleichzeitige Zugriffe zu einer Überlastung führen können.

Diese permanente Verfügbarkeit eines eigenen Servers bringt jedoch auch erhebliche Herausforderungen mit sich, insbesondere in Bezug auf die Skalierung und die hohen initialen Investitionskosten. Große Server-Infrastrukturen sind sehr teuer, da sie kontinuierlich betrieben und auf einem hohen Leistungsniveau gehalten werden müssen. Hier bietet Cloud Computing eine kostengünstige und einsteigerfreundliche Lösung.

[6] Vgl. Apperson, Lem: Beginning Game Development: Client-Server Architecture, in: Medium, 17.11.2024, [online] https://medium.com/@lemapp09/beginning-game-development-client-server-architecture-1b7676d80dea (abgerufen am 08.01.2025).

3.2 Cloud Computing

Cloud Computing bezeichnet ein Modell in der Informatik, bei dem IT-Ressourcen, wie Rechenleistung, Speicher, Netzwerke und Anwendungen, bei Bedarf als Dienstleistung über ein Netzwerk, typischerweise das Internet, bereitgestellt und nach Nutzungsumfang abgerechnet werden.[7] Dies hat den großen Vorteil, dass kein großer manueller Verwaltungsaufwand nötig ist. Anstatt eigene Server oder Rechenzentren zu betreiben, beziehen Kunden die benötigte IT-Infrastruktur und Dienste bedarfsgerecht von einem Cloud-Anbieter und müssen sich nicht um Wartung oder Updates der zugrunde liegenden Hardware kümmern.[8] Dieses Modell ermöglicht es Nutzern, flexibel und schnell auf IT-Ressourcen zuzugreifen, ohne hohe Anfangsinvestitionen tätigen zu müssen.[9]

3.2.1 Vorteile und Nutzen

Cloud Computing bietet gegenüber traditionellen lokalen Serverlösungen eine Reihe von Vorteilen, die auch für die Bereitstellung von asynchronen Multiplayer-Spielen relevant sind:

Skalierbarkeit und Flexibilität: Cloud-Ressourcen lassen sich bei Bedarf nahezu in Echtzeit in unterschiedlichen Dimensionen erweitern oder verringern. Unternehmen können diese Ressourcen mithilfe des Cloud-Anbieters flexibel und automatisiert skalieren und so ihre Dienste zeitnah dem aktuellen Bedarf anpassen.[10] Das ist ein großer Vorteil für asynchrone Multiplayer, da deren Spielerzahlen und somit auch die benötigten Ressourcen oft stark schwanken.[11]

---

[7] Vgl. BSI (Bundesamt für Sicherheit in der Informationstechnik): Cloud Computing Grundlagen, o. D., [online] https://www.bsi.bund.de/dok/6622124, Kapitel 1: Was ist Cloud Computing (abgerufen am 12.12.2024).
[8] Vgl. Shrivastava, Saurabh/Neelanjali Srivastav/Alberto Artasanchez/Imtiaz Sayed: AWS for Solutions Architects, 2. Aufl., Packt Publishing, 28.04.2023, S. 2 f.
[9] Vgl. Amazon Web Services, Inc.: AWS Whitepaper: Overview of Amazon Web Services, 27.08.2024b, [PDF] https://docs.aws.amazon.com/pdfs/whitepapers/latest/aws-overview/aws-overview.pdf, S. 3.
[10] Vgl. Shrivastava et al., 2023, S. 14 f.
[11] Vgl. Kapitel 8.1.3 „CCU und Lastenspitzen“

Kosteneffizienz: Anstelle hoher Investitionskosten für eigene Server-Hardware fallen bei Cloud-Diensten lediglich nutzungsabhängige Ausgaben an. Die Abrechnung erfolgt nach Verbrauch. Es werden demnach nur die Ressourcen bezahlt, die tatsächlich genutzt wurden, wodurch Überkapazitäten entfallen.[12] Diese Vorteile sind einerseits für Indie-Studios nützlich, da ihnen die finanzielle Unterstützung durch Publisher fehlt, andererseits ermöglichen sie eine kostengünstige Umsetzung eines Machbarkeitsnachweises für asynchrone Mechaniken. Hinzu kommt, dass durch den Skaleneffekt große, global agierende Cloud-Anbieter ihre Infrastruktur bzw. Dienstleistungen sehr günstig anbieten können.[13]

Weniger Administrationsaufwand: Wartung und Betrieb der zugrunde liegenden Infrastruktur übernimmt der Cloud-Anbieter. Updates, Hardwareaustausch und Sicherheitsmaßnahmen werden vom Provider durchgeführt, sodass sich Unternehmen auf ihre Kernkompetenzen und die Entwicklung ihrer Anwendungen konzentrieren können.[14]

Zuverlässigkeit und Sicherheit: Cloud-Anbieter betreiben ihre Rechenzentren redundant und überwachen diese rund um die Uhr. Dadurch werden eine hohe Verfügbarkeit der Dienste und ein Schutz vor Datenverlust gewährleistet. Außerdem profitieren Kunden von modernen Sicherheitsmaßnahmen der Anbieter, die einzelne Unternehmen in diesem Umfang oft nicht eigenständig umsetzen könnten.[15] Da asynchrone Multiplayer größtenteils kompetitive Spiele sind, tragen diese Vorteile dazu bei, ein faires und stabiles Spielerlebnis ohne großen Aufwand zu gewährleisten.

[12] Vgl. Amazon Web Services, Inc., 2024b, S. 3.
[13] Vgl. Labes, Stine: Grundlagen des Cloud Computing – Konzept und Bewertung von Cloud Computing, Universitätsverlag der TU Berlin, Bd. 1, 11.2012, [PDF] https://api-depositonce.tu-berlin.de/server/api/core/bitstreams/02384947-8d6d-47dd-9dfc-163b580abed4/content, S. 22
[14] Vgl. Shrivastava et al., 2023, S. 24.
[15] Vgl. ebd., S. 18-23.

3.2.2 Servicemodelle

Die angebotenen Dienste eines Cloud-Anbieters untergliedern sich in unterschiedliche Servicemodelle, die sich durch den Grad der vom Anbieter übernommenen Infrastruktur unterscheiden. Die Entscheidung, welche Servicemodelle verwendet werden sollen, hängt von den vorhandenen Ressourcen bzw. dem Fachwissen der Entwickler ab und davon, wie viel Freiheit sie in der Auswahl der Dienste und deren Bereitstellungsmöglichkeiten benötigen.

Infrastructure as a Service (IaaS): Der Cloud-Anbieter stellt grundlegende IT-Ressourcen, wie virtuelle Server, Speicher und Netzwerke, als Dienste bereit. Auf dieser Infrastruktur können dann eigene Betriebssysteme installiert und Anwendungen ausgeführt werden. Der Kunde hat hier die meiste Kontrolle, da er ab dem Betriebssystem aufwärts alles selbst verwaltet.[16]

Platform as a Service (PaaS): Dieses Modell stellt Entwicklern eine komplette Plattform zur Verfügung, auf der sie eigene Anwendungen entwickeln und betreiben können, ohne sich um die zugrunde liegende Infrastruktur kümmern zu müssen. Der Cloud-Anbieter stellt die komplette Laufzeitumgebung inklusive Betriebssystems, Middleware und Datenbank-Backend bereit. Der Kunde ist nur noch für seine Anwendungen und Daten verantwortlich.[17]

Software as a Service (SaaS): Fertige Software-Anwendungen werden vom Cloud-Anbieter als Dienst über das Internet zur Verfügung gestellt. Der Endnutzer greift in der Regel über einen Webbrowser auf die Anwendungen zu, ohne diese selbst installieren oder warten zu müssen. Die gesamte Verwaltung, vom Server bis zur Anwendung, übernimmt der Cloud-Anbieter.[18]

[16] Vgl. Shrivastava et al., 2023, S. 65; BSI, o. D., Kapitel 3.1: Infrastructure as a Service (IaaS).
[17] Vgl. Shrivastava et al., 2023, S. 71 f.; BSI, o. D., Kapitel 3.2: Platform as a Service (PaaS).
[18] Vgl. Shrivastava et al., 2023, S. 67; BSI, o. D., Kapitel 3.3: Software as a Service (SaaS).

3.2.3 Bereitstellungsmodelle

Neben den Service-Modellen unterscheiden sich Cloud-Umgebungen auch in der Bereitstellung, bzw. wie Infrastruktur an Kunden vermietet wird. Dabei unterscheidet man zwischen zwei Haupt-Modellen, der öffentlichen und privaten Cloud, die sich in weitere Unterkategorien oder Hybridmodelle aufteilen lassen. Diese Arbeit nutzt ausschließlich das Modell der öffentlichen Cloud, um die Vorteile des Cloud Computing bestmöglich nutzen zu können.

Öffentliche Cloud: Cloud-Dienste und IT-Ressourcen werden von einem externen Cloud-Anbieter über das öffentliche Internet bereitgestellt und über mehrere Kunden verteilt angeboten.[19] Die Nutzer mieten sich hierbei Anteile an einer großen, vom Anbieter betriebenen Infrastruktur. Zum Beispiel laufen Dienste und Anwendungen der Kunden zwar in einer privaten, virtuell getrennten Umgebung, teilen sich aber dieselbe Hardware, wie CPU oder RAM.[20] Deshalb erfolgt bei diesem Modell die Abrechnung nutzungsbasiert, ohne dass der Kunde in eigene Hardware investieren muss.

Private Cloud: Eine Private Cloud ist eine Cloud-Umgebung, die ausschließlich für bzw. von einem einzelnen Unternehmen betrieben wird. Die zugrundeliegende Infrastruktur wird entweder im eigenen Rechenzentrum des Unternehmens oder von einem dritten Dienstleister exklusiv für dieses Unternehmen betrieben.[21] Private Clouds kommen vor allem zum Einsatz, wenn sensible Daten oder geschäftskritische Anwendungen aus Sicherheits- oder Compliance-Gründen nicht in einer öffentlichen Umgebung laufen sollen oder wenn sehr spezifische Anforderungen an die Infrastruktur bestehen.[22]

[19] Vgl. Labes, 2012, S. 16; BSI, o. D., Kapitel 2.2.
[20] Wird auch als „Virtual Private Cloud“ (VPC) bezeichnet. Näheres dazu im Kapitel 4.1.1.
[21] Vgl. BSI, o. D., Kapitel 2.1.
[22] Vgl. Labes, 2012, S. 16 f.

3.2.4 Cloud-Architekturen

Cloud-Architekturen beschreiben, wie verschiedene Dienste innerhalb eines Cloud-Anbieters zusammenwirken, um eine sinnvolle Cloud-Umgebung zu schaffen. Im Fokus steht dabei eine reibungslose Integration aller Komponenten. Das Ziel ist eine individuell zugeschnittene und gleichzeitig flexible Umgebung zu realisieren, die sich auch an zukünftige Anforderungen problemlos anpassen lässt.

Die Konzeption einer solchen Architektur erfordert eine gründliche Analyse der betrieblichen Anforderungen und eine sorgfältige Planung. Nur so kann gewährleistet werden, dass die eingesetzten Komponenten optimal miteinander interagieren. Die Herausforderung liegt darin, die Komplexität des Zusammenspiels der Dienste zu minimieren und gleichzeitig eine Umgebung zu schaffen, die anpassungsfähig und zukunftssicher ist.

Solch eine nach Anforderungen angefertigte Architektur wird als Lösungs-Architektur bezeichnet, da sie die zugrunde liegenden Software-Probleme zielgerichtet adressieren.[23] Diese Lösungen bieten einen maßgeschneiderten Plan, um Herausforderungen, wie mangelnde Skalierbarkeit, unzureichende Integration von Diensten und hohe Komplexität, zu überwinden. Das daraus entstehende eigenständige Berufsbild des Lösungs-Architekten zeigt, wie vielseitig und komplex das Thema rund um Cloud Computing ist.[24]

[23] Vgl. Shrivastava, Saurabh/Neelanjali Srivastav: Solutions Architect's Handbook, 3. Aufl., Packt Publishing, 29.03.2024, S. 2 f.
[24] Vgl. ebd., S. 4 ff.

Diese Arbeit soll daher den Einstieg für die Entwicklung von Lösungs-Architekturen in der abgegrenzten Problemdomäne der asynchronen Multiplayer-Architekturen vereinfachen und dafür eine spezifische beste Vorgehensweise, angelehnt an das Vorgehen eines Lösungs-Architekten, ableiten. Der Beruf des Lösungs-Architekten kann in mehrere spezialisierte Typen unterteilt werden.[25] Diese Arbeit bezieht sich auf die Definition des Cloud-Architekten, der laut Shrivastava und Srivastav Lösungs-Architekturen plant, designt und bereitstellt. [26] Dieser Berufstyp benötigt ein tiefgreifendes Wissen über die verschiedenen Dienste der Cloud-Anbieter sowie deren Zusammenspiel in cloudnativen Entwurfsmustern.

3.2.5 Entwurfsmuster

Damit eine Architektur die Vorteile aus Kapitel 3.2.1 perfekt umsetzen bzw. nutzen kann, entstanden über die Jahre verschiedene Prinzipien zum Designen von Lösungs-Architekturen in der Cloud. Diese Architektur-Prinzipien werden als cloudnative Entwurfsmuster bezeichnet, also Ansätze, welche die Anwendungen optimal auf eine Cloud-Umgebung abstimmen. Auch die im Folgenden analysierten und verglichenen Architekturen bauen auf solchen Prinzipien auf.

Schichtenarchitektur: Hier wird eine Software in logisch abgegrenzte Ebenen unterteilt, wodurch eine klare Trennung der Verantwortlichkeiten erreicht wird. Jede Ebene konzentriert sich auf einen spezifischen Aufgabenbereich. [27] Übliche Schichten sind beispielsweise die Präsentationsschicht, die Geschäftslogik und der Datenbankzugriff. Dies verbessert die Wartbarkeit und Skalierbarkeit der Anwendung im Gegensatz zu einer monolithischen Architektur, in der diese Schichten nicht logisch voneinander getrennt sind.[28]

[25] Vgl. Shrivastava/Srivastav, 2024, S. 7.
[26] Vgl. ebd., S. 8 f.
[27] Vgl. Shrivastava et al., 2023, S. 510.
[28] Vgl. ebd., S. 511 f.

Diese Trennung ermöglicht den Einsatz spezialisierter Dienste in den einzelnen Schichten sowie eine einfache Austauschbarkeit einzelner Komponenten. Dadurch, dass in diesem Entwurfsmuster oft nur die oberste Schicht öffentlich zugänglich ist, wird zudem die Systemsicherheit erhöht.

Microservices: Dieses Entwurfsmuster bietet eine weitverbreitete Lösung auf die Probleme klassischer monolithischer Systeme, welche durch ihre starke Kopplung in Wartbarkeit und Skalierung extrem eingeschränkt sind.[29] Microservices stellen einen cloudnativen Ansatz dar, bei dem eine Softwareanwendung als eine Ansammlung kleiner, unabhängiger Dienste realisiert wird, die jeweils einem klar definierten Geschäftskontext zugeordnet sind. Diese lassen sich unabhängig voneinander skalieren und kommunizieren über definierte Schnittstellen, sogenannter APIs, was eine lose Kopplung ermöglicht.[30]

3.2.6 Architekturparadigmen

Architektur-Paradigmen im Bereich der Cloud-Architekturen sind Denkmodelle bzw. Leitfäden für die Planung und Bereitstellung dieser Entwurfsmuster.

Containerisierung: Dieses Prinzip bezeichnet den Prozess, bei dem alle Teile einer Anwendung mit allen benötigten Abhängigkeiten in isolierte Container verpackt werden, die in einer virtualisierten Umgebung laufen. Diese Container ermöglichen eine konsistente und portable Ausführung der Anwendung über verschiedene Betriebssysteme und Infrastrukturen hinweg.[31] Containerisierung eignet sich zudem sehr gut für die Bereitstellung, Orchestrierung und Skalierung von Microservices.[32]

[29] Vgl. Shrivastava et al., 2023, S. 508.
[30] Vgl. ebd., S. 506 f.
[31] Vgl. ebd., S. 463-467.
[32] Vgl. Singh, Vindeep/Sateesh K Peddoju: Container-based microservice architecture for cloud applications, IEEE, 05.2017, [PDF] doi:10.1109/ccaa.2017.8229914, S. 848.

Serverless Computing: Das Prinzip des Serverless Computing[33] ist ein moderner Ansatz, der die zugrunde liegende Infrastruktur vollständig abstrahiert, sodass Entwickler sich ausschließlich auf den Anwendungs- bzw. Funktionscode konzentrieren können. Dabei wird der Programmcode ereignisgesteuert ausgeführt und die Ressourcenallokation erfolgt dynamisch durch den Cloud-Anbieter.[34] Auch hiermit lassen sich Microservices bereitstellen, sogar noch einfacher und feinteiliger als bei dem Prinzip der Containerisierung, da nun nicht mehr zwischen Anwendungen, sondern einzelnen Funktionen unterschieden werden kann.

Hierfür werden spezialisierte Servicemodelle bereitgestellt, die als Function as a Service und Backend as a Service (FaaS & BaaS) bezeichnet werden. Diese Servicemodelle sind wesentliche Bestandteile des Konzepts der serverlosen Datenverarbeitung.[35] FaaS ermöglicht die Ausführung von zustandslosem Programmcode als Reaktion auf Ereignisse, während BaaS vorgefertigte Backend-Dienste, wie Datenbanken oder API-Schnittstellen bereitstellt, ohne dass Entwickler sich um den Betrieb, die Wartung und die Ausführung der zugrunde liegenden Server kümmern müssen. Beide sind verfeinerte Kategorien von PaaS. Auch containerbasierte Architekturen können in der Cloud serverlos betrieben werden. Dieses Servicemodell wird als Container as a Service (CaaS) bezeichnet.[36] Kapitel 4.2.4 wird näher darauf eingehen.

[33] engl. für serverlose Datenverarbeitung
[34] Vgl. Li, Yongkang/Yanying Lin/Yang Wang/Kejiang Ye/Chengzhong Xu: Serverless Computing: State-of-the-Art, Challenges and Opportunities, in: IEEE Transactions On Services Computing, Bd. 16, Nr. 2, 12.04.2022, [PDF] doi:10.1109/tsc.2022.3166553, S. 1522.
[35] Vgl. ebd., S.1522, 1529; vgl. Shrivastava et al., 2023, S. 180.
[36] Vgl. Shrivastava et al., 2023, S. 498.

Event-Driven Architecture (EDA): Die ereignisgesteuerte Architektur basiert auf dem Prinzip, dass Komponenten eines Systems über Ereignisse asynchron kommunizieren. [37] Diese Entkopplung trägt ebenfalls zu einer hohen Flexibilität und Skalierbarkeit bei. Es unterstützt bzw. stellt sicher, dass Dienste untereinander optimal kommunizieren können und gleichzeitig die lose Kopplung aller Dienste erhalten bleibt.[38]

3.2.7 Architekturdiagramme

Architekturdiagramme dienen dazu, komplexe Cloud-Architekturen übersichtlich darzustellen und einen schnellen Überblick über die wesentlichen Systemkomponenten und ihre Zusammenhänge zu ermöglichen. Ziel ist es, durch eine vereinfachte Visualisierung die Verständlichkeit und Wartbarkeit von Architekturen zu verbessern.[39]

Es gibt keine standardisierten Vorschriften für solche Diagramme, da jeder Cloud-Anbieter eigene Regeln und Symboliken definiert. AWS unterteilt seine Architekturdiagramme in verschiedene Unterkategorien.[40] In dieser Arbeit werden Systemarchitekturen verwendet, die Infrastruktur oberflächlich darstellt und nicht zu sehr auf die individuellen Aufgaben der Dienste eingeht, was in Beispiel- oder Referenzarchitekturen nicht erforderlich ist. Für die in dieser Arbeit erstellten Diagramme kommen die aktuellen AWS-Architektursymbole[41] zum Einsatz und als Zeichentool wird draw.io[42] verwendet. Das gewährleistet eine konsistente Darstellung der entworfenen Architekturen.

[37] Vgl. Shrivastava et al., S. 513 ff.
[38] Vgl. ebd., 2023, S. 521 f.
[39] Vgl. Amazon Web Services, Inc.: What is Architecture Diagramming?, o. D.d, [online] https://aws.amazon.com/what-is/architecture-diagramming/ (abgerufen am 20.01.2025).
[40] Vgl. ebd., Kapitel 5: What are the types of architectural diagrams?
[41] Amazon Web Services, Inc.: Architecture Icon Page, o. D.a, [online] https://aws.amazon.com/architecture/icons/ (abgerufen am 22.01.2025); siehe Kapitel 4.5.
[42] Draw.io: o. D., [online] https://www.drawio.com/.

3.3 Amazon Web Services

Amazon Web Services (AWS)[43] ist die Cloud-Computing-Plattform des US-Konzerns Amazon und gilt als einer der weltweit führenden Anbieter von Cloud-Diensten. AWS wurde im Jahr 2006 gestartet, als Amazon begann, seine IT-Infrastruktur als Dienstleistungen für externe Kunden anzubieten.[44] Dieses Angebot prägte den Begriff des Cloud Computing und trug maßgeblich zur Popularisierung und praktischen Umsetzung von Cloud-Diensten bei. Seitdem hat sich AWS zum größten Public-Cloud-Anbieter entwickelt und behauptet trotz wachsender Konkurrenz eine führende Position am Markt (siehe Abbildung 1).[45]

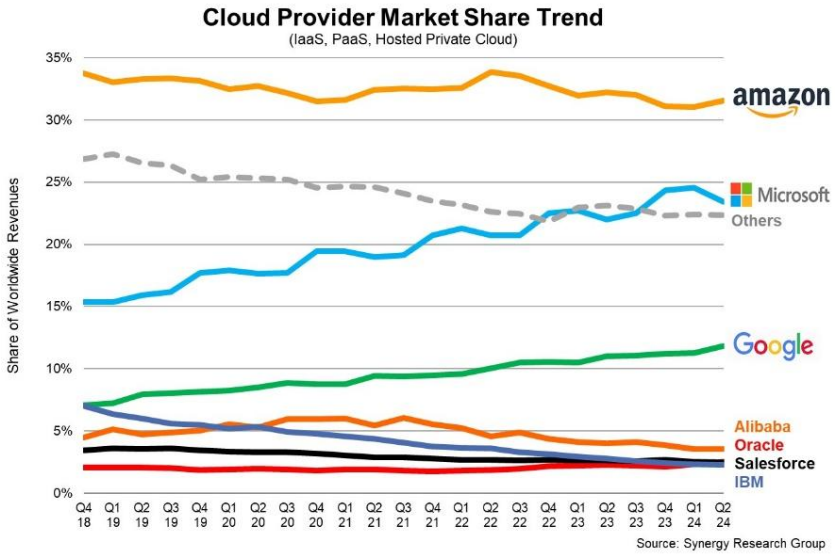


Abbildung 1: Marktanteile führender Cloud-Anbieter von Q4 2018 bis Q2 2024 auf Basis weltweiter Umsätze Quelle: Synergy Research Group, 2024

[43] Amazon Web Services, Inc.: Start building on AWS today, o. D.b, [online] https://aws.amazon.com/ (abgerufen am 18.12.2024).
[44] Vgl. Clark, Jack: How Amazon exposed its guts: The History of AWS's EC2, in: ZDNET, 07.06.2012, [online] https://www.zdnet.com/article/how-amazon-exposed-its-guts-the-history-of-awss-ec2/ (abgerufen am 18.12.2024); vgl. Shrivastava et al., 2023, S. 181.
[45] Vgl. Shrivastava et al., 2023, S. 9.

AWS bietet ein vielseitiges Angebot an Cloud-Diensten, das alle gängigen Servicemodelle von IaaS über PaaS bis hin zu serverlosen Architekturen abdeckt. Ergänzt wird das Angebot durch diverse Zusatzdienste, wie Datenbanken, skalierbare Speicherlösungen und integrierte Sicherheitsfunktionen, die eine umfassende Entwicklung der Projekte ermöglichen.[46]

3.3.1 Warum AWS?

In der vorliegenden Arbeit wird AWS als zentrale Cloud-Plattform verwendet. Für diese Auswahl wurden im Voraus die drei größten Anbieter im Bereich Cloud Computing (AWS, Microsoft Azure und Google Cloud Plattform) verglichen.

Durch das große Service-Portfolio von über 200 verschiedenen Diensten, die von grundlegenden Infrastrukturkomponenten bis hin zu spezialisierten, branchenspezifischen Lösungen reichen, bietet AWS die flexibelste und am weitesten entwickelte Plattform. [47] Dies ermöglicht einen differenzierten Vergleich, da nahezu alle relevanten Aspekte moderner Cloud-Anwendungen abgedeckt werden können.

Zudem sollten durch den führenden Marktanteil von AWS mehr Fallstudien im Bereich der asynchronen Multiplayer-Spiele vorliegen. Diese praktischen Anwendungen und Erklärungen von AWS-Diensten in diesen Fallstudien liefern empirische Daten, die sich als Grundlage für die Erstellung des Leitfadens eignen.

[46] Vgl. Shrivastava et al., 2023, S. 11-13.
[47] Vgl. ebd., S. 7 ff.

4 Analyse verfügbarer Dienste

AWS bietet eine überwältigende Anzahl von Produkten[48] an. Zum Zeitpunkt dieser Arbeit können Kunden zwischen insgesamt 354 Produkte wählen, die sich in 22 Produktkategorien unterteilen lassen, weswegen sie in dieser Arbeit nicht behandelt werden.[49]

In diesem Kapitel werden existierende Dienste, die sich für Multiplayer-Spiele-Architekturen eignen, analysiert. Sie werden dabei grob in ihren Zuständigkeitsbereich gruppiert. Um sich der Vorgehensweise eines Lösungs-Architekten anzunähern, ist ein gutes Verständnis der einzelnen Dienste nötig. Auf Grundlage dieses Wissens, kann dann eine passende Lösungs-Architektur entworfen werden.

Für viele Dienste bietet AWS alternative Dienste an, die im Wesentlichen denselben Service bereitstellen, sich jedoch in bestimmten Merkmalen, wie etwa den verwendeten Standards, unterscheiden. Um ihren grundlegenden Zweck besser darzustellen und die Analyse übersichtlich zu gestalten, konzentriert sich dieses Kapitel im Folgenden auf einen einzelnen Dienst aus diesen Alternativen. Kapitel 8.3 wird näher darauf eingehen, wann welcher alternative Dienst eingesetzt werden sollte.

[48] AWS-Produkte sind nicht gleichzusetzen mit AWS-Diensten, da ein AWS-Dienst mehrere Produkte in sich vereinen kann.
[49] Dies beinhaltet Dienste aus den folgenden Produktkategorien: Artificial Intelligence, Blockchain, Business Applications, End User Computing, Migration & Transfer, Quantum Technologies, Robotics und Satellite.

4.1 Netzwerk und Infrastruktur

4.1.1 Amazon Virtual Private Cloud (VPC)

Amazon VPC ermöglicht es, AWS-Ressourcen in einem logisch isolierten virtuellen Netzwerk in der AWS-Cloud bereitzustellen. [50] Ein VPC wird in einer bestimmten AWS-Region angelegt, die aus mehreren räumlich getrennten Rechenzentrumseinheiten, den Availability Zones (AZ), besteht.

Innerhalb des VPC kann der Nutzer Subnetze anlegen, wobei jedes Subnetz an genau eine AZ gebunden ist. Die Nutzer haben die vollständige Kontrolle über den IP-Adressbereich, die Routing-Tabellen, die Netzwerkgateways und die Sicherheitsfunktionen ihrer VPC. So können beispielsweise öffentliche Subnetze für Webserver mit Internetzugang über ein Internet-Gateway und private Subnetze für Datenbankserver ohne externen Zugriff eingerichtet werden.

Amazon VPC bildet die Grundlage für nahezu alle Cloud-Anwendungen auf AWS, da es eine isolierte und sichere Netzwerkumgebung für viele Dienste bereitstellt. Es ist daher ein essenzieller Dienst, der es Nutzern ermöglicht, ihre Infrastruktur segmentiert, sicher und den eigenen Netzwerkanforderungen entsprechend zu betreiben.[51] Abbildung 2 illustriert, wie diese Unterteilungen in einem Architekturdiagramm dargestellt werden.

[50] Vgl. Amazon Virtual Private Cloud (VPC): in: AWS Produkte, o. D., [online] https://aws.amazon.com/vpc/ (abgerufen am 23.01.2025).
[51] vgl. Amazon Web Services, Inc.: Amazon Virtual Private Cloud - User Guide, 09.12.2024a, [PDF] https://docs.aws.amazon.com/pdfs/vpc/latest/userguide/vpc-ug.pdf, S. 1;
vgl. Amazon Web Services, Inc., 2024b, S. 115; vgl. Shrivastava et al., 2023, S. 106 f., S. 112 ff.

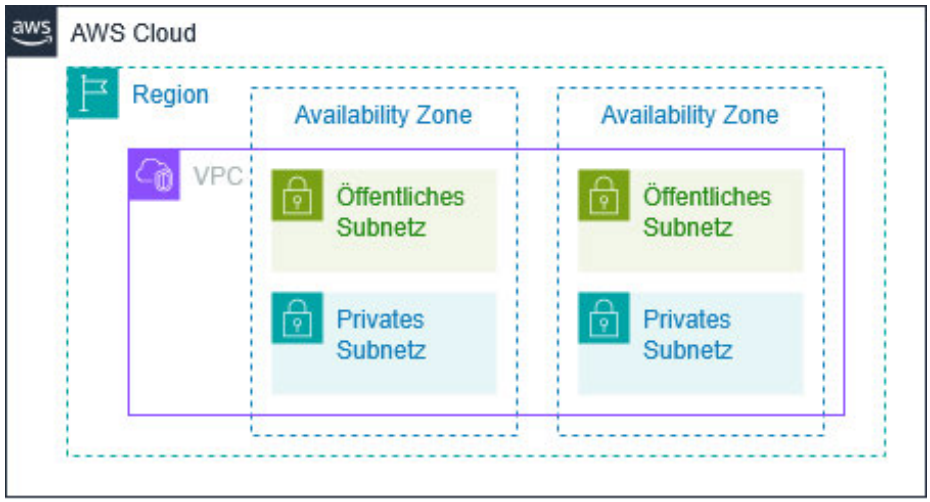


Abbildung 2: Darstellung von Regionen, VPC, AZs und Subnetzen in einem AWS-Architekturdiagramm
Quelle: Eigene Darstellung

4.1.2 Amazon Route 53

Amazon Route 53 ist ein vollständig verwalteter und skalierbarer Dienst für ein Domain-Name-System (DNS).[52] Er ermöglicht die Registrierung von Domainnamen und die Verwaltung von DNS-Zonen, um Nutzeranfragen an die entsprechenden Server zu leiten. Route 53 unterstützt viele Routing-Strategien, um den Datenverkehr optimal zu verteilen und Ausfallsicherheit durch automatische Failover-Mechanismen zu gewährleisten. Damit können Anfragen von Endnutzern effizient an die nächstgelegenen oder verfügbaren Infrastrukturkomponenten geleitet werden, was Latenzen reduziert und die Zuverlässigkeit erhöht.[53]

[52] Vgl. Amazon Route 53: in: AWS Produkte, o. D., [online] https://aws.amazon.com/route53/ (abgerufen am 10.01.2025).
[53] Vgl. Amazon Web Services, Inc., 2024b, S. 114 f.; vgl. Shrivastava et al., 2023, S. 122 ff.

4.1.3 Amazon API Gateway

Amazon API Gateway ist ein Dienst zur Erstellung, Veröffentlichung und Überwachung von APIs.[54] Mit API Gateway können RESTful HTTP-APIs und WebSocket-APIs definiert werden, die als Einstiegspunkt für Anwendungen dienen. Der Dienst übernimmt dabei Aufgaben wie das Routing von Anfragen zu anderen Diensten, die Implementierung von Authentifizierung und Autorisierung, die Traffic-Drosselung sowie die Überwachung der APIs. API Gateway skaliert automatisch und bietet Caching-Optionen. Durch diese Funktionen können APIs mit minimalem administrativem Aufwand performant, sicher und verlässlich bereitgestellt werden.[55]

4.1.4 AWS Global Accelerator

AWS Global Accelerator ist ein verwalteter Netzwerkdienst, der die Geschwindigkeit und Verfügbarkeit verteilter Anwendungen verbessert. [56] Der Dienst stellt zwei statische, globale IP-Adressen bereit, die als feste Einstiegspunkte zu einer Anwendung dienen. Diese IP-Adressen werden über AWS-Edge-Standorte weltweit als Zugangspunkte bekanntgegeben. Dadurch wird eingehender Datenverkehr automatisch an den geografisch nächstgelegenen Edge-Standort geleitet und von dort in das AWS-Netzwerk eingespeist. Anschließend durchquert der Datenverkehr das dedizierte AWS-Backbone-Netzwerk bis zum Zielserver der Anwendung in einer AWS-Region, z.B. eine EC2-Instanz oder ein Load Balancer.[57]

[54] Vgl. Amazon API Gateway: in: AWS Produkte, o. D., [online] https://aws.amazon.com/api-gateway/ (abgerufen am 11.01.2025).
[55] Vgl. Amazon Web Services, Inc., 2024b, S. 114; vgl. Shrivastava et al., 2023, S. 509, 579.
[56] Vgl. AWS Global Accelerator: in: AWS Produkte, o. D., [online] https://aws.amazon.com/global-accelerator/ (abgerufen am 11.01.2025).
[57] Vgl. Amazon Web Services, Inc., 2024b, S. 117 f.; vgl. Shrivastava et al., 2023, S. 126 f.

4.1.5 Amazon CloudFront

Amazon CloudFront ist ein Content-Delivery-Netzwerk (CDN), das die Auslieferung von Webinhalten an Nutzer weltweit beschleunigt.[58] Es verteilt Inhalte über ein globales Netzwerk von Rechenzentren an verschiedenen Standorten, sogenannte Edge-Servern, die näher bei den Endbenutzern liegen. Statische Inhalte wie Bilder, Videos oder Stylesheets sowie dynamische Daten können dadurch mit geringer Latenz bereitgestellt werden. Ruft ein Nutzer eine Datei oder Ressource ab, leitet CloudFront die Anfrage automatisch an den Edge-Server weiter, der dem Nutzer geografisch am nächsten ist. Befindet sich der angefragte Inhalt dort bereits im Cache, wird er unmittelbar von diesem Edge-Standort aus ausgeliefert, was die Wartezeit deutlich reduziert.[59]

4.1.6 Elastic Load Balancing (ELB)

Elastic Load Balancing ist ein AWS-Dienst für automatischen Lastenausgleich, der eingehenden Netzwerkverkehr auf mehrere EC2-Instanzen verteilt, um die Skalierbarkeit und Fehlertoleranz von Anwendungen zu erhöhen.[60] AWS bietet drei Varianten von Load Balancern an, den Application Load Balancer (ALB) für HTTP/HTTPS-Traffic auf Anwendungsschicht (7. OSI-Schicht), den Network Load Balancer (NLB) für Transport-Layer-Traffic (4. OSI-Schicht) mit hoher Leistung und niedriger Latenz sowie den veralteten klassischen Load Balancer (CLB), der auf beiden Schichten operiert. Durch Betrieb in mehreren AZs stellt ELB sicher, dass auch bei Ausfall einzelner Instanzen oder ganzer AZs der Dienst verfügbar bleibt und der Traffic automatisch auf verbleibende Server umgeleitet wird.[61]

[58] Vgl. Amazon CloudFront: in: AWS Produkte, o. D., [online] https://aws.amazon.com/cloudfront/ (abgerufen am 02.02.2025).
[59] Vgl. Amazon Web Services, Inc., 2024b, S. 114; vgl. Shrivastava et al., 2023, S. 124 ff.
[60] Vgl. Elastic Load Balancing: in: AWS Produkte, o. D., [online] https://aws.amazon.com/elasticloadbalancing (abgerufen am 11.01.2025).
[61] Vgl. Amazon Web Services, Inc., 2024b, S. 120; vgl. Shrivastava et al., 2023, S. 207.

4.2 Compute

4.2.1 Amazon Elastic Compute Cloud (EC2)

Amazon EC2 ist ein IaaS-Dienst und stellt skalierbare und bedarfsgesteuerte Rechenkapazitäten in Form von virtuellen Maschinen bzw. Instanzen in der AWS-Cloud bereit.[62] Der Dienst ermöglicht es, schnell neue Serverinstanzen zu starten und bietet volle Kontrolle über das Betriebssystem und interne Netzwerkkonfiguration.[63] EC2 hat eine große Auswahl an Instanztypen (T3, M5, C5 usw.)[64], die unterschiedliche Hardwareanforderungen, wie CPU-Leistung, Arbeitsspeicher, Speicher und Netzwerkdurchsatz, für verschiedene Anwendungszwecke bereitstellen. Nutzer können je nach Bedarf Instanzen manuell oder automatisch mittels horizontaler Auto-Skalierung hoch- oder herunterskalieren.[65]

4.2.2 AWS Lambda

AWS Lambda ist das FaaS-Angebot für serverloses Computing von AWS.[66] Es ermöglicht das Ausführen von Programmcode[67] in Form kleiner Funktionen, ohne dass Entwickler Server bereitstellen oder verwalten müssen. Lambda startet die vom Nutzer geschriebenen Funktionen automatisch als Reaktion auf Ereignisse oder Aufrufe und skaliert die benötigte Recheninfrastruktur vollautomatisch im Hintergrund. Man bezahlt dabei nur für die tatsächliche Ausführungszeit und Aufrufanzahl des Programmcodes.[68]

---

[62] Vgl. Amazon Elastic Compute Cloud: in: AWS Produkte, o. D., [online] https://aws.amazon.com/de/ec2/ (abgerufen am 24.01.2024).
[63] Vgl. Shrivastava et al., 2023, S. 181-184.
[64] Vgl. ebd., S. 185 ff.
[65] Vgl. Amazon Web Services, Inc., 2024b, S. 37;
[66] Vgl. AWS Lambda: in: AWS Produkte, o. D., [online] https://aws.amazon.com/lambda/ (abgerufen am 24.01.2025).
[67] Zum Zeitpunkt dieser Arbeit werden folgende Programmiersprachen unterstützt: Java, Go, PowerShell, Node.js, C#, Python, and Ruby
[68] Vgl. Amazon Web Services, Inc., 2024b, S. 40; vgl. Shrivastava et al., 2023, S. 211 f.

4.2.3 Amazon Elastic Kubernetes Service (EKS)

Amazon EKS ist ein verwalteter Kubernetes-Dienst, der es erleichtert, containerisierte Anwendungen auf AWS auszuführen, ohne einen eigenen Kubernetes-Kontrollebene betreiben zu müssen. [69] EKS übernimmt die Einrichtung und den Betrieb der Kubernetes-Masterkomponenten über mehrere AZs. Die Worker-Knoten eines EKS-Clusters können entweder auf EC2-Instanzen oder serverlos über Fargate betrieben werden. Da EKS vollständig kompatibel zu Standard-Kubernetes ist, können bestehende Anwendungen oder Tools, die auf Kubernetes basieren, ohne Anpassungen genutzt werden.[70]

4.2.4 AWS Fargate

AWS Fargate ist ein serverloses CaaS Angebot für Container.[71] Es übernimmt die Bereitstellung und Verwaltung der zugrunde liegenden Infrastruktur, sodass Container direkt ausgeführt werden können, ohne eigene EC2-Instanzen für das Container-Management zu betreiben. Durch die Abstraktion der Serverebene vereinfacht Fargate den Betrieb von Container-Anwendungen erheblich und erhöht die Sicherheit, da jeder Task bzw. jeder Pod isoliert in einer eigenen Runtime-Umgebung läuft.[72] AWS Fargate erweitert die Funktionalitäten von Amazon EKS und Amazon Elastic Container Service (ECS)[73], ersetzt sie aber nicht. Es handelt sich um eine zusätzliche Methode zum Ausführen von Containern, die den operativen Aufwand deutlich reduziert.[74]

---

[69] Vgl. Amazon Elastic Kubernetes Service: in: AWS Produkte, o. D., [online] https://aws.amazon.com/eks/ (abgerufen am 24.01.2025).
[70] Vgl. Amazon Web Services, Inc., 2024b, S. 45; vgl. Shrivastava et al., 2023, S. 489.
[71] Vgl. AWS Fargate: in: AWS Produkte, o. D., [online] https://aws.amazon.com/fargate/ (abgerufen am 24.01.2025).
[72] Vgl. Amazon Web Services, Inc., 2024b, S. 40; vgl. Shrivastava et al., 2023, S. 212 f.
[73] Amazon ECS ist ein nativer AWS-Dienst zum Orchestrieren von Containern, nutzt aber nicht den verbreiteten Kubernetes-Standard (siehe Kapitel 8.3.4).
[74] Vgl. Shrivastava et al., 2023, S. 497 f.

4.2.5 Amazon GameLift

Amazon GameLift ist ein PaaS-Angebot, speziell für sitzungsbasierte Multiplayer-Spiele, welches dedizierte Spieleserver in der AWS-Cloud hostet.[75] Es ermöglicht Spieleentwicklern, Multiplayer-Spiele zu betreiben, ohne eine eigene Serverinfrastruktur aufzubauen. GameLift automatisiert die Bereitstellung, Skalierung und Verwaltung von EC2-Instanzen und ist auf geringe Latenzen optimiert. Zudem verteilt es die Spieleserver global über AWS-Regionen, passt die Ressourcen dynamisch an die Spieleranzahl an und bietet Funktionen, wie integriertes Matchmaking (FlexMatch) sowie Spieler-Sitzungsverwaltung. Darüber hinaus schützt GameLift Spiele mit dem integrierten DDoS-Schutz AWS Shield und bietet das GameLift SDK für gängige Spiele-Engines wie Unity oder Unreal zur einfacheren Integration an.[76]

[75] Vgl. Amazon GameLift: in: AWS Produkte, o. D., [online] https://aws.amazon.com/gamelift/ (abgerufen am 15.12.2024).
[76] Vgl. Amazon Web Services, Inc.: Amazon GameLift - Hosting Guide, 14.01.2025a, [PDF] https://docs.aws.amazon.com/pdfs/gamelift/latest/developerguide/gamelift-dg.pdf, S. 1 ff.

4.3 Datenspeicherung

4.3.1 Amazon Simple Storage Service (S3)

Amazon S3 ist ein Objektspeicherdienst, der zur Speicherung und zum Abruf beliebiger Datenmengen über das Internet dient.[77] S3 speichert Daten als Objekte, die in sogenannten Buckets hinterlegt werden. Jedes Objekt besteht aus den eigentlichen Daten, Metadaten und einem eindeutigen Schlüssel, über den es adressiert wird. Im Gegensatz zu herkömmlichen Dateisystemen mit Ordnerstruktur werden Objekte in einem Bucket ohne Pfad abgelegt, können aber anhand ihres Schlüssels logisch gruppiert werden. Der Zugriff auf S3 erfolgt über standardisierte Web-Protokolle (HTTP/HTTPS) oder AWS-SDKs, um Daten hochzuladen, abzurufen oder zu ändern.[78]

4.3.2 Amazon Aurora

Amazon Aurora ist ein relationaler Datenbankservice für MySQL und PostgreSQL und ist Teil von Amazons Relationalen Datenbank Service (RDS).[79] Dieses BaaS-Angebot wurde entwickelt, um deutlich höhere Leistung und Skalierbarkeit als herkömmliche relationale Datenbanken in der AWS-Cloud zu erreichen. Die Daten werden automatisch in einem verteilten Speichersystem über mehrere AZs repliziert, was eine hohe Ausfallsicherheit ermöglicht. Zusätzlich können Lesereplikate eingerichtet werden, um Lesezugriffe zu beschleunigen.[80]

[77] Vgl. Amazon S3: in: AWS Produkte, o. D., [online] https://aws.amazon.com/s3/ (abgerufen am 04.02.2025).
[78] Vgl. Amazon Web Services, Inc., 2024b, S. 142 f.; vgl. Amazon Web Services, Inc.: What is Amazon S3?, o. D.c, [PDF] https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html (abgerufen am 05.02.2025).
[79] Vgl. Amazon Aurora: in: AWS Produkte, o. D., [online] https://aws.amazon.com/rds/aurora/ (abgerufen am 04.02.2025).
[80] Vgl. Amazon Web Services, Inc., 2024b, S. 49 f.; vgl. Shrivastava et al., 2023, S. 234 ff.

4.3.3 Amazon DynamoDB

Amazon DynamoDB ist ein vollständig verwalteter, serverloser NoSQL-Datenbankservice, der hohe Skalierbarkeit und geringe Latenzen verspricht.[81] Als schlüsselwert- und dokumentenorientierte Datenbank kann DynamoDB schemalose Datenmodelle speichern. Im Hintergrund werden Daten automatisch über mehrere AZs repliziert, was ebenfalls eine hohe Verfügbarkeit sicherstellt. Der Dienst skaliert automatisch je nach Anfrageaufkommen und benötigt keine Infrastrukturverwaltung.[82]

4.3.4 Amazon ElastiCache

Amazon ElastiCache ist ein vollständig verwalteter In-Memory-Datenspeicherdienst, der die Open-Source Tools Redis und Memcached unterstützt.[83] Er ermöglicht es, Daten in einem Cache zwischenzuspeichern, um Datenbankanfragen zu beschleunigen und die Latenz innerhalb der Anwendungen zu reduzieren. Dadurch werden Anwendungen schneller und skalierbarer, da wiederholte Datenzugriffe direkt aus dem Cache erfolgen können. Typische Anwendungsfälle sind Caching, Session-Management, Echtzeit-Analysen und die Entlastung von Back-End-Datenbanken.[84]

[81] Vgl. Amazon DynamoDB: in: AWS Produkte, o. D., [online] https://aws.amazon.com/dynamodb/ (abgerufen am 04.02.2025).
[82] Vgl. Amazon Web Services, Inc., 2024b, S. 50.; vgl. Shrivastava et al., 2023, S. 240 f.
[83] Vgl. Amazon ElastiCache: in: AWS Produkte, o. D., [online] https://aws.amazon.com/elasticache/ (abgerufen am 04.02.2025).
[84] Vgl. Amazon Web Services, Inc., 2024b, S. 51.; vgl. Shrivastava et al., 2023, S. 248 ff.

4.4 Anwendungsintegration

4.4.1 Amazon Simple Notification Service (SNS)

Amazon SNS ist ein Publish/Subscribe-Ereignisdienst, der die Zustellung von Mitteilungen von produzierenden Diensten an konsumierende Dienste ermöglicht.[85] Anwendungen oder Dienste können Ereignisse an ein Thema senden, das als logischer Verteiler dient. Alle Dienste, die dieses Thema abonniert haben, erhalten dieses Ereignis. SNS unterstützt verschiedene Abonnement-Typen bzw. Endpunkte, darunter Warteschlangen, Lambda-Funktionen, HTTP/HTTPS-Webhooks, E-Mail, SMS und mobile Push-Benachrichtigungen. Dadurch kann ein einzelnes Ereignis automatisch und parallel an viele Empfänger verteilt werden.[86]

4.4.2 Amazon Simple Queue Service (SQS)

Amazon SQS ist ein Ereigniswarteschlangendienst, der die asynchrone Übermittlung von Ereignissen zwischen Softwarekomponenten ermöglicht.[87] SQS entkoppelt sendende und empfangende Dienste, indem Ereignisse zunächst in einer Warteschlange zwischengespeichert werden, bis der Empfänger sie verarbeitet. SQS bietet Standard-Queues mit hohem Durchsatz sowie FIFO-Queues für strikt geordnete, genau-einmalige Zustellung.[88]

[85] Vgl. Amazon Simple Notification Service: in: AWS Produkte, o. D., [online] https://aws.amazon.com/sns/ (abgerufen am 27.01.2025).
[86] Vgl. Amazon Web Services, Inc., 2024b, S. 23.; vgl. Shrivastava et al., 2023, S. 518 f.
[87] Vgl. Amazon Simple Queue Service: in: AWS Produkte, o. D., [online] https://aws.amazon.com/sqs/ (abgerufen am 27.01.2025).
[88] Vgl. Amazon Web Services, Inc., 2024b, S. 23.; vgl. Shrivastava et al., 2023, S. 518.

4.4.3 AWS Step Functions

AWS Step Functions ist ein serverloser Dienst, mit dem sich Abläufe über mehrere AWS-Dienste hinweg als zustandsbasierte State-Machine modellieren und ausführen lassen.[89] Entwickler definieren hierbei einen Ablauf als Reihe von Zuständen, die zum Beispiel sequenziell Lambda-Funktionen ausführen, bestimmte Wartezeiten einhalten oder Entscheidungen basierend auf Bedingungen treffen. Durch die Orchestrierung mit Step Functions können komplexere Prozesse umgesetzt werden, ohne dafür eine eigene Orchestrierungslogik im Programmcode schreiben zu müssen. Dies vereinfacht die Entwicklung und erhöht die Wartbarkeit verteilter Systeme.[90]

4.4.4 Amazon EventBridge

Amazon EventBridge ist ein serverloser Ereignisbus-Dienst, der Anwendungen und Dienste über Ereignisse lose koppelt.[91] Unterschiedliche Quellen können Ereignisse in einen EventBus einspeisen. Anhand von definierten Regeln filtert und leitet EventBridge diese Ereignisse an Zielsysteme weiter. Auf diese Weise ermöglicht EventBridge ereignisgesteuerte Architekturen, in denen ein Event in System A automatisch eine Reaktion in System B auslösen kann, ohne dass die Systeme direkt integriert sind. Im Vergleich zu SNS bietet EventBridge flexiblere Filter- und Routing-Möglichkeiten für komplexe Eventströme und eignet sich besonders, um in verteilten Systemen Ereignisquellen mit verschiedenen Zielen zu verbinden, während die einzelnen Komponenten entkoppelt und unabhängig skalierbar bleiben.[92]

[89] Vgl. AWS Step Functions: in: AWS Produkte, o. D., [online] https://aws.amazon.com/step-functions/ (abgerufen am 27.01.2025).
[90] Vgl. Amazon Web Services, Inc., 2024b, S. 21.
[91] Vgl. Amazon EventBridge: in: AWS Produkte, o. D., [online] https://aws.amazon.com/eventbridge/ (abgerufen am 28.01.2025).
[92] Vgl. Amazon Web Services, Inc., 2024b, S. 22.; vgl. Shrivastava et al., 2023, S. 324 f.

4.5 AWS-Architektursymbole

Abbildung 3 zeigt die von AWS bereitgestellten Architektursymbole für alle analysierte Dienste. Diese werden in allen folgenden Architekturdiagrammen verwendet und stammen aus dem Symbolpaket[93] der Version 20 vom 07.02.2025.

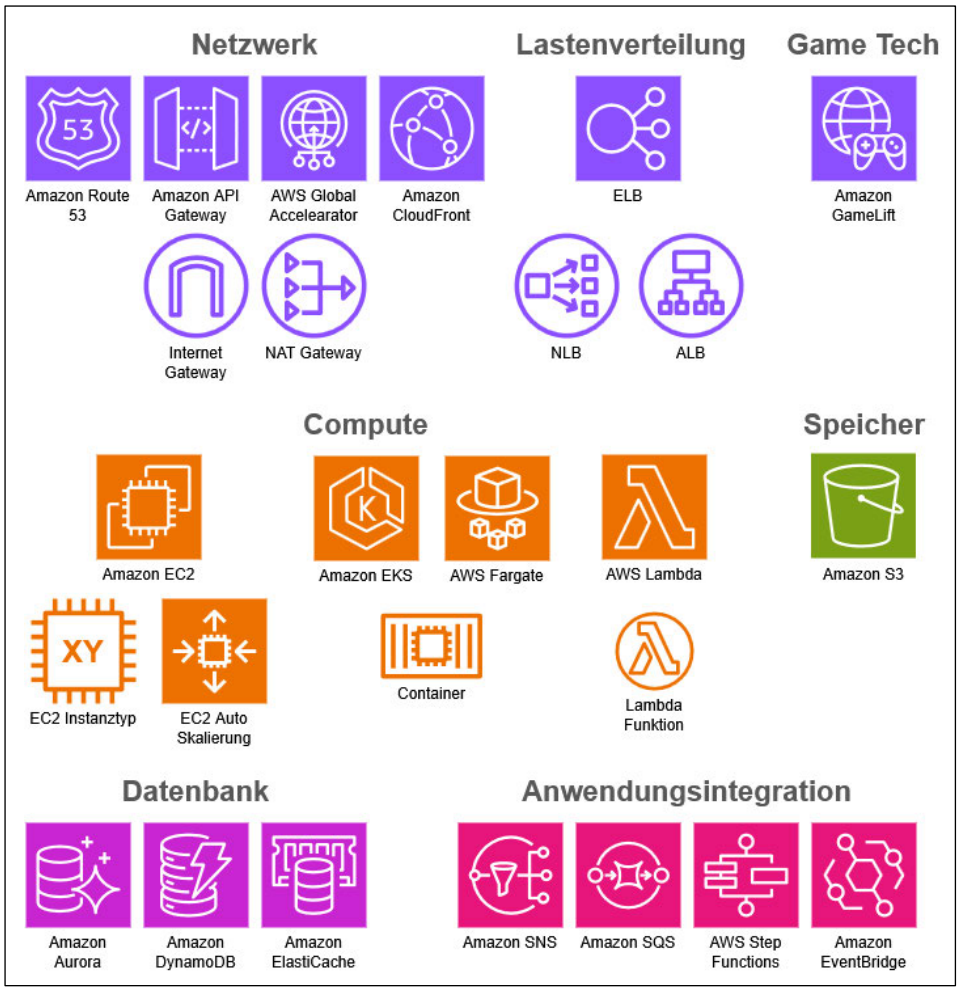


Abbildung 3: Gruppierte Architektursymbole und ihre dazugehörigen Dienste
Quelle: Eigene Darstellung

[93] AWS-Architektursymbolpaket: 07.02.2025, [online] https://d1.awsstatic.com/webteam/architecture-icons/q1-2025/Asset-Package_02072025.dee42cd0a6eaacc3da1ad9519579357fb546f803.zip.

5 Fallbeispiele aus der Spieleindustrie

Amazon zeigt auf seiner Webseite zahlreiche Erfolgsgeschichten seiner Kunden und deren Umstieg auf AWS. Darunter befinden sich viele Fallbeispiele aus der Spieleindustrie und auch wenige aus dem Teilbereich der asynchronen Multiplayer-Spiele. Dieses Kapitel untersucht anhand dieser Fallbeispiele, wie Implementierungen von AWS in der Spieleindustrie realisiert werden. Darauf folgen Analysen von Referenzarchitekturen aus Dokumentationen, Anleitungen und Entwickler-Blogs. Ziel ist es, aus beiden Perspektiven unterschiedliche und praxisnahe Architekturen für den Vergleich in Kapitel 7 zu erstellen.

5.1 AWS Erfolgsgeschichten

Bei den AWS Erfolgsgeschichten[94] ist zu beachten, dass diese in erster Linie dem Marketing dienen und daher einer offensichtlichen werblichen Ausrichtung und einer fehlenden Neutralität unterliegen. Aus diesem Grund eignen sie sich nur bedingt als Grundlage für eine fundierte bzw. wissenschaftliche Ableitung. Nichtsdestotrotz können die darin beschriebenen oder visuell dargestellten Systemarchitekturen zumindest für eine erste, oberflächliche Betrachtung herangezogen werden, da sie exemplarisch den praktischen Einsatz moderner Cloud-Technologien illustrieren und dabei gängige technische Standards, bewährte Entwurfsmuster und Strategien zur Skalierung von Systemen aufzeigen.

[94] Customer Success Stories: o. D., [online] https://aws.amazon.com/solutions/case-studies/ (abgerufen am 21.12.2024).

5.1.1 Clash of Clans

Der finnische Entwickler Supercell betreibt eines der erfolgreichsten asynchronen Mobile Games mithilfe von AWS. Clash of Clans erreicht jeden Monat über 200 Millionen Spieler. Um diese gewaltige Anzahl an Spielern zu bewältigen, setzt Supercell auf eine Microservice-Architektur mit mehr als 2.000 EC2-Instanzen und 300 Aurora-Datenbanken.[95]

Supercell nutzt Amazon Aurora, um Ausfallzeiten zu minimieren und einen durchgängigen Betrieb des Spiels zu gewährleisten. Ergänzend dazu wird Amazon DynamoDB verwendet, der aufgrund seiner hohen Skalierbarkeit und Geschwindigkeit die Verwaltung großer Datenmengen ermöglicht, die anschließend für analytische Prozesse genutzt werden können.[96] Latenzen werden mithilfe von Amazon CloudFront und AWS Global Accelerator niedrig gehalten. Spielspezifische Anfragen werden über einen ELB direkt an die EC2-Instanzen verteilt. Andere kleinere Prozesse werden mithilfe einer serverlosen Lösung über API Gateway und Lambda-Funktionen verarbeitet.[97]

Aus diesen Erfolgsgeschichten geht nicht hervor, ob Supercell einen containerbasierten Ansatz für seine Microservice-Architektur verwendet. Dennoch wird hier ein hybrider Ansatz betrieben. Einerseits kommen serverlose Dienste zum Einsatz, die eine flexible, ereignisbasierte Skalierung ermöglichen. Andererseits werden auch fest zugewiesene EC2-Instanzen genutzt, die die Basis für rechenintensive und kontinuierliche Spielprozesse bereitstellen.

[95] Vgl. Amazon Web Services, Inc.: How Supercell Runs Giant Games with Tiny Teams on AWS, 2018, [online] https://aws.amazon.com/solutions/case-studies/supercell-video/ (abgerufen am 09.01.2025); vgl. Amazon Web Services, Inc.: Supercell Leverages AWS for Seamless and Scalable Gaming Experience, 2024c, [online] https://aws.amazon.com/solutions/case-studies/supercell-aurora-case-study/ (abgerufen am 09.01.2025).
[96] Vgl. Amazon Web Services, Inc., 2018.; vgl. Amazon Web Services, Inc.: Supercell Case Study, 2014, [online] https://aws.amazon.com/solutions/case-studies/supercell/ (abgerufen am 09.01.2025).
[97] Vgl. Amazon Web Services, Inc.: Supercell | Edge Services | AWS Case Study, 2022, [online] https://aws.amazon.com/solutions/case-studies/supercell-edge-services-case-study/ (abgerufen am 09.01.2025).

5.1.2 Ludo King

Ludo King ist eine beliebte mobile Brettspiel-Plattform, die besonders in Indien hunderte Millionen Downloads erzielte. Der Entwickler Gametion hat früh auf AWS umgestellt, um mit dem schnellen Wachstum Schritt zu halten. Wie auch bei Clash of Clans wurde sich für eine Microservice-Architektur entschieden, deren Serverpower von EC2-Instanzen gestützt wird.[98]

Die Serverinfrastruktur basiert auf EC2 Instanzen, die dynamisch mit einem ELB skaliert werden können. Amazon ElastiCache wird eingesetzt, um durch Caching die Anfragen und Berechnungen schneller und flüssiger abarbeiten zu können. Große Datenmengen, wie Spielassets und Log-Dateien, werden in Amazon S3 gespeichert. Mit Amazon Route 53 und CloudFront wird der eingehende Traffic verteilt, sodass Nutzer stets eine optimale Verbindung zu den Servern erhalten.[99] Es wird dabei jedoch keine spezielle Datenbanklösung wie Amazon Aurora oder DynamoDB erwähnt, und es lassen sich auch keine Rückschlüsse darauf ziehen, wie die Spieleserver im Detail implementiert sind. Diese Erfolgsgeschichte fokussiert sich auf die grundlegenden Infrastrukturkomponenten

Sowohl Supercell als auch Gametion nutzen Datenstromdienste wie Amazon Kinesis und Amazon MSK[100], um Daten in Echtzeit für analytische Zwecke zu verarbeiten. Diese Lösungen ermöglichen es, Einblicke in das Nutzerverhalten und die Performance zu gewinnen, spielen jedoch keine direkte Rolle in der Spielinfrastruktur.[101] Daher wird nicht weiter auf die Details der Implementierung dieser Services eingegangen.

[98] Vgl. Amazon Web Services, Inc.: AWS case study - Gametion, 2020, [online] https://aws.amazon.com/solutions/case-studies/gametion/ (abgerufen am 11.01.2025).
[99] Vgl. ebd., Kap. AWS Services Used.
[100] Abk. für „Managed Streaming for Apache Kafka“.
[101] Vgl. Amazon Web Services, Inc., 2020; vgl. Amazon Web Services, Inc., 2014.

5.1.3 Whatwapp

Das italienische Studio Whatwapp migrierte sein Backend vollständig auf AWS und nutzt eine containerbasierte Lösung für seine asynchronen Mobile-Spiele. Zentraler Baustein ist EKS, der es ermöglicht, den Open-Source-Game-Server Nakama in einem Kubernetes-Cluster zu betreiben und so Multiplayer-Funktionalitäten wie Logins, Chats, Turniere und asynchrones Matchmaking zu verwalten. Amazon S3 wird zur Speicherung von Spieldaten und statischen Inhalten genutzt, während Amazon CloudFront als Content Delivery Network dafür sorgt, dass diese Inhalte weltweit schnell und zuverlässig ausgeliefert werden. Daten wie Spielerprofile und Spielstände werden über Amazon RDS verwaltet.[102]

Whatwapp wechselte zu AWS, um eine standardisierte und skalierbare Infrastruktur aufzubauen, die den stetig wachsenden Anforderungen ihrer Spiele gerecht wird. Dies hatte zur Folge, dass der Entwicklungsaufwand signifikant reduziert werden konnte. Gleichzeitig profitierte Whatwapp von der Kosteneffizienz und der schnelleren Bereitstellung von Updates.[103]

Diese Erfolgsgeschichte erwähnt nicht, welche Entwurfsmuster eingesetzt werden. Daher können keine Rückschlüsse auf eine Microservice-Architektur gezogen werden. Dennoch zeigt sie, dass man durch die Nutzung von Amazon EKS unabhängiger agieren kann und in der Lage ist, Open-Source-Tools innerhalb der AWS-Umgebung zu integrieren.

[102] Vgl. Amazon Web Services, Inc.: Delivering Engaging Games at Scale Using AWS with Whatwapp, 2023a, [online] https://aws.amazon.com/solutions/case-studies/whatwapp-case-study/ (abgerufen am 12.01.2025), Kap. Solution.
[103] Vgl. ebd., Kap. Opportunity.

5.1.4 Marvel Snap

Marvel Snap ist ein asynchrones Online-Deckbau-Spiel, entwickelt von dem Indie-Studio Second Dinner. Es setzt auf eine vollständig serverlose Architektur mit AWS Lambda. Amazon API Gateway fungiert als zentraler Einstiegspunkt, über den alle Spieleranfragen geleitet werden, während die eigentliche Spiellogik von vielen AWS Lambda-Funktionen übernommen wird. Im Kern steht hier eine ereignisgesteuerte Architektur, die mit komplett serverlosen AWS-Diensten realisiert wird.[104]

Zur Orchestrierung von Ereignissen und zur Koordination von Abläufen innerhalb der Architektur kommt Amazon EventBridge zum Einsatz. Die Datenspeicherung erfolgt über Amazon DynamoDB, der alle wichtigen Spielerdaten serverlos verwaltet. Für das Matchmaking greift Second Dinner auf Amazon GameLift FlexMatch zurück. Hier ist besonders interessant, dass nur der Matchmaking-Dienst von GameLift verwendet wird und keine GameLift-Instanz, wodurch die Architektur serverlos bleibt.[105]

Diese Architektur wurde so konzipiert, dass Second Dinner sich voll auf die Weiterentwicklung und Optimierung der Spielefeatures konzentrieren kann, ohne sich um den Betrieb der Infrastruktur kümmern zu müssen. Durch den Einsatz serverloser Funktionen werden Lastspitzen automatisiert bewältigt, während das komplett nutzungsbasierte Abrechnungsmodell sicherstellt, dass lediglich die in Anspruch genommenen Ressourcen bezahlt werden müssen.[106]

[104] Vgl. Amazon Web Services, Inc.: MARVEL SNAP scales player matchmaking with Amazon GameLift FlexMatch, 2023b, [online] https://aws.amazon.com/solutions/case-studies/second-dinner-nuverse-case-study/ (abgerufen am 12.01.2025), Kap. Solution.
[105] Vgl. ebd., Kap. Solution.
[106] Vgl. ebd., Kap. Opportunity.

5.2 Referenzarchitekturen

Referenzarchitekturen sind vorgefertigte, standardisierte Beispiellösungen, die bewährte Ansätze und Best Practices im Architekturen-Design veranschaulichen. Sie bieten eine neutrale und detaillierte Grundlage für die Implementierungen von verschiedenen Architekturlösungen. Im Folgenden werden drei Referenzarchitekturen im Hinblick auf asynchrone Multiplayer-Spiele analysiert.

5.2.1 Referenzarchitektur für asynchrones Online-Gaming

AWS stellt eine Referenzarchitektur für asynchrones Online-Gaming zu Verfügung.[107] Diese beschreibt eine klassische 3-Schichten-Architektur. Das Architekturdiagramm aus Abbildung 4 zeigt die gesamte Architektur und ihre verwendeten AWS-Dienste und deren Beziehungen zueinander.

AWS beschreibt diese Architektur als speziell für asynchrone Mobile- und Online-Spiele konzipiert, um die unvorhersehbaren Lastschwankungen und die hohen Anforderungen optimal zu bewältigen. Sie bietet eine grundlegende Basis, um Ressourcen flexibel an den Bedarf anzupassen. Bei steigendem Spieleraufkommen kann die Architektur erweitert und bei geringerer Nutzung wieder reduziert werden.

AWS behauptet, dass die verwendeten Dienste und Strategien sich an den Best Practices großer Spieleentwickler orientieren.[108] Diese Aussage konnte während der Analyse nicht belegt werden, da der Großteil der gefundenen realen Architekturen auf feinteiligeren Microservices basiert. Allerdings ist die Datenlage zu dünn, um diese Aussage zu widerlegen.

[107] vgl. Amazon Web Services, Inc.: Asynchronous-Online-Gaming - Basic, 24.03.2021a, [PDF] https://d1.awsstatic.com/architecture-diagrams/Asynchronous-Online-Gaming%20-%20Basic.pdf (abgerufen am 17.01.2025).
[108] Vgl. ebd., Beschreibung.

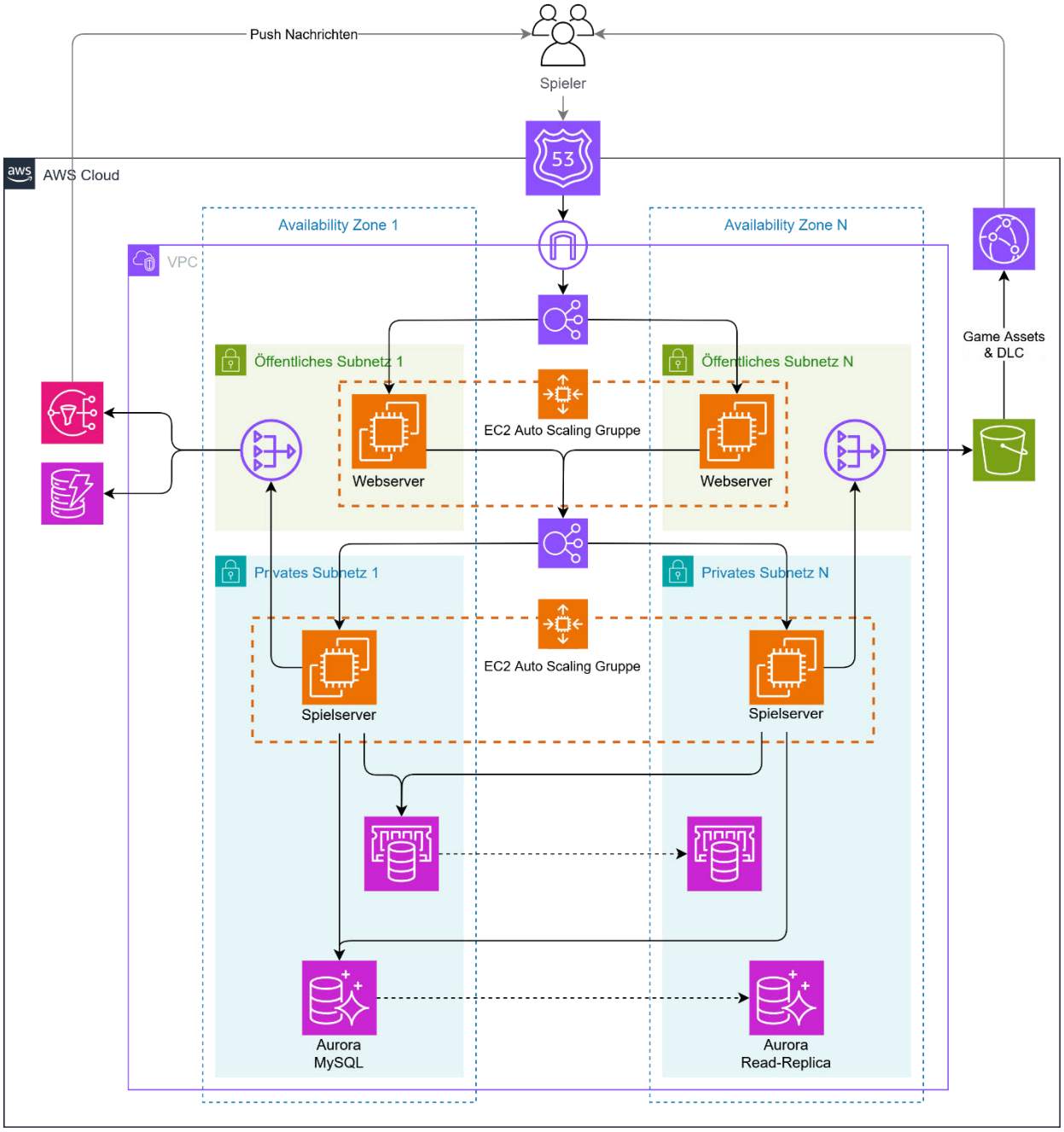


Abbildung 4: Referenzarchitektur für asynchrones Online-Gaming
Quelle: (Übersetzt und auf aktuellen Symbol-Standard aktualisiert) Anantha, Mounika: AWS Reference Architecture for Asynchronous Online Gaming, in: Medium, 31.10.2024, [online] https://medium.com/@ananthamounika123/aws-reference-architecture-for-asynchronous-online-gaming-b457fe763567 (abgerufen am 28.02.2025).

Beschreibung:

Zunächst laufen alle Anfragen über Amazon Route 53, der mithilfe von DNS-Routing die jeweils nächstgelegene AZ auswählt. Anschließend wird die Anfrage mithilfe ELB auf verschiedene Webserver verteilt, wodurch Überlastungen vermieden werden. Diese Webserver befinden sich in öffentlichen Subnetzen und sollen vor allem dazu genutzt werden, statische Inhalte bereitzustellen und Anfragen zu validieren. Das Bereitstellen der Spielserver in privaten Subnetzen bietet einen zusätzlichen Schutz gegenüber externen Angriffen. Beide Servertypen werden auf EC2 Instanzen ausgeführt und skaliert.[109]

Damit häufig abgefragte Daten schnell bereitgestellt werden können, nutzt das System einen Amazon ElastiCache als Zwischenspeicher. Die Datenspeicherung übernimmt eine Mischung von Amazon Aurora und Amazon DynamoDB. Amazon Aurora ist über mehrere Availability Zones verteilt, um bei Ausfällen einzelner Instanzen den Betrieb aufrechtzuerhalten.[110]

Um Spieler auf dem Laufenden zu halten, setzt das System auf Amazon SNS. Dadurch werden in dem Beispiel Benachrichtigungen zu neuen Levels oder Erfolgen direkt an die Nutzer versendet. Parallel dazu werden alle großen statischen Spiel-Assets, wie Grafiken oder Audiodateien, in Amazon S3 gespeichert und über das Amazon CloudFront CDN bereitgestellt. [111]

[109] vgl. Anantha, 2024, Architecture Flow Explanation 2 - 5.
[110] Vgl. ebd., Architecture Flow Explanation 6 - 8.
[111] vgl. ebd., Architecture Flow Explanation 9 -10.

5.2.2 Simple Trivia Service

Der Lösungs-Architekt Time Bruce erklärt in dem Blog-Beitrag „Building a serverless multi-player game that scales“ die Erstellung eines kleinen Quizspiels mit dem Namen Simple Trivia Service.[112] Darin werden sowohl Singleplayer- als auch Multiplayer-Mechaniken mithilfe von AWS-Diensten implementiert. Es wird zwar nicht erwähnt, dass es sich hierbei um asynchrone Multiplayer-Mechaniken handelt, aber durch den rundenbasierten Ansatz kann nur ein Spieler zeitgleich agieren, was eine asynchrone Mechanik beschreibt.

Das Architekturdiagramm in Abbildung 5 zeigt eine vereinfachte, neutrale Backendstruktur des Simple Trivia Service. Hier werden nicht wie im vorherigen Beispiel dauerhafte Server-Instanzen betrieben, stattdessen nutzt diese Architektur ereignisgesteuerte serverlose Dienste. Die Basis bilden AWS Lambda-Funktionen, die sämtliche Spiellogik in Form von Microservices umsetzen. Jede Lambda-Funktion wird dabei nur bei Bedarf durch bestimmte Ereignisse ausgelöst.

Bruce behauptet, dass eine serverlose Architektur den Weg von der Idee zur Markteinführung erheblich verkürzt, da sie den operativen Aufwand minimiert und eine schnellere Umsetzung ermöglicht. Außerdem senkt sie die Kosten, indem sie eine kleinteilige, bedarfsorientierte Ressourcennutzung erlaubt und so Überdimensionierung vermeidet. Für Entwickler bedeutet dies, dass sich diese mehr auf die Verbesserung des Gameplays und die Entwicklung der Inhalte konzentrieren können, was zu einer besseren Spielerfahrung und einem beschleunigten Produktrelease führt.[113]

[112] Vgl. Bruce, Tim: Building a serverless multi-player game that scales, in: AWS Compute Blog, 24.02.2021, https://aws.amazon.com/de/blogs/compute/building-a-serverless-multiplayer-game-that-scales/ (abgerufen am 13.01.2025).
[113] Vgl. ebd., Abs. 1.

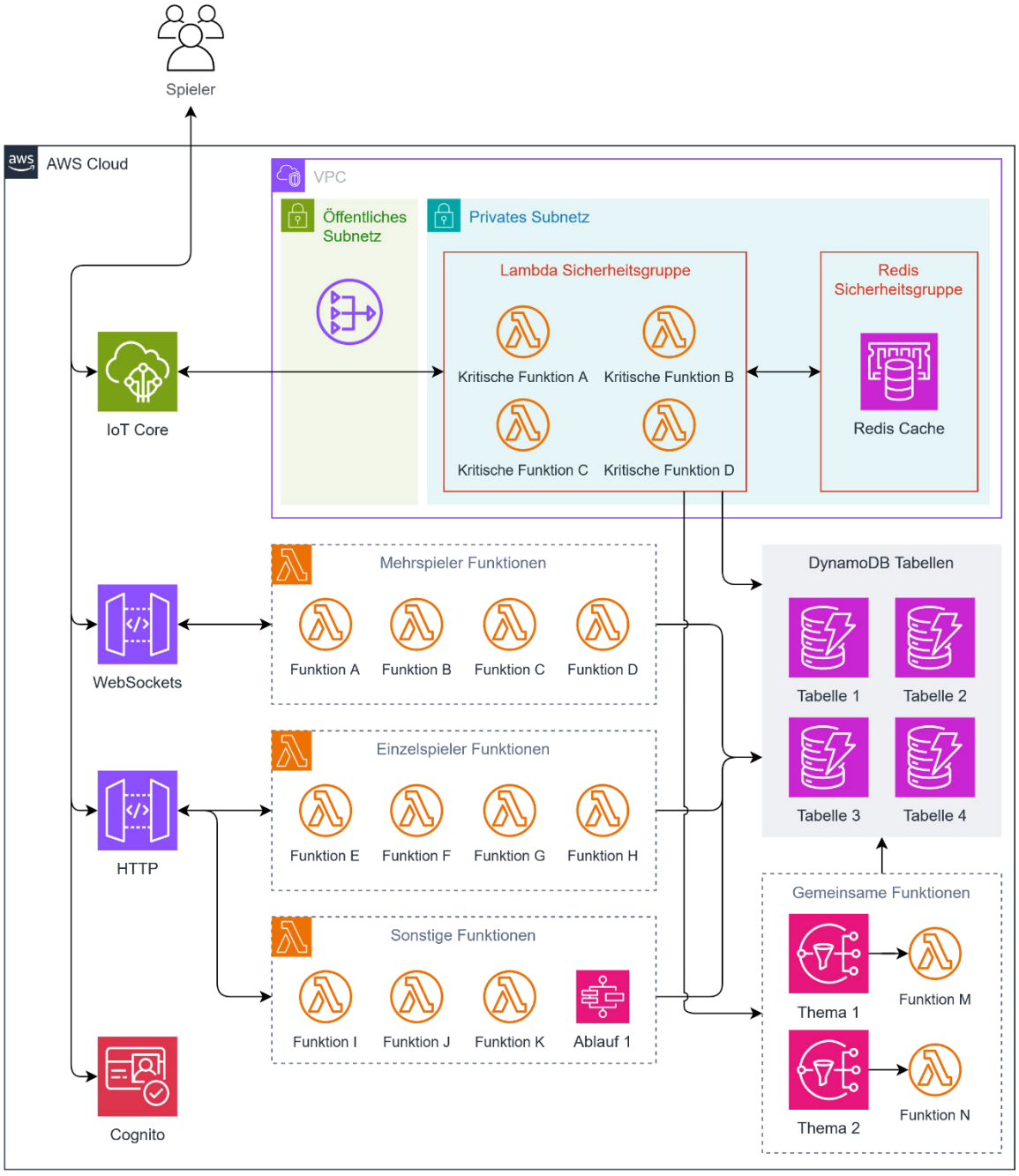


Abbildung 5: Vereinfachte Lösungs-Architektur des Simple Trivia Service
Quelle: (Übersetzt, vereinfacht und auf aktuellen Symbol-Standard aktualisiert) Beswick, 2021, Simple Trivia Service backend architecture.

Beschreibung:

Um REST-Anfragen und Socket-Kommunikation entgegenzunehmen, wird Amazon API Gateway als zentraler Einstiegspunkt eingesetzt. Zusätzlich kommt AWS IoT Core zum Einsatz, damit die Clients sich per MQTT-WebSocket verbinden können, um bestimmte Nachrichten-Themen abonnieren zu können. Daraus entsteht ein Publish/Subscribe-Modell für Echtzeit-Nachrichten. Auf diese Weise werden Quizfragen oder andere Spielinformationen gleichzeitig an mehrere Spieler verteilt. Für die dauerhafte Speicherung aller Daten wird der serverlose Dienst Amazon DynamoDB verwendet.[114]

Asynchrone Abläufe zwischen den unterschiedlichen Diensten werden über Amazon SNS koordiniert, der als internes Ereignis-System fungiert. Wenn etwa eine Spielrunde endet, kann ein Ereignis ausgelöst werden, das andere Lambda-Funktionen im Hintergrund startet, um Statistiken oder Spielerfortschritt zu aktualisieren. Dadurch müssen Spieler nicht auf die abschließende Datenverarbeitung warten. Komplexere Abläufe, die mehrere Lambda-Funktionen und Zwischenschritte umfassen, werden mittels AWS Step Functions orchestriert, um eine klare Trennung der Aufgaben und eine einfachere Wartung zu gewährleisten.[115]

Amazon Cognito bietet eine Nutzerverifizierung und ein Benutzerverzeichnis. Kritische Lambda-Funktionen werden in eine private VPC eingebunden, wodurch sie auf den nicht serverlosen Dienst Amazon ElastiCache zugreifen können. Dadurch benötigen sie jedoch ein NAT-Gateway, um auf externe Dienste wie DynamoDB zuzugreifen. Mit einer Sicherheitsgruppe lassen sich die Zugriffe der gruppierten Dienste einfacher konfigurieren.[116]

[114] Vgl. Bruce, 2021, Kap. 2 Services used, Kap. 4.3 “Multi-player – Live Scoreboard” architecture.
[115] Vgl. Bruce, 2021, Kap. 2 Services used, Kap. 4.1 “Single Player” quiz architecture.
[116] Vgl. Bruce, 2021, Kap. 2 Services used, Kap. 4.3 “Multi-player – Live Scoreboard” architecture.

5.2.3 Referenzarchitektur für containerbasierte Multiplayer-Backends

In der AWS-Dokumentation „Games Industry Lens - AWS Well-Architected Framework“ gibt es unter anderem eine Referenzarchitektur für Multiplayer-Spiele, die auf einem containerbasierten Microservice-Ansatz aufbaut.[117] Auch hier wird nicht explizit das Wort asynchron erwähnt, dennoch ist eine Nutzung dieser Architektur für asynchrone Multiplayer-Spiele möglich. Im Kern werden hier anstelle von Lambda-Funktionen, Microservices in EC2-Instanzen mittels Amazon EKS betrieben. Diese Architektur stellt Empfehlungen bereit, welche Dienste am besten für ein global verteiltes Game-Backend verwendet werden sollten.

Interessant ist bei dieser Architektur, dass die Dokumentation den Einsatz von serverlosen Verarbeitungsworkflows für Aufgaben im Hintergrund empfiehlt. Es ist also auch eine hybride Kombination aus serverlosen Diensten, wie AWS Lambda oder Amazon SNS/SQS und containerbasierten Diensten, wie Amazon EKS, sinnvoll. Dann werden aber nur kleinere und simplerer Aufgaben mit Lambda-Funktionen unterstützend erledigt. Der meiste Spielcode läuft dennoch auf Servern in EC2-Instanzen. Für langlaufende oder komplex koordinierte Workflows wird geraten, den gesamten Ablauf mit Step Functions zu orchestrieren, während Amazon EventBridge genutzt werden kann, um Funktionen als Reaktion auf AWS-Dienste oder benutzerdefinierte Ereignisse auszulösen.[118]

Das Architekturdiagramm in Abbildung 6 illustriert diese containerbasierte Referenzarchitektur. Es zeigt zudem anhand von vier beispielhaften Services, wie einzelne Container in solchen Diagrammen dargestellt werden können.

[117] Vgl. Amazon Web Services, Inc.: Games Industry Lens - AWS Well-Architected Framework, 19.11.2021b, [PDF] https://docs.aws.amazon.com/pdfs/wellarchitected/latest/games-industry-lens/games-industry-lens.pdf, S. 17 f.
[118] Vgl. ebd., S. 18, Abs. 6.

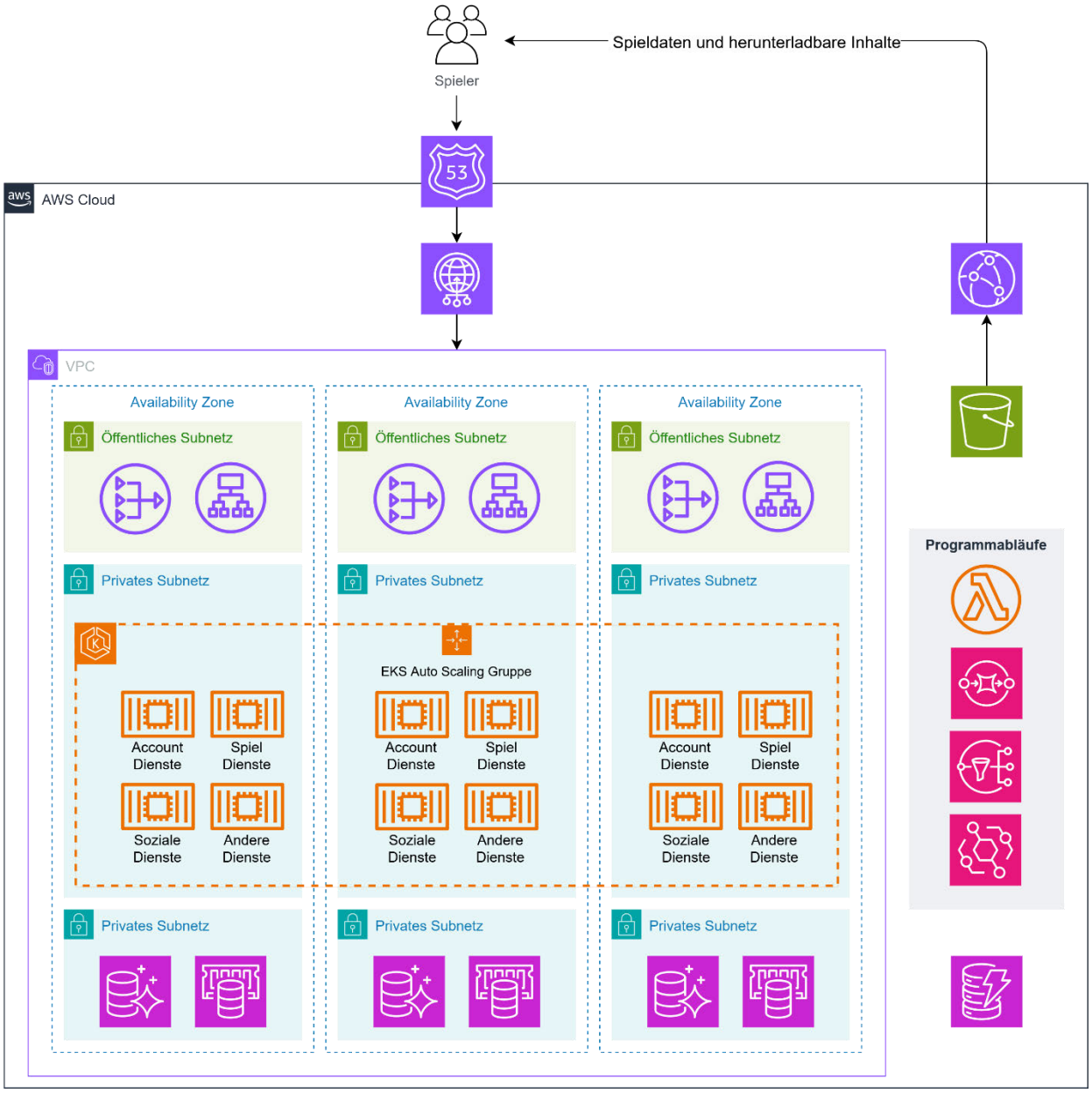


Abbildung 6: Referenzarchitektur für eine containerbasierte Multiplayer-Architektur
Quelle: (Übersetzt und auf aktuellen Symbol-Standard aktualisiert) Amazon Web Services, Inc., 2021b, S. 17.

Beschreibung:

Dieser Ansatz ist vergleichbar mit der in Abschnitt 5.2.1 dargestellten Referenzarchitektur, da beide Architekturen auf ähnlichen AWS-Diensten beruhen. Um die Vorteile der gemeinsam genutzten Dienste nicht zu wiederholen, folgt eine kurze Auflistung der Gemeinsamkeiten.

In beiden Architekturen werden Ressourcen über mehrere Availability Zones verteilt, wobei VPCs, öffentliche und private Subnetze sowie NAT-Gateways für sichere Verbindungen genutzt werden. Zudem kommen in beiden Architekturen Amazon Aurora und Amazon ElastiCache zum Einsatz. Außerdem werden Amazon CloudFront, Amazon S3 und Amazon Route 53 verwendet, um Inhalte weltweit effizient bereitzustellen und den Datenverkehr optimal weiterzuleiten. Im containerbasierten Ansatz wird zusätzlich AWS Global Accelerator eingesetzt, was den Vorteil bietet, dass die weltweiten Anfragen noch gezielter und mit geringerer Latenz an die jeweiligen Endpunkte weitergeleitet werden.[119]

Während im schichtenbasierten Modell klassische EC2-Instanzen zum Einsatz kommen, basiert der containerbasierte Ansatz auf Amazon EKS. Dies stellt einen wesentlichen Unterschied dar, der eine feinere Abstimmung der Skalierung ermöglicht. Die automatische Skalierung erfolgt nämlich in zwei Ebenen. Zunächst passt der Cluster-Autoscaler die Anzahl der Rechenknoten dynamisch an den aktuellen Bedarf an, und der Horizontal Pod Autoscaler sorgt dafür, dass die einzelnen Container je nach Auslastung skaliert werden. Auf diese Weise wird eine feingliedrigere Nutzung der vorhandenen Ressourcen erreicht.[120]

[119] Vgl. Amazon Web Services, Inc., 2021b, S. 17 f., Abs. 1 - 3, 5.
[120] Vgl. ebd., S. 18, Abs. 4.

5.3 Erkenntnisse

Die Analyse der Fallbeispiele und Referenzarchitekturen zeigt, dass heutige Architekturen für asynchrone Multiplayer-Spiele grundsätzlich auf einigen gemeinsamen Ansätzen beruhen. So wird in nahezu allen Ansätzen eine klare Trennung zwischen öffentlichen und privaten Subnetzen vorgenommen, um sicherheitsrelevante Aufgaben von der internen Spiellogik und den sensiblen Datenverarbeitungsprozessen zu kapseln. Während öffentliche Subnetze üblicherweise für den Empfang und der Validierung von Anfragen und den Zugriff auf statische Inhalte genutzt werden, werden in den privaten Subnetzen kritische Prozesse, wie die Verarbeitung von Spielerdaten und Transaktionen, isoliert und geschützt.

Ein weiteres zentrales Merkmal ist der Einsatz ereignisgesteuerter Lösungen in Kombination mit Microservices-Architekturen. Diese Architekturen nutzen Dienste wie AWS Lambda, EventBridge oder SNS/SQS, um Spielereignisse zu koordinieren. Durch die Aufteilung der Anwendungslogik in einzelne Microservices lassen sich Aufgaben modular abbilden und bedarfsgerecht skalieren. Dies führt zu einer hohen Flexibilität und ermöglicht, dass einzelne Komponenten unabhängig voneinander angepasst oder erweitert werden können. Interessanterweise wählt die Referenzarchitektur für asynchrones Online-Gaming einen anderen Ansatz. Anstatt auf eine feingliedrige Microservices-Struktur zu setzen, wird hier ein klassisches Drei-Schichten-Modell verfolgt, bei dem der gesamte Datenverkehr und die Spiellogik über zentrale Web- und Applikationsserver abgewickelt werden.

Zudem fällt auf, dass kleinere Projekte wie Simple Trivia Service oder Indie Studios, wie das von Marvel Snap, serverlose Ansätze bevorzugen. Da diese Lösungen besonders geeignet sind für Anwendungen, bei denen der Betrieb und die Skalierung einzelner Komponenten automatisiert erfolgen kann, ohne dass ein hoher Verwaltungsaufwand entsteht.

Im Gegensatz dazu nutzen die großen Spielestudios (Supercell, Gametion und Whatwapp), die Millionen von Spielern pro Monat managen müssen, nur serverlose Anwendungen zur Koordination von Abläufen. Es werden vielmehr eigene Implementierungen verwendet, die auf EC2-Instanzen aufbauen. Dabei wurde nur bei Whatwapp erwähnt, dass es sich hier um einen containerbasierten Ansatz handelt, der der Referenzarchitektur für eine containerbasierte Multiplayer-Architektur ähnelt.

Interessant ist auch die Abwesenheit eines erwarteten Dienstes in allen Architekturen. Der speziell für Gaming konzipierte Dienst Amazon GameLift findet keine Verwendung als Spieleserver in asynchronen Multiplayer-Architekturen. Es wird vermutet, dass GameLift-Server primär für den Einsatz in MOBAs[121] und First-Person-Shooter entwickelt worden, in denen extreme Leistungsfähigkeit und niedrige Latenzzeiten essenziell sind. In asynchronen Multiplayer-Spielen, bei denen das Spielgeschehen nicht in Echtzeit abläuft, wäre ein solcher Ansatz überdimensioniert. Zur Überprüfung der Vermutung wird eine auf GameLift basierende Architektur in den Vergleich einbezogen (siehe Kapitel 6.4).

Insgesamt lassen sich aus dieser Analyse verschiedene praxisnahe Lösungen ableiten, die sich für die Verwendung in einem Vergleich von asynchronen Multiplayer-Architekturen eignen.

[121] Engl. Abk. für Multiplayer Online Battle Arena – übersetzt „Mehrspieler-Online-Kampfarena“

6 Die vier Architekturen

In diesem Kapitel werden vier Architekturen vorgestellt, die aus der Analyse des vorangegangenen Kapitels abgeleitet wurden. Dabei werden drei, im Kern unterschiedliche, praxisnahe Lösungen beschrieben und anhand von Architekturdiagrammen illustriert. Es wurden ein klassischer Ansatz, ein serverloser Ansatz sowie ein containerbasierter Ansatz herausgearbeitet. Die Auswahl dieser Modelle wird in den folgenden Unterkapiteln ausführlich begründet. Zur kritischen Reflexion der Architekturentscheidungen wird zudem ein negatives Beispiel verwendet, um im anschließenden Vergleich die Folgen der Nutzung einer weniger optimalen Architektur besser darzustellen.

Um einen fairen Vergleich zu ermöglichen, ist eine gemeinsame Grundlage erforderlich. Daher werden für jede Architektur ausschließlich die minimal erforderlichen Dienste berücksichtigt, die notwendig sind, um sie als solche zu definieren. Andere Dienste, die die Architektur lediglich erweitern, wie Amazon CloudWatch (Überwachung), Amazon WAF (Firewall), Amazon ElastiCache (Zwischenspeicher) oder Ähnliches, werden nicht verwendet. Zudem beschränkt sich die Darstellung der Architekturen auf zwei AZs in einer Region, um diese übersichtlicher zu halten.

Durch die Vielzahl an AWS-Diensten ergeben sich oft alternative Lösungen, die im Wesentlichen denselben Ansatz verfolgen. Dies führt zu zahlreichen Architekturvarianten, die sich jedoch kaum voneinander unterscheiden. Um den Vergleich nicht durch zu kleinteilige Lösungen zu verwässern, wird auf mögliche Alternativen zum Austausch einzelner AWS-Dienste erst im Kapitel 8.3 genauer eingegangen.

6.1 Die klassische Architektur

Die erste Architektur, folgend als A1 abgekürzt, wird primär aus der AWS-Referenzarchitektur für asynchrone Online-Games (siehe Kapitel 5.2.1) abgeleitet. Sie wurde in keinem der Erfolgsgeschichten als Lösung verwendet, stellt aber dennoch ein klassisches Ausgangsmodell dar und sollte deshalb im Vergleich mitberücksichtigt werden. Diese Architektur wurde gewählt, da sie Entwicklern einen ersten und klar strukturierten Überblick bietet und von AWS selbst als eine sinnvolle Referenzarchitektur für asynchrone Online-Games beschrieben wird. Die abgeleitete vereinfachte Umsetzung der AWS-Referenzarchitektur ermöglicht einen Vergleich mit den anderen eher moderneren Lösungen.

Abbildung 7 zeigt die Verwendung von A1 auf zwei AZs. Jede Zone ist in ein öffentliches und ein privates Subnetz unterteilt. Darin wird jeweils eine EC2-T3-Instanz bereitgestellt, die für allgemeine Zwecke und niedrige Kosten optimiert ist. Die darin betriebenen Web- und Spieleserver werden automatisch skaliert. Als Datenspeicher wird eine Aurora MySQL Datenbank im privaten Subnetz verwendet.

Der Durchlauf der Anfragen ist sehr ähnlich zu der Referenzarchitektur und beginnt damit, dass Route 53 die Anfragen auf die bestmögliche Region weiterleitet. Danach werden sie von einem ALB auf die öffentlichen Webserver verteilt, die Anfragen validieren oder direkt beantworten. Spielspezifische Anfragen werden dann über einen weiteren ALB auf die Spieleserver verteilt. Diese können, falls es notwendig ist, Daten mittels der Aurora Datenbank bearbeiten oder über das NAT-Gateway eigene Verbindungen zum Internet herstellen. Um Daten über mehrere AZs synchron zu halten, verwendet diese Architektur ein Aurora Cluster, das schreibgeschützte Replikate auf anderen AZs bereitstellt. Dadurch können schnellere Lesezugriffe gewährleistet werden. Schreibzugriffe müssen dennoch auf der primären Aurora-Datenbank erfolgen.

6.1.1 Architekturdiagramm

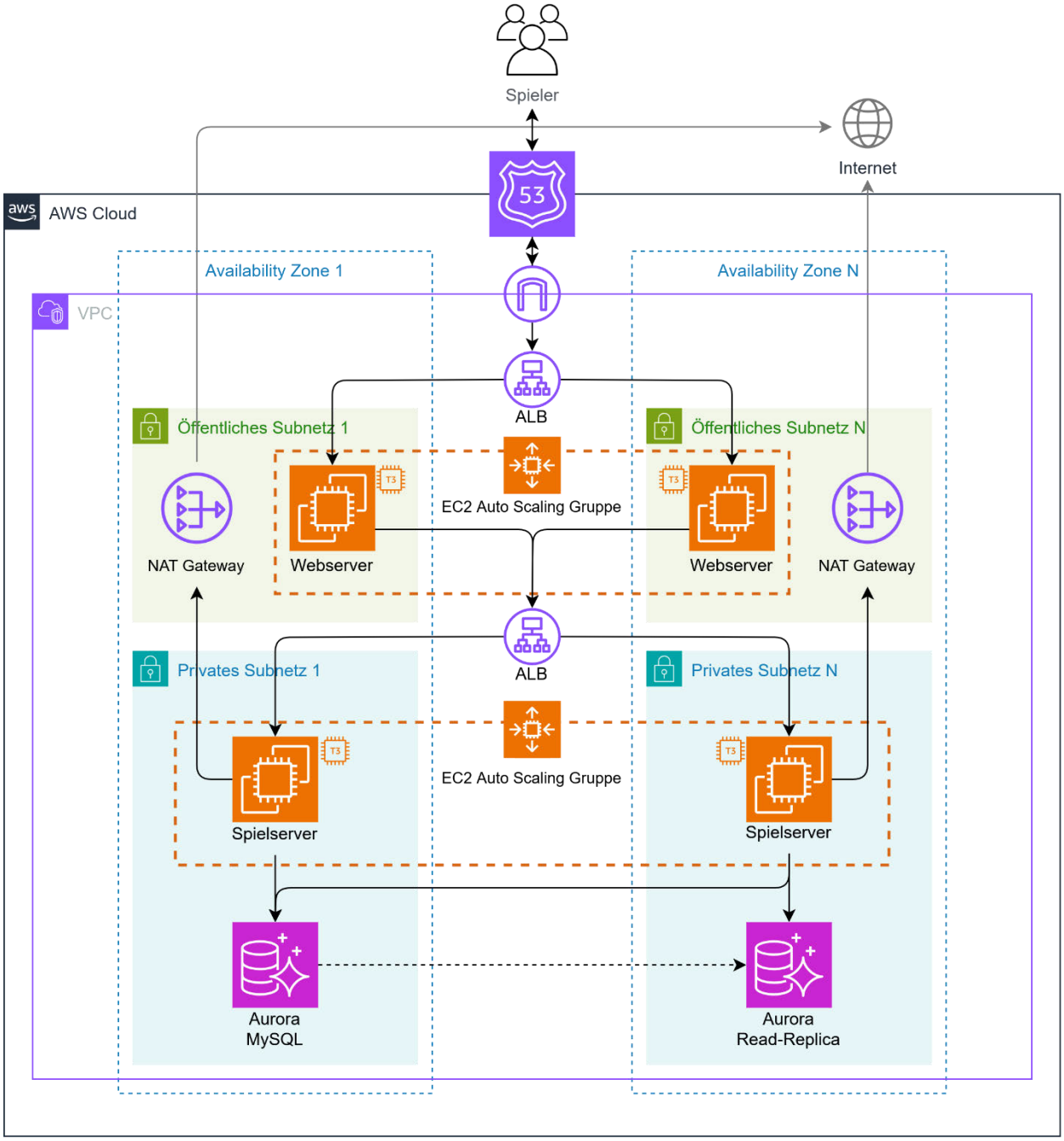


Abbildung 7: Vergleichbare klassische Drei-Schichten-Architektur
Quelle: Eigene Darstellung

6.2 Die serverlose Architektur

Die zweite Architektur, folgend als A2 abgekürzt, setzt eine komplett serverlose Lösung um. Dieser Ansatz sticht durch einen kleinen Betriebsaufwand und einer ereignisgesteuerten Skalierung hervor. Basierend auf den Erkenntnissen, dass kleinere Projekte wie Simple Trivia Service (siehe Kapitel 5.2.2) Service und Marvel Snap (siehe Kapitel 5.1.4) erfolgreich AWS Lambda verwenden, soll diese Architektur die Vorteile der Realisierung modularer Microservices in einer Umgebung ohne permanente Server aufzeigen.

A2 hat das übersichtlichste Architekturdiagramm, da weniger Services genutzt und keine AZs benötigt werden. Deshalb werden für den Ablauf verschiedene Möglichkeiten einer Orchestrierung dargestellt. Anfragen der Spieler werden vom Amazon API Gateway entgegengenommen. Dieses wandelt die Anfragen in Ereignisse um, die dann regelbasiert weitergeleitet werden können.

Lambda-Funktionen reagieren auf diese Ereignisse direkt oder können mittels AWS Step Functions, Amazon SNS und Amazon SQS transparenter orchestriert werden. Als Datenbank dient der serverlose NoSQL Dienst DynamoDB. Lambda-Funktionen oder Änderungen an den Daten können weitere Ereignisse auslösen, auf die wiederum andere Funktionen reagieren. So entsteht eine ereignisbasierte serverlose Microservice-Architektur.

6.2.1 Architekturdiagramm

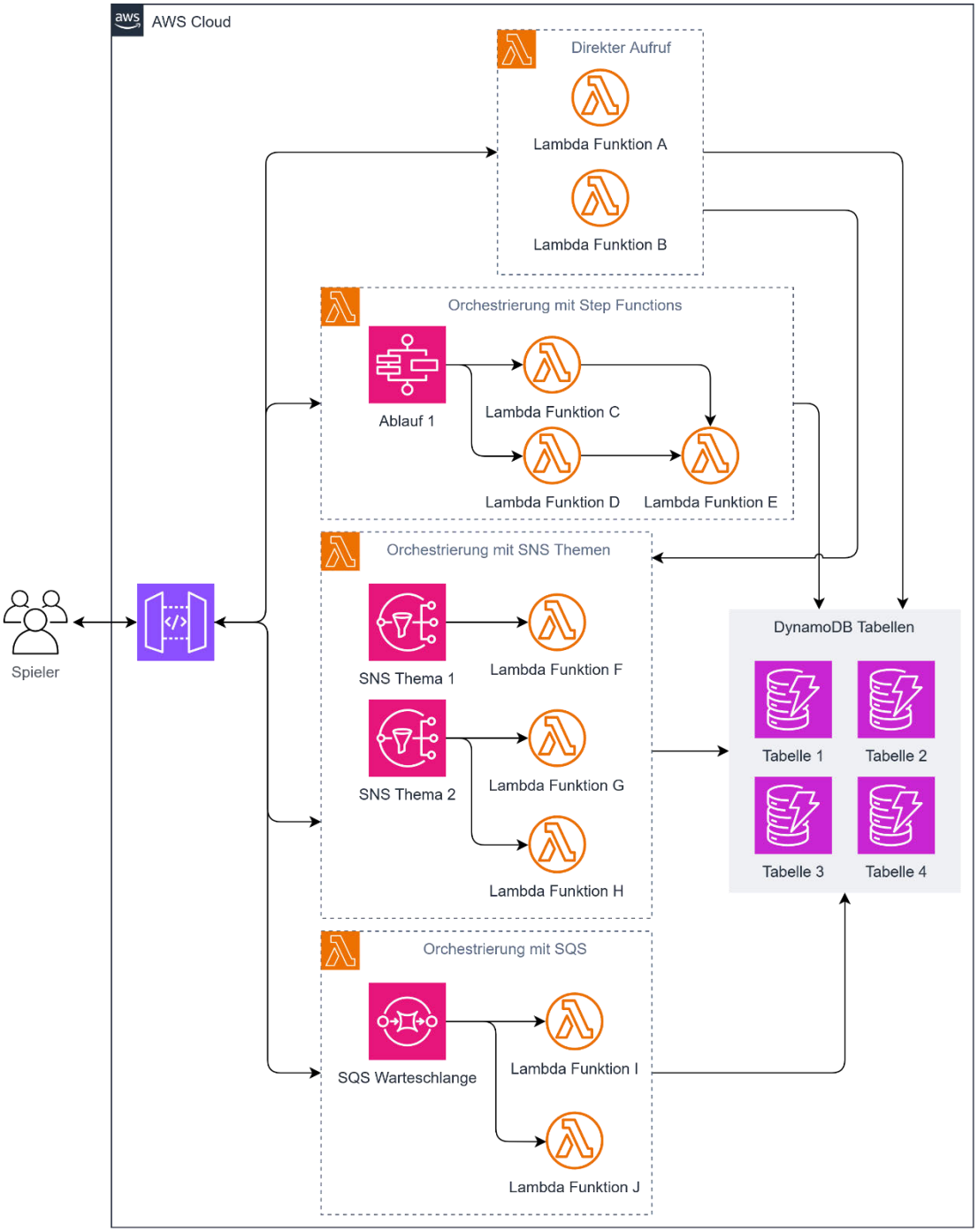


Abbildung 8: Vergleichbare serverlose Microservice-Architektur
Quelle: Eigene Darstellung

6.3 Die containerbasierte Architektur

Die dritte Architektur, folgend als A3 abgekürzt, baut auf einem containerbasierten Ansatz auf. Sie basiert auf den Erkenntnissen der großen Spielestudios (siehe Kapitel 5.1.1 - 5.1.3). Sie vereint diese mit den Grundlagen der Referenzarchitektur aus Kapitel 5.2.3. Diese Architektur soll die Vorteile der asynchronen Multiplayer-Entwicklung mithilfe von Containern als Grundlage für eine Microservice-Lösung aufzeigen.

Die Container werden von dem Dienst Amazon EKS verwaltet, der sie automatisch skaliert und überwacht. In einem EKS-Cluster werden 4 Container betrieben, in denen jeweils mehrere Spiel-Services auf einer EC2-T3-Instanz laufen. Die Daten werden in einer Aurora MySQL Datenbank hinterlegt und per Read-Replica auf die andern AZ verteilt.

Route 53 leitet die Anfragen der Spieler auf die bestmögliche Region weiter. Danach werden sie von einem ALB auf die AZs verteilt und an Amazon EventBridge gesendet. EventBridge dient als zentraler, serverloser Event-Bus, der Ereignisse von verschiedenen Quellen sammelt, filtert und an definierte Services weiterleitet. Zusätzlich wird Amazon SNS für schnellere und skalierbarere Verteilung von Ereignissen an mehrere Services eingesetzt. Zusammen bilden sie ein ereignisbasiertes System für die Orchestrierung der Microservices. EKS und Aurora befinden sich in einem privaten Subnetz. Alle Services können mithilfe eines NAT-Gateways, das sich in einem öffentlichen Subnetz befindet, mit dem Internet kommunizieren.

6.3.1 Architekturdiagramm

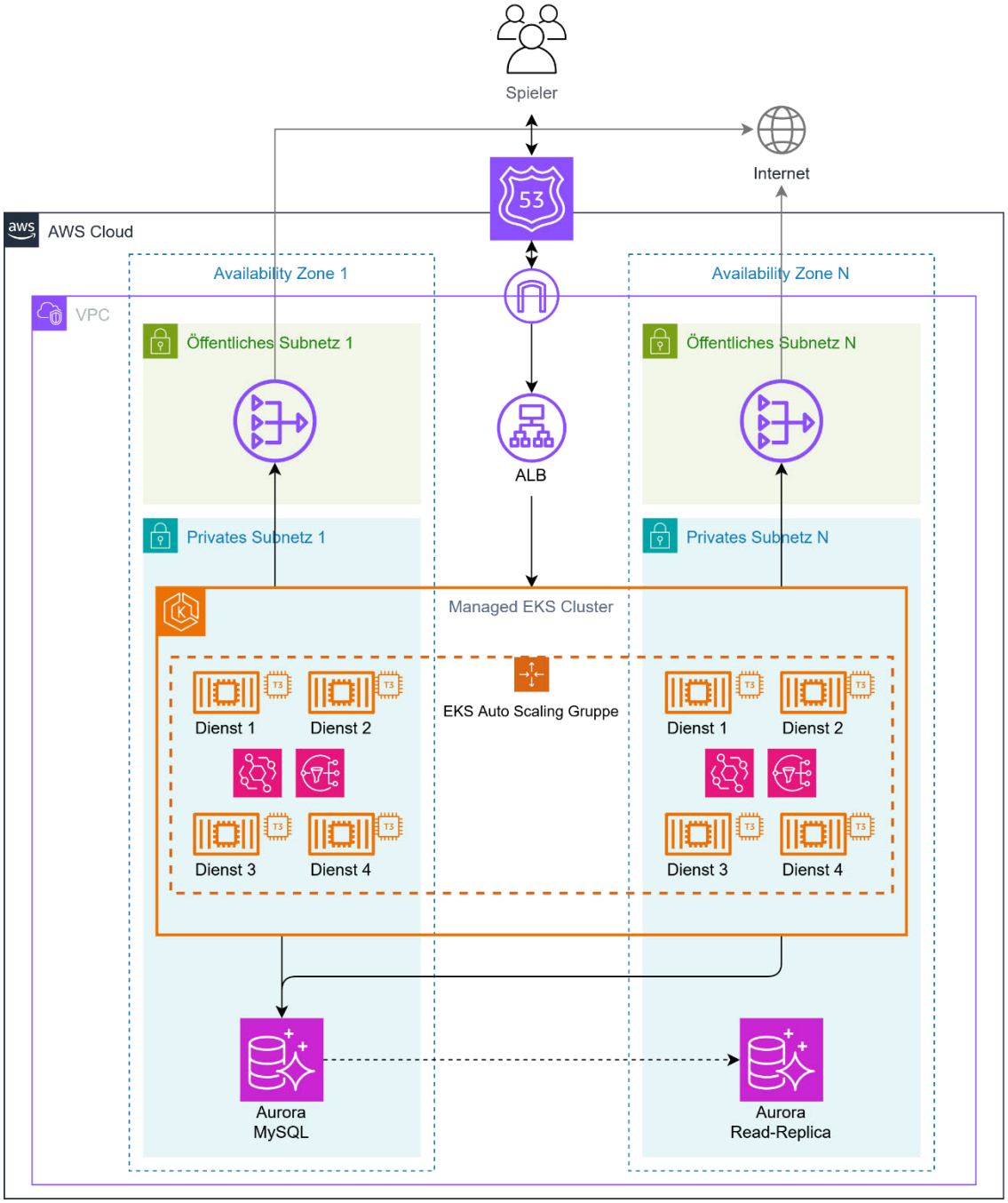


Abbildung 9: Vergleichbare containerbasierte Microservice-Architektur
Quelle: Eigene Darstellung

6.4 Die spezialisierte Architektur

Die vierte und letzte Architektur, folgend als A4 abgekürzt, integriert Amazon GameLift in Kombination mit einem serverlosen Backend. Dieser Ansatz ist relevant für Multiplayer-Spiele, die eine große Anzahl von Spielern in Sessions auf dauerhaften Servern gleichzeitig bewältigen müssen. Auch wenn asynchrone Spiele meist ohne Sessions und dauerhafte Server auskommen, könnte GameLift in Szenarien eingesetzt werden, in denen ein zukünftiger Umstieg auf synchrone Mechaniken nicht ausgeschlossen werden kann. Dennoch ist diese Architektur für den Einsatz ungeeignet und wird im Vergleich als Negativbeispiel verwendet, um die Auswirkungen der Verwendung einer falsch ausgewählten Architektur besser herausarbeiten zu können.

6.4.1 Architekturdiagramm

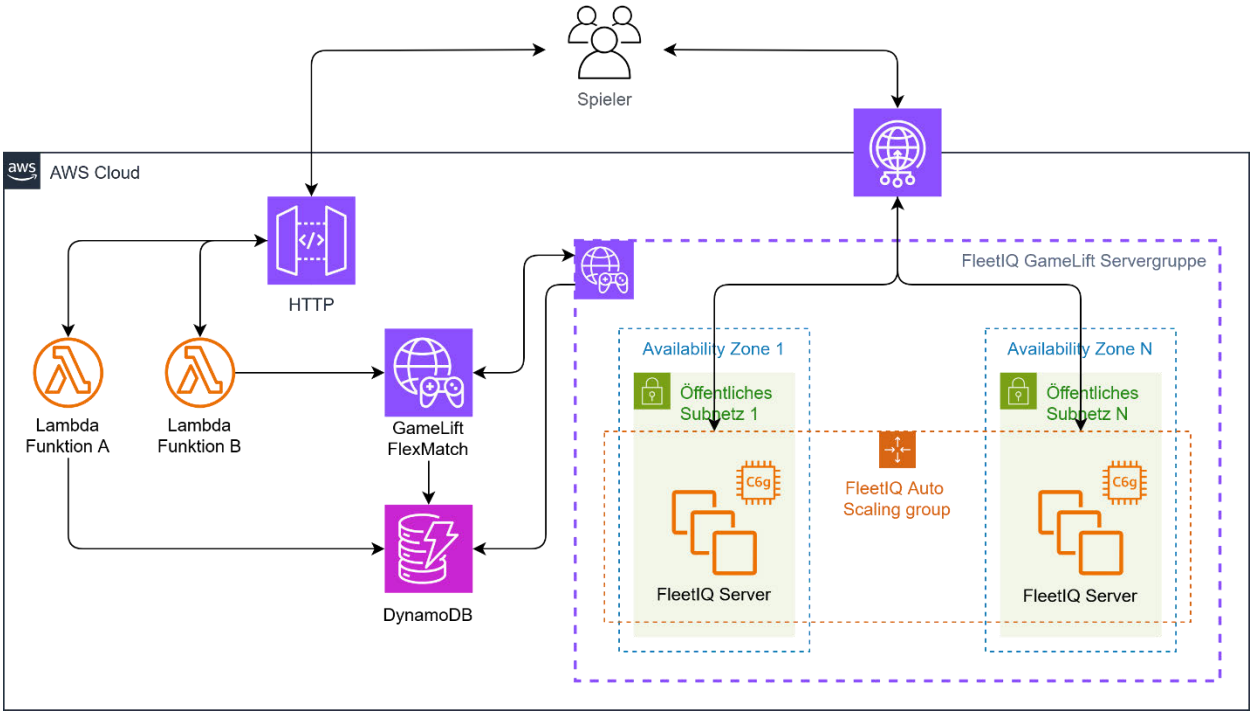


Abbildung 10: Vergleichbare GameLift-Architektur
Quelle: Eigene Darstellung

Amazon GameLift übernimmt in dieser Architektur das automatisierte Management einer GameLift Fleet von EC2-C6g-Instanzen, auf denen der Spielserver ausgeführt wird. Für jede Partie oder Lobby wird GameLift entweder eine neue Server-Instanz starten oder einem bereits laufenden Spielprozess weitere Spieler zuweisen. Mithilfe des integrierten Matchmaking-Systems FlexMatch lassen sich automatisch passende Spielergruppen bilden. Dabei wird die Anzahl der aktiven Serverinstanzen entsprechend dem Bedarf hoch- oder heruntergefahren, um sowohl Kosteneffizienz als auch Skalierbarkeit sicherzustellen.

Parallel zu GameLift wird ein serverloses Backend eingesetzt. Dieses besteht aus Amazon DynamoDB zur Speicherung von Spielerprofilen und Matchmaking-Daten sowie Amazon API Gateway für die Bereitstellung der API-Endpunkte. AWS Lambda übernimmt nicht wie in A2 die komplette Spiellogik, sondern die Ausführung der zugehörigen Anwendungslogik, wie zum Beispiel Matchmaking-Anfragen oder Änderungen an Spielerstatistiken.

Der exemplarische Ablauf beginnt damit, dass sich Spieler einen Request zum Beitritt in ein Match stellen. Eine Lambda-Funktion legt daraufhin ein Matchmaking-Ticket in DynamoDB an und ruft die GameLift-FlexMatch-API auf. Sobald eine ausreichende Anzahl passender Spieler gefunden ist, startet GameLift eine neue Spielsession, wobei der bereitgestellte Spielserver automatisch auf die EC2-Instanzen verteilt wird. Die Client-Seite erhält anschließend eine IP-Adresse und einen Port, um sich direkt mit dem Spielserver zu verbinden. Gleichzeitig benachrichtigt das Backend andere Lambda-Funktionen oder aktualisiert die Datenbank. Während des Spiels werden Spielzüge in Echtzeit zwischen Client und GameLift-Server ausgetauscht. Nach Ende der Partie meldet der GameLift-Server die Ergebnisse an eine Lambda-Funktion, die schlussendlich in DynamoDB gespeichert werden.

7 Vergleich

Alle vier Architekturen haben ihre individuellen Vor- und Nachteile. Um diese differenziert herauszuarbeiten, werden die analysierten Architekturen im Folgenden auf ihre qualitativen und quantitativen Merkmale verglichen. Tabelle 1 zeigt eine Aufzählung aller verwendeten AWS-Dienste jeder Lösungs-Architektur. Diese Auflistung ist insbesondere für den Vergleich der Betriebskosten in Abschnitt 7.3 wichtig.

Tabelle 1: Aufzählung der verwendeten AWS-Dienste der vier abgeleiteten Lösungs-Architekturen
Quelle: Eigene Darstellung

Architektur 1 (Klassisch)	Architektur 2 (Serverlos)
- Amazon Route 53 - Amazon VPC NAT-Gateway (x2) - Amazon ELB (x2) - Amazon EC2 T3 Instanzen (x4) - Amazon Aurora DB (x1 + Read-Replica)	- Amazon API Gateway - AWS Lambda-Funktionen (x20) - AWS Step Functions - Amazon SNS - Amazon SQS - Amazon Dynamo DB
Architektur 3 (Container)	**Architektur 4 (GameLift)**
- Amazon Route 53 - Amazon VPC NAT-Gateway (x2) - Amazon ELB - Amazon EKS Cluster - Amazon EC2 T3 Instanzen (x4) - Amazon EventBridge - Amazon SNS - Amazon Aurora DB (x1 + Read-Replica)	- Amazon API Gateway - AWS Global Accelerator - Amazon GameLift Server - Amazon GameLift FlexMatch - AWS Lambda-Funktionen (x2) - Amazon Dynamo DB

7.1 Implementierung und Komplexität

Eine einfache Implementierung einer Architektur spart viel Zeit, die dann stattdessen für die Umsetzung des Projekts verwendet werden kann. Die untersuchten Architekturen erfordern jeweils ein unterschiedliches Maß an Fachwissen, was zu einer Implementierungshürde werden kann, da zunächst das erforderliche Wissen aufgebaut werden muss. Je mehr Funktionalitäten die Architektur dem Entwickler abnimmt, desto schneller kann diese implementiert werden, was zu Kosteneinsparungen führen kann. Zudem muss eine Architektur, die in Produktion geht, auch gewartet werden. Je komplexer die Architektur, desto schwieriger ist sie zu warten. Die Komplexität einer Architektur steigt mit der Anzahl der verwendeten Services, die untereinander kommunizieren müssen. Damit alles reibungslos abläuft, müssen diese Services regelmäßig gewartet werden, was viel Zeit in Anspruch nimmt. Eine einfache Architektur oder eine Lösung, die Abläufe automatisiert, kann somit ebenfalls erheblich Zeit einsparen. Im Folgenden werden die vier abgeleiteten Architekturen hinsichtlich ihrer Implementierungsdauer und Wartungskomplexität bewertet und verglichen.

7.1.1 Begründete Einschätzung

Bei der Recherche wurde schnell klar, dass serverlose Architekturen, wie sie in A2 eingesetzt werden, hinsichtlich der Implementierungsdauer als die schnellste Variante gelten.[122] Durch die Nutzung der verwalteten Dienste AWS Lambda und API Gateway entfällt viel initialer Konfigurationsaufwand. Es muss nicht wie in A1 ein eigenes Netzwerkrouting erstellt oder wie in A3 eigene Container-Deployment-Pipelines entworfen werden. Zudem umgeht der serverlose Ansatz die Verwaltung von Serverinstanzen komplett.

[122] Vgl. Li et al., 2022, Kap. 1, 2.1; vgl. Shrivastava/Srivastav, 2024, S. 189; vgl. Shrivastava et al., 2023, S. 210 f; vgl. Bruce, 2021, Abs. 1.

Der Wartungsaufwand in A2 und A3 steigt mit der Anzahl der Microservices. Bei einer großen Anzahl muss die Architektur sehr gut dokumentiert werden, um alle Beziehungen schnell überblicken zu können. Das Debuggen solcher Architekturen ist sehr aufwendig und kostet in der Regel viel Zeit, weil die Suche nach Fehlerquellen in einem verteilten System deutlich komplexer ist.[123]

A1 und A3 sind in ihrer Implementierungsdauer ähnlich, da in beiden Fällen VPCs und Subnetze konfiguriert werden müssen. Die Vor- und Nachteile dieser beiden Architekturen lassen sich daher auf das zugrundeliegende System reduzieren, es handelt sich um einen Vergleich zwischen einem Drei-Schichten-System und einem verteilten System. Da die Anwendungslogik eines Drei-Schichten-Systems in zwei Anwendungen zusammengefasst ist, können Änderungen an den Kernfunktionen einfacher und einheitlicher umgesetzt werden. Allerdings betreffen diese Änderungen oft die gesamte Codebasis, was zu längeren Entwicklungszyklen und komplexeren Deployments führt.[124]

In einem verteilten System können Entwickler einzelne Dienste unabhängig voneinander entwickeln, testen und deployen. Neue Features können als eigenständiger Service eingeführt werden, ohne dass das gesamte System betroffen ist. Dies führt jedoch zu einem höheren operativen Aufwand und einer größeren Implementierungshürde, da eine Vielzahl von Microservices verwaltet werden muss und das Konzept der Containerisierung beherrscht werden muss. Wie auch in A2 werden hier Integrationstests und das Debugging komplexer, da Fehlerquellen über mehrere Services verstreut sind.[125]

[123] vgl. Velepucha, Victor/Pamela Flores: A Survey on Microservices Architecture: Principles, Patterns and Migration Challenges, in: IEEE Access, Bd. 11, 01.01.2023, doi:10.1109/access.2023.3305687, Abschn. 4.D; vgl. Shrivastava/Srivastav, 2024, S. 151.
[124] vgl. Velepucha/Flores, 2023, Abschn. 4.A.
[125] vgl. Velepucha/Flores, 2023, Abschn. 4.B, 4.C; vgl. Shrivastava et al., 2023, S. 467.

Die Umsetzung von A4 ist weniger aufwendig, da AWS GameLift viele Schritte der Implementierung automatisiert. Dadurch kann sich vollständig auf die Spielentwicklung konzentriert werden. Der entwickelte Spielserver wird automatisch in die AWS GameLift Umgebung integriert, was zu einer Minderung der Systemkomplexität führt. Es muss sich mit dem GameLift SDK auseinandergesetzt werden, um die Vorteile der Dienste wie FlexMatch zur regelbasierten Spielerzuordnung nutzen zu können. Dieses SDK ist in vielen Programmiersprachen verfügbar, die von allen führenden Spiel-Engines unterstützt werden.[126] Nachdem diese Anfangshürde überwunden ist, profitiert das Gesamtsystem von den Vorzügen eines Ansatzes, der ohne den Betrieb eigener Server auskommt, wodurch die Implementierung erheblich vereinfacht wird. Zudem fällt der Wartungsaufwand gering aus, da nur wenige Systemkomponenten und ein serverloser Ansatz zum Einsatz kommen. Etwaige Fehler treten überwiegend im Spielserver auf, können dort lokal behoben und anschließend durch eine aktualisierte Version in das System integriert werden.

7.1.2 Erkenntnisse

A1 bietet eine solide und bekannte Grundlage, erfordert jedoch einen moderaten initialen Konfigurationsaufwand und längere Entwicklungszyklen. Sie sollte sich insbesondere für die Konvertierung bestehender Spiele nach AWS eignen. Der serverlose Ansatz in A2 ermöglicht die schnellste Implementierung und reduziert die Wartungskomplexität der Microservices durch die Nutzung von verwalteten Diensten, verlangt jedoch eine präzise Datenmodellierung und Einarbeitung in serverlose Konzepte. A3 bietet eine hohe Flexibilität in der Entwicklung und Erweiterung des Spiels. Sie erfordert jedoch auch einen hohen organisatorischen und technischen Aufwand, um das Konzept der Containerisierung optimal umsetzen zu können.

[126] Vgl. Amazon Web Services, Inc., 2025a, S. 4, S. 56.

A4 findet einen guten Mittelweg, denn sobald die Implementierungshürde bei der Einarbeitung in das GameLift SDK einmal überwunden ist, stehen alle automatisierten Dienstleistungen zur Verfügung, die ähnlich wie bei A2 die Komplexität stark reduzieren. Allerdings werden viele Vorteile des Dienstes, wie die minimale Latenz, bei asynchronen Multiplayer-Spielen nicht optimal ausgeschöpft.

Die Erkenntnisse dieses Vergleichs lassen sich in eine lineare Rangfolge einordnen: von links absteigend, beginnend mit einfachster Implementierung und niedrigster Komplexität und endend mit schwerster Implementierung und höchster Komplexität.

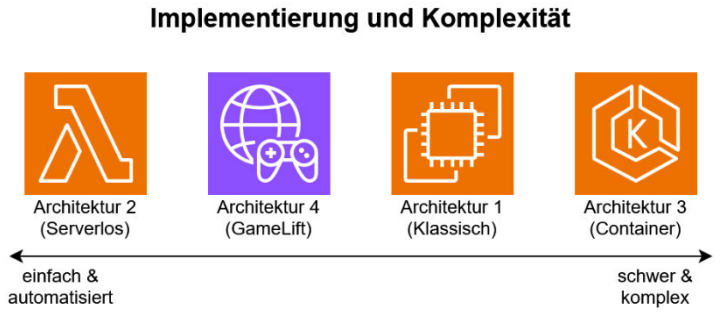


Abbildung 11: Implementierungsaufwand und Komplexitätsgrad der vier Architekturen
Quelle: Eigene Darstellung

AWS bietet Dienste, die Implementierung und Wartung automatisieren. Ein zentraler Ansatz dabei ist die Definition der Infrastruktur als Code. Dieses Paradigma, bekannt als Infrastructure as Code (IaC), vereinfacht die Implementierung verschiedener Dienste erheblich. AWS CloudFormation setzt diesen Ansatz um, indem es alle gemäß gängiger IaC-Standards definierten AWS-Dienste automatisch implementiert. Somit lassen sich in der Praxis alle vier Architekturen als IaC definieren und umsetzen, wenngleich dies ein tiefgehendes Wissen über alle verwendeten Dienste und deren Interaktionen erfordert.[127]

[127] vgl. Shrivastava et al., 2023, S. 330 f.

7.2 Skalierbarkeit

Skalierbarkeit ist eines der wichtigsten Verkaufsargumente für den Wechsel zu Cloud-Anbietern, da sie die Flexibilität bietet, wechselnde Nutzerzahlen und Lastspitzen effizient zu bewältigen und gleichzeitig die Kosten optimiert. Die vier Architekturen bieten ein unterschiedliches Maß an automatisierter Skalierbarkeit und nutzen verschiedene Varianten von Skalierungsdiensten. Der folgende Vergleich soll die individuellen Stärken und Schwächen dieser Varianten aufzeigen.

Skalierung unterteilt sich in zwei unterschiedliche Methoden, die vertikale Skalierung und die horizontale Skalierung. Bei der vertikalen Skalierung wird die Leistung eines einzelnen Servers gesteigert, indem diesem mehr Ressourcen, wie CPU oder Arbeitsspeicher, hinzugefügt werden. Bei der horizontalen Skalierung hingegen wird die Systemkapazität durch das Hinzufügen zusätzlicher Container oder Server erweitert. Dieser Ansatz ist vor allem in verteilten Systemen und modernen Cloud-Architekturen verbreitet, da er eine hohe Flexibilität und Ausfallsicherheit bietet.[128]

7.2.1 Vertikale Skalierung

Da AWS keine dedizierten Dienste bereitstellt, die eine automatisierte vertikale Skalierung ermöglichen, erfolgt die automatisierte Skalierung in allen vier Architekturen primär horizontal. Die vertikale Skalierung muss manuell durchgeführt werden, indem Instanztypen auf ein höheres oder niedrigeres Leistungsniveau geändert werden. Zwar ist eine automatisierte vertikale Skalierung möglich, sie erfordert jedoch die Entwicklung eigener Skripte, die diese Instanztypen ereignisgesteuert modifizieren. Der große Nachteil der vertikalen Skalierung besteht darin, dass der Server neu gestartet werden muss, was zu einer Unterbrechung des Dienstes führt.

[128] vgl. Golon, Piotr: Horizontal scaling and vertical scaling in AWS, in: BlowStack, 26.01.2024, https://blowstack.com/blog/horizontal-scaling-and-vertical-scaling-in-aws (abgerufen am 28.02.2025).

7.2.2 Verwendete Skalierungsmethoden

In A1 erfolgt die Skalierung durch den kombinierten Einsatz von ELB und EC2 Auto-Skalierungsgruppen. Dabei wird der gesamte eingehende Traffic über den Load Balancer auf eine Gruppe von EC2-Instanzen verteilt. Die Auto-Skalierungsgruppe erweitert oder reduziert die Anzahl dieser Instanzen horizontal und in fest definierten Größen.[129] Es werden also mehrere Instanzen gleichzeitig hinzugefügt oder entfernt, anstatt dass einzelne Komponenten isoliert skaliert werden. Dies führt zu einer weniger feingranularen Steuerung, da nur die einzelnen Schichten unabhängig skaliert werden können und immer ganze Blöcke an Servern bereitgestellt werden.

Im Gegensatz dazu nutzt A2 serverlose Dienste wie AWS Lambda, API Gateway und DynamoDB. Hier wird jede eingehende Anfrage einzeln behandelt. AWS Lambda startet bei Bedarf für jede neue Anfrage eine separate Instanz der Funktion, was eine extrem feingranulare und dynamische Skalierung ermöglicht.[130] DynamoDB passt sich dynamisch an die Anzahl der Anfragen an, indem es den Datenverkehr auf mehrere Partitionen verteilt, während API Gateway den Traffic automatisch verwaltet und skaliert. A2 reagiert also sehr präzise auf den aktuellen Bedarf, ohne dass manuelle Eingriffe notwendig sind. Der Nachteil bei dieser Skalierung ist, dass für jeden Aufruf einer Lambda-Funktion die darunterliegende Ausführungsumgebung zunächst einmal initialisiert werden muss, was Zeit kostet und somit die Latenz des Systems erhöht. Da asynchrone Multiplayer-Spiele nicht stark von Latenzen abhängen, fällt dieser Nachteil aber nicht stark ins Gewicht.

[129] vgl. Shrivastava et al., 2023, S. 184; vgl. Golon, 2024, Kap. 3.
[130] vgl. Amazon Web Services, Inc.: AWS Lambda - Developer Guide, 20.02.2025b, [PDF] https://docs.aws.amazon.com/pdfs/lambda/latest/dg/lambda-dg.pdf, S.35, S. 425 ff.

In der containerbasierten A3 wird die automatische Skalierung mithilfe von Amazon EKS realisiert. Dabei übernimmt die horizontale Pod-Autoskalierung (HPA) die dynamische Anpassung einzelner Microservices. Das HPA überwacht kontinuierlich Leistungskennzahlen, wie CPU und Speicherauslastung, und startet bei steigender Last zusätzliche Container Pods, um die Leistungsanforderungen der Anwendung zu erfüllen. Kubernetes bietet zudem eine vertikale Pod-Autoskalierung (VPA) als Add-On, die die zur Verfügung stehenden Ressourcen dynamisch auf die Container-Pods verteilt und sie somit vertikal skaliert. Es werden hierbei aber keine zusätzlichen Ressourcen durch AWS automatisch hinzugefügt. Ergänzend zu HPA und VPA sorgt die Cluster-Autoskalierung von EKS dafür, dass die zugrunde liegende EC2 Infrastruktur automatisch erweitert wird, wenn die vorhandenen Server nicht mehr ausreichen.[131] Somit reagiert das System sowohl auf veränderte Anforderungen auf Ebene einzelner Container als auch auf die Notwendigkeit, die zugrunde liegende Hardware anzupassen. Obwohl diese Skalierung granulärer ist als in Architektur 1, erfolgt sie weiterhin auf Ebene der Container und Pods und kann keine einzelnen Anwendungsfunktionen innerhalb eines Pods isoliert skalieren.

Bei A4 basiert die automatische Skalierung auf der Verwaltung kompletter sogenannter GameLift Server Flotten. Hier wird die Skalierung auf Ebene ganzer Spielinstanzen umgesetzt, je nach Spieleraufkommen werden ganze Instanzen hoch oder heruntergefahren. Jeder gestarteten Session wird eine komplette Instanz zugewiesen, sodass jede Session über alle notwendigen Ressourcen verfügt.[132] Dieser Ansatz ist weniger feingranular, da bei einem plötzlichen Anstieg der Anfragen zunächst alle vorhandenen Slots belegt sind und erst durch das Hochfahren neuer Spielinstanzen zusätzliche Kapazitäten geschaffen werden können.

[131] vgl. Shrivastava et al., 2023, S. 493.
[132] Vgl. Amazon Web Services, Inc., 2025, S. 407 f.

7.2.3 Erkenntnisse

A1 und A4 skalieren immer ganze Servergruppen gleichzeitig, was bei Schwankungen häufig zu einer Überkapazität führt und somit unnötige Kosten verursacht. Im Gegensatz dazu passen sich die Systeme in A2 und A3 viel präziser an den aktuellen Bedarf an. Architektur A2 teilt exakt so viele Ressourcen den Lambda-Funktionen zu, wie gerade benötigt werden, während Architektur A3 sowohl einzelne Dienste als auch die zugrunde liegende Infrastruktur flexibel anpasst, um effizient auf Laständerungen reagieren zu können.

Die Erkenntnisse dieses Vergleichs lassen sich ebenfalls in eine lineare Rangfolge einordnen: von links absteigend, beginnend mit der feinteiligsten automatischen Skalierung und dem größten Angebot an Einstellungsmöglichkeiten und endend mit der gröbsten automatischen Skalierung.

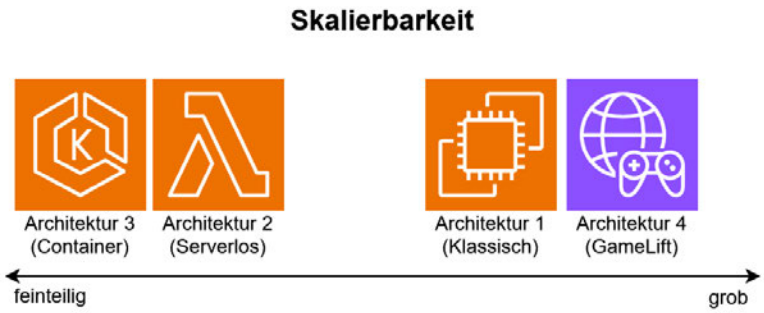


Abbildung 12: Skalierbarkeit der vier Architekturen
Quelle: Eigene Darstellung

7.3 Betriebskosten

Die Kosten, die durch den Betrieb einer Architektur entstehen, sind ein zentrales Merkmal, das bei der Planung und Auswahl berücksichtigt werden muss. Selbst wenn eine Architektur alle funktionalen und qualitativen Anforderungen erfüllt, ist sie aus kommerzieller Sicht nicht tragfähig, sofern die Betriebskosten das zur Verfügung stehende Budget überschreiten. Aufgrund des nutzungsbasierten Preismodells von AWS ist eine exakte Kostenprognose nicht möglich, da ausschließlich die tatsächlich in Anspruch genommenen Ressourcen oder Betriebszeiten in Rechnung gestellt werden. Ähnlich wie bei einem Stromvertrag wird der Verbrauch monatlich abgerechnet.

7.3.1 Kostenabschätzung

AWS hat eine sehr komplexe Kostenstruktur. Um die Kostenabschätzung zu vereinfachen und transparenter zu gestalten, stellt AWS den AWS-Preisrechner[133] zur Verfügung. Mit diesem Tool kann für jeden verwendeten AWS-Dienst eine geschätzte Auslastung angegeben werden, um eine monatliche Kostenabschätzung zu erhalten. Es wird verwendet, um die Betriebskosten der vier Architekturen zu berechnen. Dabei ist zu beachten, dass zwar teilweise dieselben AWS-Dienste in den Architekturen genutzt werden, diese jedoch unterschiedlich stark beansprucht werden. Rabatte oder kostenlose Testzeiträume fließen in die Kostenkalkulation nicht ein.

Um eine Vergleichbarkeit der Kosten der unterschiedlichen Architekturvarianten herzustellen, wird die Spieleranzahl pro Tag als zentrale Bezugsgröße für die Kostenabschätzung verwendet. Auf Basis von vier Kostenabschätzungen für 100, 1.000, 5.000 und 10.000 tägliche Spieler wird eine Kostenfunktion berechnet, die den Zusammenhang zwischen der täglichen Spieleranzahl und den monatlichen Betriebskosten jeder Architektur beschreibt.

[133] AWS Pricing Calculator: o. D., [online] https://calculator.aws/ (abgerufen am 03.03.2025).

Zudem beschränkt sich der Vergleich auf die Bereitstellung der Dienste in zwei AZs der AWS-Region Europa/Frankfurt (eu-central-1). Diese Einschränkung dient dazu, infrastrukturelle Rahmenbedingungen zu vereinheitlichen und zu vereinfachen, wodurch potenzielle Fehlerquellen in der Kostenabschätzung minimiert werden.

Obwohl versucht wurde, möglichst realitätsnahe Werte auszuwählen, hängen diese stark von den spezifischen Eigenschaften des zu implementierenden Spiels ab. Im Anhang A.1 und A.2 wird eine detaillierte Aufschlüsselung der Berechnungen für alle verwendeten Werte dargestellt.

Neben der täglichen Spieleranzahl werden folgende Konstanten für die Berechnung als gegeben betrachtet:

Tabelle 2: Grundlegende Annahmen für die Kostenabschätzungen
Quelle: Eigene Darstellung

Beschreibung	Wert	Bezeichner
Anzahl gesendeter Anfragen pro Spieler pro Tag	500	N_{Req}
Datengröße einer Anfrage	5KB	S_{Req}
Bearbeitungsdauer einer Anfrage	50ms	t_{Req}
Anzahl an Microservices in verteilten Systemen	20	N_{MS}

7.3.2 Ergebnisse der Kostenabschätzungen

Tabelle 3: Ergebnisse der Kostenabschätzung der vier Architekturen pro täglicher Spieleranzahl
Quelle: Eigene Darstellung

Spieler pro Tag	A1 (Klassisch)	A2 (Serverlos)	A3 (Container)	A4 (GameLift)
100 Spieler	346,85 €	63,33 €	400,61 €	154,61 €
1.000 Spieler	496,76 €	503,04 €	579,73 €	698,77 €
5.000 Spieler	1.056,34 €	2.077,61 €	1.348,13 €	2.970,38 €
10.000 Spieler	1.952,96 €	4.276,31 €	2.248,33€	5.875,97 €

Tabelle 3 listet die Ergebnisse der vier Kostenabschätzungen für jede Architektur auf. Diese zeigen einen linearen Zusammenhang zwischen der täglichen Spieleranzahl und den monatlichen Betriebskosten. Da die Ressourcen und die Betriebszeit auf Basis der täglichen Spieleranzahl linear berechnet werden, kann daraus vermutet werden, dass AWS diese Parameter ebenfalls linear in seine Kostenberechnung einbezieht. Aus den ermittelten Werten lässt sich daher eine lineare Kostenfunktion ableiten, die eine Abschätzung der monatlichen Betriebskosten pro täglicher Spieler ermöglicht. Diese Erkenntnis bietet eine solide Grundlage für den Vergleich der Betriebskosten, da sie darlegt, dass eine Zunahme der Spieleranzahl zu einer proportionalen Erhöhung der Betriebskosten führt. Die einzelnen Kostenfunktionen sind im Anhang unter A.2.1 - A.2.4 einsehbar. Folgende Funktionen konnten abgeleitet werden:

Tabelle 4: Kostenfunktion der vier Architektur mit Gütemaß und Mittelwert
Quelle: Eigene Darstellung

Architektur	Kostenfunktion	R^2	Mittelwert
A1 (Klassisch)	$k_{A1}(p) = 0{,}1607p + 316{,}49$	0,9965	~ 1.128,03 €
A2 (Serverlos)	$k_{A2}(p) = 0{,}4216p + 33{,}05$	0,9993	~ 2.161,72 €
A3 (Container)	$k_{A3}(p) = 0{,}1866p + 393{,}08$	0,9997	~ 1.335,41 €
A4 (GameLift)	$k_{A4}(p) = 0{,}5765p + 104{,}63$	0,9999	~ 3.015,96 €

Der Mittelwert im Intervall von [100; 10.000] jeder Kostenfunktion wurde mit folgender Formel berechnet:

$$k_M = \frac{1}{9900} \int_{100}^{10000} k(p)\, dp$$

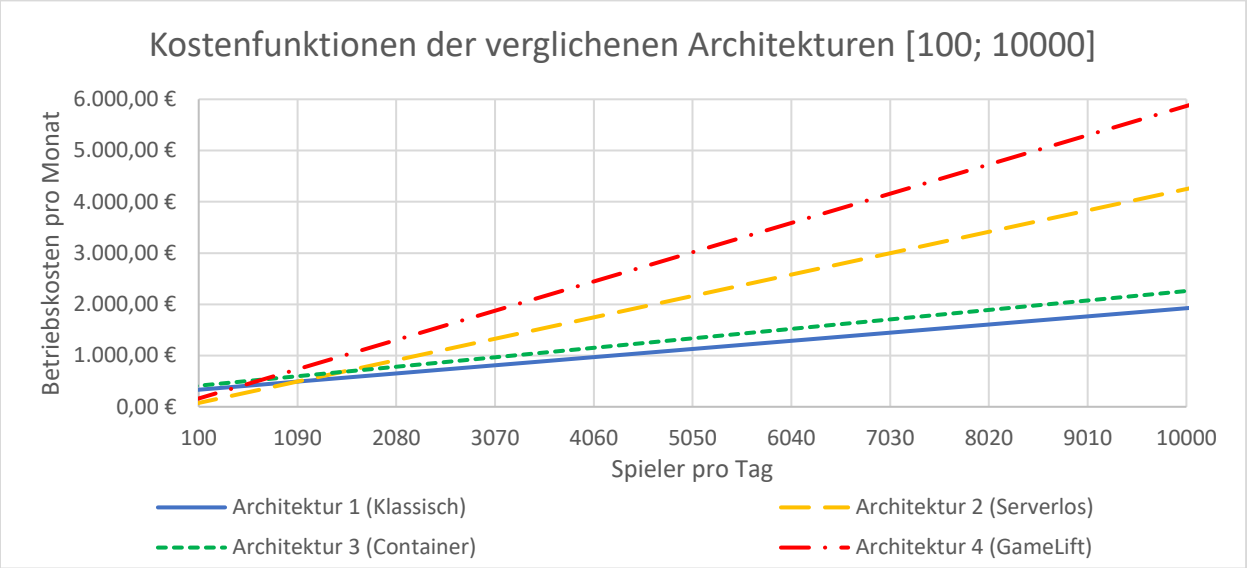


Abbildung 13: Kostenfunktionen im grafischen Vergleich im Intervall von 100 bis 10.000 täglichen Spielern
Quelle: Eigene Darstellung

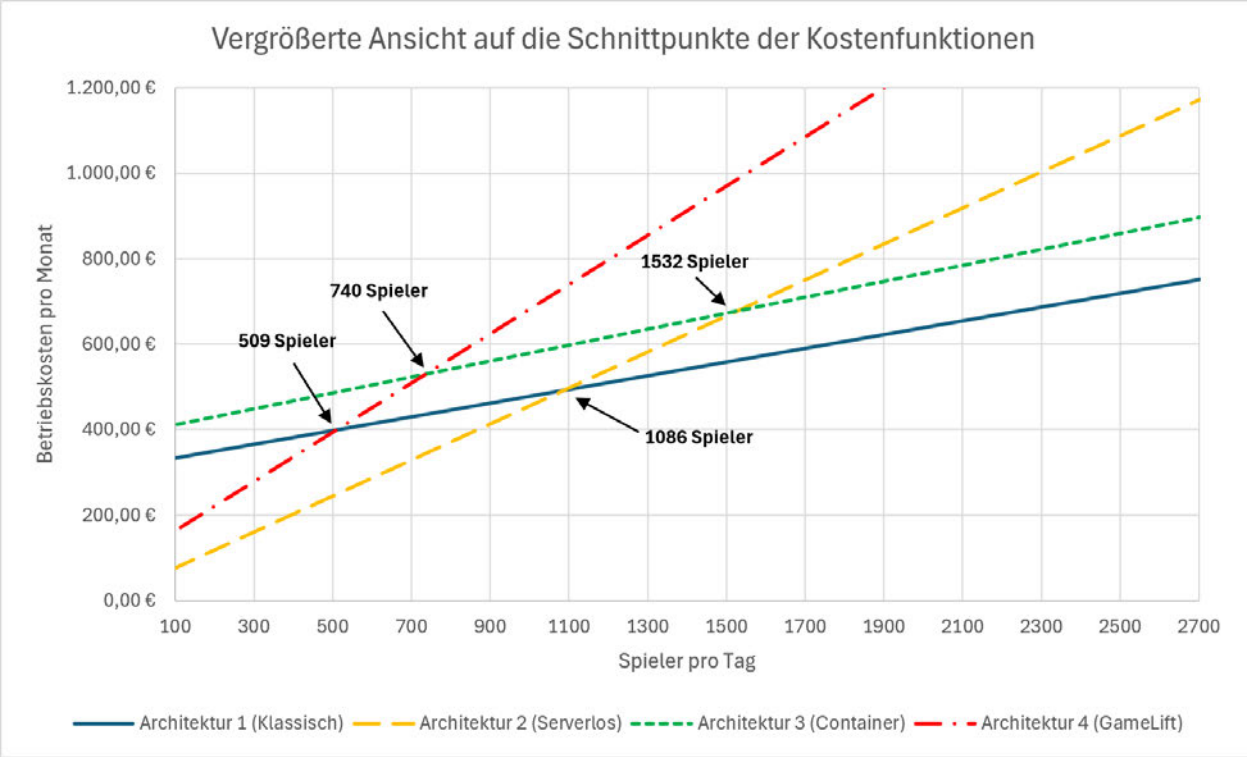


Abbildung 14: Schnittpunkte der Kostenfunktionen
Quelle: Eigene Darstellung

7.3.3 Auswertung

Die Vergleiche in den Abbildungen 13 und 14 zeigen, dass keine der untersuchten Architekturen durchgängig als kostengünstigste oder teuerste einzustufen ist. Dennoch ist A1 über das gesamte Intervall von 100 bis 10.000 Spielern im Mittel mit rund 1.128 Euro die günstigste Lösung, während A4 mit rund 3.016 Euro als teuerste Variante gilt.

Wird eine geringere tägliche Spieleranzahl von unter 1.100 Spielern erwartet, erweist sich A2 als die beste Wahl. Durch den serverlosen Ansatz werden lediglich die tatsächlich in Anspruch genommenen Leistungen in Rechnung gestellt. Im Gegensatz zu den anderen Architekturen laufen hier keine Serverinstanzen im Leerlauf, wenn keine Anfragen von Spielern gestellt werden. Befindet sich A2 allerdings selten im Leerlauf, was bei höheren Spielerzahlen immer wahrscheinlicher ist, steigen die Kosten stark an. In diesem Vergleich wird A2 ab etwa 1.100 täglichen Spielern teurer als A1 und nach etwa 1.500 täglichen Spielern teurer als A3.

A3 erweist sich als weniger attraktiv, auch wenn sie mit einer durchschnittlichen Kostenschätzung von etwa 1.335 Euro den zweiten Platz belegt. Wie auch in A1 sind aufgrund des Betriebs mehrerer EC2-Instanzen die anfänglichen Kosten zwar am höchsten, steigen jedoch danach nur geringfügig an. Ab etwa 2.200 täglichen Spielern wird A3 günstiger als A2, bleibt jedoch im gesamten Intervall teurer als A1, weshalb sie in keinen der Fälle als günstigste Lösung hervorgeht.

Der suboptimale Einsatz von GameLift-Instanzen in A4 führt zu einem steilen Kostenanstieg und resultiert in durchschnittlichen Kosten von etwa 3.016 Euro. Sie ist in diesem Szenario mit Abstand die teuerste Architektur.

7.3.4 Erkenntnisse

Die Kosten hängen maßgeblich von der zu erwartenden täglichen Spieleranzahl ab. Für einen Betrieb über ein breites Spektrum an Spielerzahlen ist A1 die kosteneffizienteste Lösung. Für Szenarien, wo niedrigere Spielerzahlen erwartet werden, ist A2 aufgrund des serverlosen Modells die beste Wahl, wobei hier jedoch die steigenden Kosten bei zunehmender Auslastung berücksichtigt werden müssen. A3 ist zwar in diesem Szenario nicht die günstigste Variante, kommt aber mit einem mittleren Unterschied von rund 200 Euro sehr nah an die Kosten von A1 heran. Dazu kommen die Vorteile der Containerisierung, die A3 daher insgesamt attraktiver erscheinen lassen als A1. Mit Abstand am teuersten ist A4, was in diesem Szenario zum größten Teil durch Amazon GameLift verursacht wird. Dies könnte die Abwesenheit des Dienstes in den Fallbeispielen erklären, da er für die geringeren Anforderungen in asynchronen Mehrspieler-Spielen zu teuer ist. Aus diesen Erkenntnissen ergibt sich folgende lineare Rangfolge:

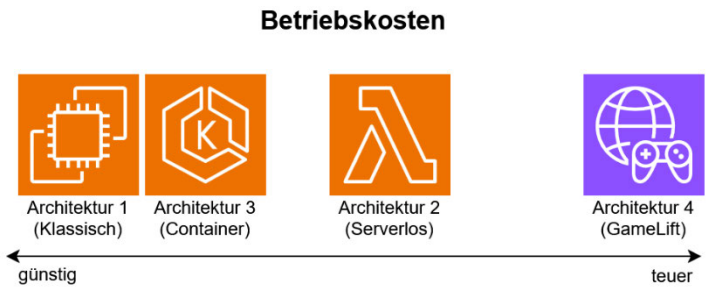


Abbildung 15: Betriebskosten der vier Architekturen
Quelle: Eigene Darstellung

In allen betrachteten Architekturen nimmt die Datenbank einen großen Anteil an den gesamten Betriebskosten ein. Eine gezielte Optimierung in diesem Bereich könnte daher zu erheblichen Kosteneinsparungen führen. Zudem lässt sich durch eine präzisere Abschätzung des tatsächlichen Ressourcenbedarfs und eine genauere Anpassung der Datenbankkonfigurationen die Leistung verbessern.

7.4 Zukunftssicherheit und Änderbarkeit

In der heutigen sich stetig verändernden IT-Landschaft ist es entscheidend, dass eine Architektur nicht nur den aktuellen Anforderungen gerecht wird, sondern auch flexibel genug ist, um zukünftige technologische Entwicklungen und Änderungen zu integrieren. In diesem qualitativen Vergleich wird versucht, für jede Architektur eine begründete Einschätzung zu erarbeiten, wie schnell diese sich an zukünftige Anforderungen anpassen könnte.

7.4.1 Begründete Einschätzungen

Das zugrundeliegende dreischichtige Modell aus A1 gilt als erprobt und stabil, da es lange als der Standard für große Systeme galt. Allerdings zeigt sich seit einigen Jahren ein klarer Trend weg von traditionellen Drei-Schichten-Modellen hin zu serverlosen Microservices-Architekturen.[134] Neben der besseren Skalierbarkeit wird dies oft durch die Aussicht auf schnellere Release-Zyklen und einer agileren Entwicklung begründet. Da die Logik eines Schichten-Systems in wenigen Anwendungen zusammengefasst ist, betreffen Änderungen an den Kernfunktionen oft einen großen Teil der Codebasis, was zu längeren Entwicklungszyklen führt.[135] Eine Migration dieser Architektur in eine Microservices-Architektur ist sehr komplex, weil die gesamte Anwendung entkoppelt und in mehrere eigenständigen Services aufgeteilt werden muss. Das trotz dieser komplexen und zeitintensiven Umwandlung viele Unternehmen auf eine Microservices-Architektur wechseln[136], zeigt wie wertvoll die Vorteile dieses Ansatzes sind und gleichzeitig wie unflexibel das dreischichtige Modell aus A1 bezüglich ihrer Änderbarkeit und Zukunftssicherheit ist.

[134] Vgl. Kratzke, Nane: A Brief History of Cloud Application Architectures, 14.08.2018, [PDF] doi:10.3390/app8081368, S. 19 f.; vgl. Velepucha/Flores, 2023, Abschn. 2.B.2.
[135] Vgl. Velepucha/Flores, 2023, Abschn. 4.B.2.
[136] Vgl. Stojanov, Z./I. Hristoski/J. Stojanov/A. Stojkov: A Tertiary Study on Microservices: Research Trends and Recommendations, in: Springer, Bd. 49, Nr. 8, 24.01.2024, [PDF]doi:10.1134/s0361768823080200, Kap. 4.3.2.

A2 und A3 basieren beide auf Microservices, unterscheiden sich jedoch erheblich in ihrer Abhängigkeit von AWS. Während beide Ansätze durch ihren modularen Aufbau und das eventbasierte Kommunikationsmuster eine hohe Änderbarkeit bieten, setzt A2 stärker auf AWS eigene Dienste. Dies führt zu einem „Vendor-Lock-in“-Effekt, was bedeutet, dass diese Architektur aufgrund der engen Bindung an AWS kaum portierbar ist und ein Wechsel zu anderen Anbietern mit einem erheblichen Anpassungsaufwand verbunden wäre. Im Gegensatz dazu nutzt A3 standardisierte Open-Source-Tools, wie Docker und Kubernetes, wodurch ihre Portabilität deutlich verbessert wird. Diese Lösung kann problemlos in andere Cloud-Umgebungen oder auf eigene Server übertragen werden, was sie letztlich zukunftssicherer macht. Allerdings erfordert A3 auch einen höheren Managementaufwand, besonders wenn sich Anforderungen ändern oder das System wächst, während A2 durch vereinfachte CI/CD-Pipelines schnellere Anpassungen ermöglicht.

Wie auch in anderen Vergleichen entstehen durch die eher unkonventionelle Verwendung von AWS GameLift mehr Nachteile als Vorteile für A4. Sie orientiert sich weniger an verteilten Systemen und verlässt sich stattdessen auf traditionelle monolithische Game-Server-Prinzipien. Zwar integriert sie weitere Dienste wie Lambda, API Gateway und SNS für die Spielverwaltung, doch im Kern laufen die eigentlichen Spielprozesse in eigenen Serverprozessen, was den Umstieg auf zukünftige Technologien, aus ähnlichen Gründen wie bei A1, erschwert. Auch wenn die Lobby- und Matchmaking-Services als serverlose Microservices realisiert sind, ist die Kernlogik der Matches fest in den GameLift-Servern verankert. Dadurch ist man an die spezifischen GameLift-APIs gebunden, was die Flexibilität einschränkt. Auch die erhofften Vorteile bei einem Wechsel von asynchronen auf echtzeitbasierenden Mechaniken bleiben aus. Denn bei solchen grundlegenden Änderungen im Spieldesign müssten erhebliche Teile der GameLift-Integration neu konzipiert werden, was die Anpassungsfähigkeit dieser Architektur umso mehr begrenzt.

7.4.2 Erkenntnisse

Der Vergleich zeigt, dass die Zukunftssicherheit und Änderbarkeit von Architekturen stark von dem zugrunde liegenden Modell abhängen als von einzelnen Services. Traditionelle Schichten-Modelle bieten Stabilität, jedoch geringere Flexibilität, während Microservices-Architekturen durch ihre modulare Struktur anpassungsfähiger sind. A3 findet hier die beste Lösung und bietet durch seine Microservices und die Nutzung standardisierter Tools eine hohe Anpassungsfähigkeit und gleichzeitig eine schwache Bindung an den Anbieter AWS. Anhand von A4 lässt sich gut zeigen, dass eine Mischung aus serverlosen und monolithischen Diensten nicht unbedingt zu besseren Lösungen führen kann.

Die Erkenntnisse dieses Vergleichs lassen sich ebenfalls in eine lineare Rangfolge einordnen: von links absteigend, beginnend mit der zukunftssichersten Architektur und endend mit der unflexibelsten Architektur.

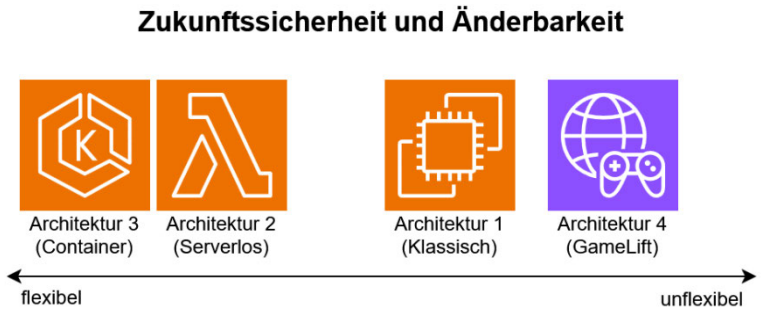


Abbildung 16: Zukunftssicherheit und Änderbarkeit der vier Architekturen
Quelle: Eigene Darstellung

7.5 Fazit

Architektur A1 bietet eine stabile und einheitliche Grundlage für asynchrone Multiplayer-Spiele. Allerdings sind Anpassungen an neue Anforderungen oder Modernisierungen sehr umständlich umsetzbar. A1 skaliert immer ganze Servergruppen und reagiert unflexibel auf Lastschwankungen, was zu Überkapazitäten und unnötigen Kosten führt. Betrachtet man allein die Abhängigkeit von der Spielerzahl, erscheint A1 zunächst als kosteneffizienteste Lösung. Unter konstanter Last können jedoch Engpässe entstehen, die mit den unnötigen Kosten der unflexiblen Skalierung schnell zu einer Kostenfalle führen und einen erheblichen manuellen Wartungs- und Optimierungsaufwand erfordern. Dies erklärt, warum dieser Ansatz in der Spieleindustrie nicht als Grundlage für neue Projekte genutzt wird. Dass Amazon diese Referenzarchitektur trotzdem für asynchrone Multiplayer empfiehlt, deutet darauf hin, dass sie vor allem als Einstieg in AWS und für kleine online/mobile Spiele konzipiert wurde.

A2 ermöglicht dank der ausschließlichen Nutzung verwalteter Dienste eine schnelle Implementierung, bei der Ressourcen exakt dem tatsächlichen Bedarf zugewiesen werden. Für Indie-Entwickler, die nicht mit einem massiven Spieleraufkommen rechnen, ist diese Feingranularität besonders kosteneffizient. Zudem erlaubt der Einsatz verwalteter Dienste eine einfache, automatisierbare Wartung, wodurch der administrative Aufwand nahezu entfällt. Indie-Entwickler können sich somit voll und ganz auf die Spielentwicklung konzentrieren, ohne sich mit komplexen Implementierungsprozessen oder der Optimierung auseinandersetzen zu müssen. Allerdings steigen die Kosten bei zunehmender Auslastung steil an, was diese Architektur für größere asynchrone Online-Spiele ineffizient macht. Gleichzeitig führt die Nutzung spezieller AWS-Dienste zu einer langfristigen Bindung an die AWS-Infrastruktur, was bei der Auswahl beachtet werden muss.

A3 eignet sich für große asynchrone Multiplayer-Spiele, da sie eine flexible und präzise Skalierung ermöglicht. Allerdings erfordert die Implementierung und Wartung dieser Architektur ein großes, spezialisiertes Team, da die Koordination der verschiedenen Dienste einen großen organisatorischen und technischen Aufwand mit sich bringt. Diese Herausforderungen von A3 sind vor allem für große Spieleunternehmen umsetzbar, die über die notwendigen personellen und finanziellen Ressourcen verfügen. Aus den Praxisbeispielen und dem Vergleich wird klar, warum diese Architektur als Industrie-Standard gilt. Denn sie wird den Anforderungen an hohe Lasten und flexible Anpassungen in groß angelegten Spielumgebungen optimal gerecht. Zudem ist ihr Betrieb langfristig gesehen wahrscheinlich günstiger als der von A1, da durch einer besseren Skalierung, einer effizienteren Ressourcennutzung und einer optimierten Lastenverteilung unnötige Kosten vermieden werden.

Das Negativbeispiel A4 ist für asynchrone Multiplayer-Spiele nicht geeignet. Im Vergleich wurde deutlich, dass der spezialisierte Dienst GameLift für Echtzeit-Spiele konzipiert wurde, in denen viele Spieler innerhalb eines sessionbasierten Systems gleichzeitig und über blitzschnelle Latenz interagieren. Zwar ist der Einsatz von GameLift in asynchronen Multiplayer-Szenarien technisch möglich, bringt jedoch erhebliche Einschränkungen mit sich. Die enge Integration der Kernlogik in die GameLift-Server erschwert grundlegende Änderungen und Anpassungen im Spieldesign, was zu höheren Umbaukosten führt. Zudem erweist sich A4 aufgrund der schlichtweg falschen Nutzung von Sessions als zu teuer und nicht effizient skalierbar. Dadurch stellt sich GameLift langfristig als weniger flexibel dar und ist für die Nutzung asynchroner Multiplayer-Spiele ungeeignet.

Zusammenfassend lässt sich feststellen, dass A2 und A3 für asynchrone Multiplayer-Spiele die beste Lösungs-Architektur bieten. Aufgrund der zeitlichen Entkopplung der Spielzüge und der Speicherung von Zuständen über unbestimmte längere Zeiträume in Datenbanken sind zustandslose und verteilte Systeme, die während ihrer Ausführung auf diese Daten zugreifen, im Vorteil. Folglich lassen sich asynchrone Multiplayer-Spiele mithilfe von RESTful- oder HTTP-APIs implementieren, da eine Echtzeitverarbeitung von Zuständen nicht erforderlich ist. Zudem sind diese Spiele hohen Lastschwankungen ausgesetzt, weshalb die feinteilige Skalierbarkeit der beiden Architekturen ebenfalls von Vorteil ist, um Betriebskosten effizient zu optimieren.

Im Kontext dieses Vergleichs zeigt sich, dass zustandslose, ereignisbasierte Architekturen in Form von Microservices als beste Lösungs-Architektur hervorgehen. Ob diese Architekturen nun serverlos oder containerbasiert umgesetzt werden, hängt von der Größe des Teams sowie vom vorhandenen Budget ab. Eine serverlose Architektur, wie in A2, ist für Indie-Entwickler oder bei kleineren Spielerzahlen aufgrund ihrer einfachen Wartung und der günstigen Einstiegspreise sinnvoll, während containerbasierte Architekturen, wie in A3, durch ihre Flexibilität und Kosteneffizienz bei hohen Spielerzahlen punkten.

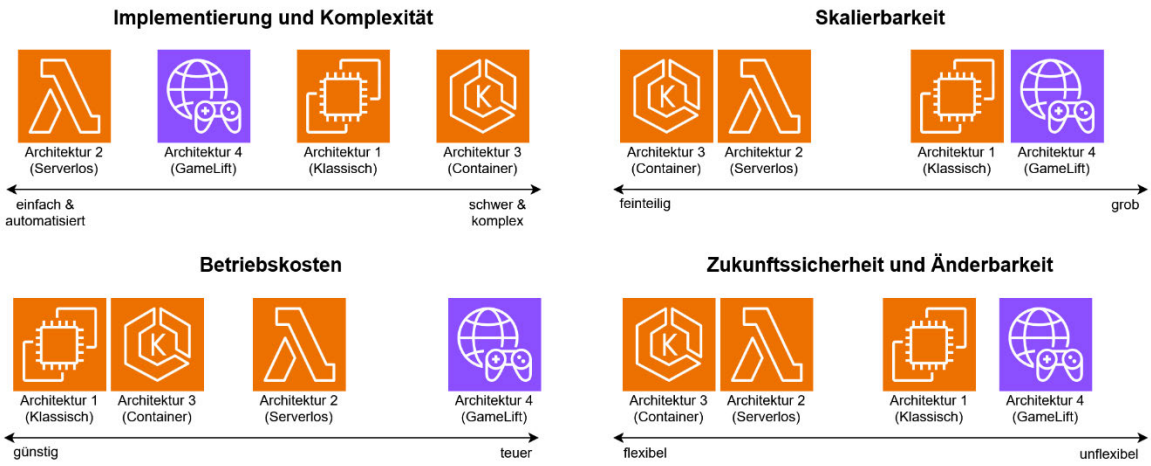


Abbildung 17: Alle Erkenntnisse aus den Vergleichen der vier Architekturen im Überblick
Quelle: Eigene Darstellung

8 Leitfaden

Mit den Erkenntnissen des Vergleichs und den Vorgehensweisen aus anderen Quellen lässt sich ein verfeinerter Leitfaden für die Entwicklung asynchroner Multiplayer-Architekturen erstellen. Hierbei ist zu erwähnen, dass der im Folgenden vorgestellte Leitfaden auf den Erkenntnissen des Vergleichs, eigenen, kritisch reflektierten Einschätzungen sowie auf Teilen der Verantwortlichkeiten eines Lösungs-Architekten nach Shrivastava und Srivastav [137] aufbaut. Er soll als Orientierungshilfe für zukünftige Projekte dienen. Es konnten vier differenzierte Schritte herausgearbeitet werden, die für den Entwurf einer passenden Lösungs-Architektur im Kontext asynchroner Multiplayer-Spiele wichtig sind.

8.1 Schritt 1: Anforderungsanalyse

Für jede Architektur, sei es eine Cloud-Architektur oder ein Software-Architektur, muss zunächst eine tiefgreifende Anforderungsanalyse durchgeführt werden, die ein solides Fundament für die darauffolgenden Entscheidungen bildet. [138] Dabei müssen sowohl funktionale als auch nicht funktionale Anforderungen genau definiert und herausgearbeitet werden. Hier wird noch keine Auswahl an Diensten getroffen, sondern es erfolgt eine genaue Auseinandersetzung mit den geplanten Mechaniken des zu implementierenden asynchronen Multiplayer-Spiels.

[137] Vgl. Shrivastava/Srivastav, 2024, S. 16 ff.
[138] Vgl. ebd., S. 17.

8.1.1 Funktionale Anforderungen

Funktionale Anforderungen stellen die Kernmechaniken und die Geschäftsprozesse des Spiels dar.[139] Hier muss überlegt werden, wie die Logik zwischen Spielclient und Cloud aufgeteilt werden soll. Welche Funktionen in die Cloud verlagert werden sollten, hängt zum Beispiel davon ab, ob es sich um eine Online-Funktionalität handelt oder ob die betreffende Mechanik ein besonders sicheres Umfeld benötigt, um Manipulationen vorzubeugen. Am Ende dieser Analyse sind alle funktionalen Anforderungen an die Cloud strukturiert aufgelistet und es ist klar formuliert, was sie im Einzelnen tun sollen.

8.1.2 Nicht-funktionale Anforderungen

Alle im Vergleich verwendeten Kategorien sind nicht-funktionale Anforderungen an die Architektur. Hier muss gut überlegt und abgeschätzt werden, für welchen Implementierungs- und Wartungsaufwand das Team ausgelegt ist, ob in naher Zukunft die Architektur erweitert oder angepasst werden muss, welches Budget für den Betrieb zur Verfügung steht und ob bereits Wissen über bestimmte Architektur-Muster oder Tools vorhanden ist. Hierfür eignet sich zum Beispiel die Durchführung einer Analyse der Stärken, Schwächen, Chancen und Risiken, auch SWOT-Analyse genannt, des Projekts und des Entwicklerteams in Bezug auf nicht-funktionale Anforderungen der Architektur.

Abbildung 18: Wichtige nicht-funktionale Anforderungen für Lösungs-Architekturen nach Shrivastava und Srivastav Quelle: (Übersetzt) Shrivastava/Srivastav, 2024, S. 19

[139] Vgl. Shrivastava/Srivastav, 2024, S. 18 ff.

8.1.3 CCU und Lastenspitzen

Der Vergleich zeigt, dass die gleichzeitige Spieleranzahl (engl. Concurrent User, kurz CCU) eine der wichtigsten Kenngrößen für asynchrone Multiplayer-Spiele ist, da sie den größten Einfluss auf die Wahl der Architektur hat. Sie ist auch eine nicht-funktionale Anforderung und sollte präzise geschätzt werden. CCUs anderer Spiele aus demselben Genre oder mit ähnlichen Mechaniken bieten einen guten Ausgangspunkt für diese Kenngröße. Abbildungen 19 und 20 zeigen, dass Videospiele für gewöhnlich stetigen Nutzerschwankungen unterliegen, die größtenteils durch den Tag-Nacht-Zyklus entstehen. Für asynchrone Multiplayer-Spiele ist also eine effiziente Skalierbarkeit notwendig, um diese Schwankungen aufzufangen und Kosten zu sparen.

Abbildung 19: Schwankungen der CCU des asynchronen Multiplayer-Spiels „Backpack Battles“ (Wochenansicht)
Quelle: SteamDB: Steam Charts, [online] https://steamdb.info/app/2427700/charts/ (abgerufen am 02.04.2025)

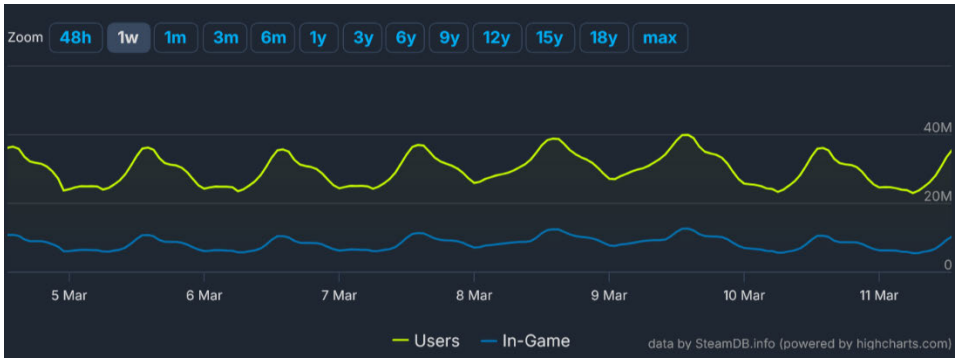


Abbildung 20: Schwankungen der CCU auf der Spiele-Vertriebsplattform Steam (Wochenansicht)
Quelle: SteamDB: Steam Charts, [online] https://steamdb.info/app/753/charts/ (abgerufen am 11.03.2025)

8.2 Schritt 2: Der richtige Entwurf

Basierend auf den erstellten Anforderungen kann nun ein passendes Entwurfsmuster gewählt werden. Dabei wird entschieden, welche Komponenten welche funktionalen Anforderungen umsetzen und wie diese untereinander kommunizieren sollen. Folgende Entscheidungen spielen dabei eine wichtige Rolle.

8.2.1 Drei-Schichten oder Microservices

Die Erkenntnisse aus dem Vergleich zeigen, dass eine klassische Drei-Schichten-Architektur nicht für asynchrone Multiplayer-Spiele geeignet ist. Sie bietet nicht die nötige Flexibilität in ihrer Änderbarkeit und Skalierbarkeit, um den Anforderungen gerecht zu werden. Sie sollte nur in Fällen gewählt werden, in denen ein bereits existierendes Spiel in AWS überführt werden muss, in denen ein Prototyp für einen schnellen Machbarkeitsnachweis entwickelt werden muss oder in denen das Spiel wenige funktionale Anforderungen an den Server hat.

Für neue Projekte ist ein Microservice-Entwurfsmuster, das auf eventbasierte Kommunikation setzt, sinnvoller. Das Entwerfen einer guten Microservice-Struktur ist nicht einfach. Jeder Service muss so designt werden, dass eine lose Kopplung und eine hohe Kohäsion gewährleistet wird. Das Software-Entwurfsmuster „Domain-Driven Design (DDD)“ kann hier als passende Unterstützung im Designprozess angewandt werden. DDD teilt das Gesamtsystem in abgegrenzte Kontexte auf, in denen jeweils ein klar umrissenes Domänenmodell existiert. Diese Kontexte können direkt auf einzelne Microservices abgebildet werden, was zu einer besseren Isolation und Unabhängigkeit der einzelnen Services führt.[140]

[140] Vgl. Shrivastava et al., 2023, S.524, S.532 ff.

Zudem fördert DDD die Trennung von Verantwortlichkeiten und die Bildung kohäsiver Module. Dies entspricht exakt dem Microservice-Prinzip, da jeder Dienst eine klar definierte Aufgabe hat und unabhängig von anderen entwickelt und skaliert werden kann.

8.2.2 Serverlose oder containerbasierte Microservices

Wenn sich für ein Microservices-Entwurfsmuster entschieden wurde, gibt es verschiedene Lösungsansätze, wie dieses in AWS umgesetzt werden kann. Aus der Analyse von existierenden Lösungen aus der Spieleindustrie und dem Vergleich haben sich zwei solcher Ansätze als vielversprechend erwiesen. Die serverlose Architektur, die primär AWS-Lambda nutzt, ist für kleine Studios oder Indie-Entwickler optimal. Ab einer hohen durchschnittlichen täglichen CCU von circa 1500 Spielern ist ein containerbasierter Ansatz die bessere Wahl. Dieser erfordert aber auch mehr Erfahrung und tiefgreifenderes Wissen über das Prinzip der Containerisierung mittels Docker und Kubernetes.

AWS bietet aber auch die Möglichkeit, die Vorteile aus beiden Ansätzen zu vereinen, also eine Mischung aus serverlosem Betrieb und containerisierten Services. Mithilfe des Dienstes AWS Fargate, eine nutzungsbasierte serverlose Datenverarbeitungs-Engine, können Container vollständig automatisiert verwaltet werden. Es entfällt also ein Großteil des Implementierungs- und Wartungsaufwands trotz der Nutzung eines containerisierten Ansatzes. Näheres hierzu in Kapitel 8.3.4.

8.2.3 Kommunikation

Die Kommunikationen unter allen Microservices-Ansätzen erfolgt ereignisbasiert. Fast alle der analysierten und verglichenen Microservices-Architekturen nutzen AWS-Dienste, wie Amazon Eventbridge, Amazon SNS oder Amazon SQS, für eine auf Ereignissen basierende Kommunikation.

Um einen optimaler Austausch von Informationen über ein verteiltes System zu gewährleisten, kann wieder ein Entwurfsmuster aus der Softwareentwicklung unterstützend genutzt werden. Die ereignisgetriebene Architektur (engl. Event-driven architcture, kurz EDA) wird häufig für die Implementierung von Microservices eingesetzt und ist optimal für die Verarbeitung von asynchronen Ereignissen.[141] Sie lässt sich zudem sehr gut mit dem DDD kombinieren. Während DDD den Fokus auf die Modellierung der Geschäftsdomäne und die Definition von klar abgegrenzten Kontexten legt, sorgt EDA dafür, dass diese Domänen über asynchrone, ereignisgesteuerte Nachrichten lose gekoppelt kommunizieren.

Dabei lassen sich die Modelle aus der EDA mithilfe der bereits genannten AWS-Dienste perfekt umsetzen. In einer EDA gibt es Services, die Ereignisse produzieren und/oder konsumieren. Das wären zum Beispiel Lambda-Funktionen oder Anwendungen innerhalb eines Containers. Diese Ereignisse können entweder mithilfe einer Warteschlange (Amazon SQS) synchron abgearbeitet, mit einem Publish/Subscribe-Modell (Amazon Eventbridge) verteilt oder in Kategorien (Amazon SNS) sortiert werden.[142]

[141] Vgl. Shrivastava et al., 2023, S.513.
[142] Vgl. ebd., S.514-520.

8.2.4 Spezialisierte Dienste

Spezialisierte Dienste, die genau auf einen bestimmten Anwendungsfall zugeschnitten sind, bieten eine einfache und unkomplizierte Lösung für genau diesen Anwendungsfall. Ein Beispiel ist Amazon GameLift, das speziell für den Betrieb großer Multiplayer-Spiele konzipiert wurde. Im Vergleich zeigt sich jedoch, dass dieser für asynchrone Multiplayer-Spiele ungeeignet ist. Dennoch lohnt es sich, solche spezialisierten Dienste genauer zu analysieren und zu prüfen, ob sie eine einfache und unkomplizierte Lösung für ein neues Projekt bieten.

8.3 Schritt 3: Der richtige AWS-Dienst

Bei der Identifizierung der richtigen Dienste kommt es oft vor, dass man sich zwischen ähnlichen Alternativen entscheiden muss. Im Folgenden werden in ihrer Funktion ähnliche Dienste gegenübergestellt und kurz auf ihre Vor- und Nachteile eingegangen.

8.3.1 Amazon Route 53, Amazon CloudFront oder AWS Global Accelerator

Amazon Route 53, Amazon CloudFront und AWS Global Accelerator bieten unterschiedliche Lösungen für die Anforderungen beim Routing von Inhalten in globalen Architekturen. Während Route 53 als DNS-basierter Service gezielt Anfragen an die besten Endpunkte weiterleitet, arbeitet Global Accelerator mit Amazons globalen Anycast-IP-Adressen. Das bedeutet, dass Anfragen über das schnellere interne AWS-Netzwerk direkt an regionale Ziele, wie Load Balancer oder EC2-Instanzen, geleitet werden. Da die niedrige Latenz bei asynchronen Multiplayer-Spielen nicht im Vordergrund steht, ist Amazon Route 53 hier die bessere Wahl. Müssen Inhalte, wie große Spieldateien, jedoch global bereitgestellt werden, bietet Amazon CloudFront eine schnelle Bereitstellung dieser in entgegengesetzter Richtung vom Server zum Client.[143]

[143] vgl. Chan, Tak Yu: Route 53 vs Cloudfront vs Global Accelerator, in: Medium, 06.01.2022, https://franky-46708.medium.com/route-53-vs-cloudfront-vs-global-accelerator-96277a65b61f (abgerufen am 26.03.2025).

8.3.2 NAT-Gateway oder NAT-Instanz

In privaten Subnetzen benötigen Instanzen einen Zugang zum Internet. Dafür kann man entweder ein NAT-Gateway oder eine NAT-Instanz einsetzen. Das NAT-Gateway ist ein vollständig verwalteter AWS-Dienst, der automatisch skaliert, in jeder AZ redundant bereitgestellt wird und damit eine hohe Verfügbarkeit bietet.

Im Gegensatz dazu handelt es sich bei einer NAT-Instanz um eine EC2-Instanz, die man selbst konfigurieren und betreiben muss. Diese bietet mehr Flexibilität, da sie auch für zusätzliche Aufgaben, wie für spezielles Routing, genutzt werden kann und direkten Zugriff auf Sicherheitsgruppen erlaubt. Allerdings ist die NAT-Instanz aufwendiger in der Einrichtung und Wartung, da man sich selbst um vieles kümmern muss, um Ausfallrisiken zu minimieren.[144]

Zusammengefasst eignet sich ein NAT-Gateway vor allem dann, wenn hohe Verfügbarkeit und geringer Verwaltungsaufwand entscheidend sind, während eine NAT-Instanz dann sinnvoll ist, wenn spezielle Anpassungen benötigt werden und der Traffic gering ist.

8.3.3 Amazon API Gateway oder Application Load Balancer

Sowohl Amazon API Gateway als auch ein ALB können als HTTP-Endpunkt für eingehende API-Aufrufe dienen und diese weiterleiten. API Gateway bietet dabei eine Reihe von API- und WebSocket-spezifische Features. Ein ALB kann ebenfalls Anfragen empfangen und weiterleiten, skaliert dabei aber ohne feste Begrenzungen, was bei hohem Traffic zu Kostenvorteilen und besserer Leistung führen kann.[145] Allerdings fehlen beim ALB einige der erweiterten Funktionen des API-Managements, die man dann selbst entwickeln müsste.

[144] Vgl. Amazon Web Services, Inc., 2024a, S. 305 f.
[145] Vgl. Shrivastava et al., 2023, S.205.

Amazon API Gateway eignet sich, wenn ein detailliertes API-Management, viele integrierte Sicherheits- und Validierungsfunktionen und direkte Service-Integrationen benötigt werden. ALB ist hingegen die bessere Wahl bei sehr hohem Traffic und wenn einfache Weiterleitung und Kosteneffizienz im Vordergrund stehen.[146]

8.3.4 Amazon EKS oder Amazon ECS

Amazon EKS und Amazon ECS sind zwei verschiedene Ansätze zur Ausführung von Containern. Amazon EKS basiert auf Kubernetes, was den Einsatz eines großen Funktionsumfangs und vieler Anpassungsmöglichkeiten erlaubt. Dies ist besonders nützlich, wenn bestehende Kubernetes-Workloads genutzt werden oder spezielle Anforderungen an das Scheduling und Netzwerk bestehen. Allerdings ist EKS komplexer und verursacht zusätzliche Kosten, beispielsweise für den Betrieb der Kubernetes-Kontrollebene.

Im Gegensatz dazu ermöglicht Amazon ECS den Betrieb von Containern ohne die Verwaltung von EC2-Instanzen. Dieser Dienst vereinfacht die Einrichtung und Integration in weitere AWS-Dienste, wie Load Balancer, was zu einem geringeren Verwaltungsaufwand führt. ECS eignet sich daher gut, wenn eine einfache und kosteneffiziente Container-Lösung benötigt wird und keine spezifischen Kubernetes-Funktionen erforderlich sind.

AWS Fargate erweitert diese Möglichkeiten, indem es den Betrieb von Containern ohne jegliche Verwaltung der zugrunde liegenden Infrastruktur serverlos ermöglicht. Dabei entfällt die Notwendigkeit, Server bereitzustellen und zu skalieren. Die Kombination von Fargate mit Amazon ECS/EKS bietet zusätzliche Flexibilität und trägt zur Kostenoptimierung bei.[147]

[146] Vgl. Shrivastava et al., 2023, S. 509, 579.
[147] Vgl. Singh, Deepak: Amazon ECS oder Amazon EKS: AWS Containerservices sinnvoll nutzen, in: AWS Germany, 22.08.2023, https://aws.amazon.com/de/blogs/germany/amazon-ecs-oder-amazon-eks-sinnvoll-nutzen/ (abgerufen am 26.03.2025); Vgl. Shrivastava et al., 2023, S. 501.

8.3.5 Amazon Aurora oder Amazon DynamoDB

Amazon Aurora eignet sich gut, wenn komplexe SQL-Abfragen, wie Joins und tabellenübergreifende Updates, benötigt werden. Es ist vor allem dann die richtige Wahl, wenn die Datenstruktur stabil ist und die Anwendung von einem festen Schema profitiert. Amazon DynamoDB hingegen ist ideal, wenn es auf hohe Skalierbarkeit, niedrige Latenzen und ein serverloses Modell ankommt. Es eignet sich besonders für Szenarien mit klar definierten Zugriffsmustern, einfachen Abfragen und Datenmodellen mit Schlüssel-Wert-Paaren.[148]

8.4 Schritt 4: Testen und Optimieren

Ist die Architektur aufgebaut und ihre Funktionen simulationsfähig, wurden die wesentlichen Schritte umgesetzt. Nun gilt es zu überprüfen, ob die nicht-funktionalen Anforderungen korrekt abgeschätzt wurden. Durch die Simulation realitätsnaher Spielsituationen, Lastspitzen und Ausfallszenarien kann die Skalierbarkeit und Belastbarkeit des Systems getestet und gegebenenfalls angepasst werden. Mithilfe des AWS Cost Explorer lassen sich zudem die entstehenden Kosten präzise vorhersagen und bewerten.[149] Die Testergebnisse sollten genutzt werden, um Schwachstellen zu identifizieren und Anpassungen vorzunehmen. Dieser Prozess sollte oft wiederholt werden, um eine optimale Systemperformance sicherzustellen.

[148] Vgl. Shrivastava et al., 2023, S. 262 f.
[149] Vgl. AWS Cost Explorer: in: AWS Produkte, o. D., [online] https://aws.amazon.com/aws-cost-management/aws-cost-explorer/ (abgerufen am 28.03.2025).

8.5 Lösungs-Architektur-Dokument

Diese Arbeit fokussiert sich auf die zentralen Schritte zur Erstellung einer passenden AWS-Architektur für asynchrone Multiplayer-Spiele. Die durch die vier Schritte entworfene Architektur ist jedoch nur eine von vielen wichtigen Teilen eines Lösungs-Architektur-Dokuments (SAD). Dieses Dokument befindet sich zwar außerhalb des Umfangs dieser Arbeit, sollte aber abschließend der Vollständigkeit halber und im Hinblick auf einen Ausblick kurz beschrieben werden.

Ein SAD bietet einen umfassenden Überblick über die gesamte Lösung und stellt sicher, dass alle Beteiligten von technischen Teams bis hin zu Kunden und Management dieselbe Vision teilen. Der Zweck des Dokuments liegt darin, die End-to-End-Lösung allen Stakeholdern nahezubringen und die Abstimmung mit den Geschäftsanforderungen und Systemvorgaben zu gewährleisten. Es unterstützt die Planung, die Abschätzungen sowie das Risikomanagement und dokumentiert zudem alle Architekturbewertungen und die entsprechenden Migrationspläne. Dabei teilt sich das Dokument in folgende Sichten auf.[150]

Aus der geschäftlichen Sicht wird der Mehrwert der Lösung vermittelt und es wird aufgezeigt, wie sich die einzelnen Anwendungsfälle in den Gesamtprozess einfügen. Die logische Sicht erläutert die inhaltlichen Komponenten und deren Zusammenwirken im System, während die Prozesssicht detailliert beschreibt, wie die kritischen Abläufe im Zusammenspiel funktionieren. Die technische Sichtweise erklärt die getroffenen technologischen Entscheidungen und beschreibt, wie das System aufgebaut ist.[151] Somit würden sich die in dieser Arbeit erarbeiteten Architektur-Lösungen bzw. Architektur-Entscheidungen in dieser Sicht wiederfinden.

[150] Vgl. Shrivastava/Srivastav, 2024, S. 477 f.
[151] Vgl. ebd., S. 478 ff.

9 Zusammenfassung

Zusammenfassend lässt sich feststellen, dass das gesetzte Ziel erfolgreich erreicht wurde. Es wurde ein präziser Leitfaden für die Entwicklung cloudbasierter, asynchroner Multiplayer-Spielarchitekturen für AWS entwickelt. Diese Arbeit stützt sich dabei auf eine Analyse von brauchbaren AWS-Diensten. Durch die empirische Untersuchung relevanter Fallbeispiele aus der Spieleindustrie konnte gezeigt werden, dass für diese Spiele in Cloud-Umgebungen dank der zahlreichen Angebote konkrete und praxisnahe Lösungs-Architekturen erarbeitet werden können.

Im Rahmen der Untersuchung wurden vier Architekturansätze herausgearbeitet: eine klassische, eine serverlose, eine containerbasierte und eine auf Amazon GameLift basierende Architektur. Diese Ansätze wurden hinsichtlich ihrer Skalierbarkeit, Implementierungsdauer, Wartungs-komplexität, Betriebskosten, Zukunftssicherheit und Anpassungsfähigkeit miteinander verglichen. Der daraus resultierende Leitfaden gibt Entwicklern klare Empfehlungen, welche Architekturvariante unter welchen Bedingungen optimal eingesetzt werden kann.

Die Arbeit leistet somit einen Beitrag zur Weiterentwicklung moderner Cloud-Lösungen für asynchrone Multiplayer-Spiele. Sie zeigt, dass das große Angebot an AWS-Diensten es ermöglicht, passende und zukunftssichere Systeme zu entwickeln. Gleichzeitig bietet der erarbeitete Leitfaden eine praxisnahe Orientierungshilfe, die Entwicklern helfen soll, fundierte Entscheidungen zu treffen, um ihre asynchronen Multiplayer-Spiele effizient und kostengünstig zu realisieren.

10 Ausblick

Im Ausblick ergeben sich Ausgangspunkte für weiterführende Forschungsarbeiten. Zwar leistet der in dieser Arbeit entwickelte Leitfaden einen wichtigen Beitrag zur Erstellung asynchroner Multiplayer-Spielarchitekturen mit AWS, doch können darauf aufbauend weitere Fragestellungen benannt werden.

- So könnte der Einsatz von AWS-Diensten für moderne Technologien, wie künstliche Intelligenz und Machine Learning, im Kontext asynchroner Multiplayer-Spiele untersucht werden, um beispielsweise Matchmaking-Prozesse weiter zu optimieren.
- Darüber hinaus können die entwickelten Architekturansätze auf weitere Online-Varianten, wie Echtzeit- oder hybride Multiplayer-Architekturen, übertragen werden, um deren spezifische Herausforderungen und Potenziale in diesen Bereichen zu analysieren.
- Auch der Aspekt der Sicherheit und der Datenspeicherung in komplexen asynchronen Multiplayer-Spielen kann in zukünftigen Arbeiten vertieft werden, um die Spielzustände optimierter und kostengünstiger über längere Zeit speichern zu können.
- Schließlich ist es interessant, ob eine tiefergehende Untersuchung der fehlenden Teile eines Lösungs-Architektur-Dokuments für asynchrone Multiplayer-Spiele sinnvoll ist. Ist es möglich, zugeschnittene Entwicklungsprozesse oder optimierte Kommunikationswege zwischen Spielteams, Kunden und Management speziell für die Entwicklung solcher Spiele zu definieren?

Insgesamt bildet diese Arbeit eine solide Grundlage, von der aus zukünftige empirische und theoretische Untersuchungen zu cloudbasierten Architektur-Lösungen für asynchrone Multiplayer-Spiele weiter vorangetrieben werden können.

Literaturverzeichnis

Amazon API Gateway: in: AWS Produkte, o. D., [online] https://aws.amazon.com/api-gateway/ (abgerufen am 11.01.2025).

Amazon Aurora: in: AWS Produkte, o. D., [online] https://aws.amazon.com/rds/aurora/ (abgerufen am 04.02.2025).

Amazon CloudFront: in: AWS Produkte, o. D., [online] https://aws.amazon.com/cloudfront/ (abgerufen am 02.02.2025).

Amazon DynamoDB: in: AWS Produkte, o. D., [online] https://aws.amazon.com/dynamodb/ (abgerufen am 04.02.2025).

Amazon Elastic Compute Cloud: in: AWS Produkte, o. D., [online] https://aws.amazon.com/de/ec2/ (abgerufen am 24.01.2024).

Amazon Elastic Kubernetes Service: in: AWS Produkte, o. D., [online] https://aws.amazon.com/eks/ (abgerufen am 24.01.2025).

Amazon ElastiCache: in: AWS Produkte, o. D., [online] https://aws.amazon.com/elasticache/ (abgerufen am 04.02.2025).

Amazon EventBridge: in: AWS Produkte, o. D., [online] https://aws.amazon.com/eventbridge/ (abgerufen am 28.01.2025).

Amazon GameLift: in: AWS Produkte, o. D., [online] https://aws.amazon.com/gamelift/ (abgerufen am 15.12.2024).

Amazon Route 53: in: AWS Produkte, o. D., [online] https://aws.amazon.com/route53/ (abgerufen am 10.01.2025).

Amazon S3: in: AWS Produkte, o. D., [online] https://aws.amazon.com/s3/ (abgerufen am 04.02.2025).

Amazon Simple Notification Service: in: AWS Produkte, o. D., [online] https://aws.amazon.com/sns/ (abgerufen am 27.01.2025).

Amazon Simple Queue Service: in: AWS Produkte, o. D., [online] https://aws.amazon.com/sqs/ (abgerufen am 27.01.2025).

Amazon Virtual Private Cloud (VPC): in: AWS Produkte, o. D., [online] https://aws.amazon.com/vpc/ (abgerufen am 23.01.2025).

Amazon Web Services, Inc.: Amazon GameLift - Hosting Guide, 14.01.2025a, [PDF] https://docs.aws.amazon.com/pdfs/gamelift/latest/developerguide/gamelift-dg.pdf (abgerufen am 25.01.2025).

Amazon Web Services, Inc.: Amazon Virtual Private Cloud - User Guide, 09.12.2024a, [PDF] https://docs.aws.amazon.com/pdfs/vpc/latest/userguide/vpc-ug.pdf (abgerufen am 23.01.2025).

Amazon Web Services, Inc.: Architecture Icon Page, o. D.a, [online] https://aws.amazon.com/architecture/icons/ (abgerufen am 22.01.2025).

Amazon Web Services, Inc.: Asynchronous-Online-Gaming - Basic, 24.03.2021a, [PDF] https://d1.awsstatic.com/architecture-diagrams/Asynchronous-Online-Gaming%20-%20Basic.pdf (abgerufen am 17.01.2025).

Amazon Web Services, Inc.: AWS case study - Gametion, 2020, [online] https://aws.amazon.com/solutions/case-studies/gametion/ (abgerufen am 11.01.2025).

Amazon Web Services, Inc.: AWS Lambda - Developer Guide, 20.02.2025b, [PDF] https://docs.aws.amazon.com/pdfs/lambda/latest/dg/lambda-dg.pdf.

Amazon Web Services, Inc.: AWS Whitepaper: Overview of Amazon Web Services, 27.08.2024b, [PDF] https://docs.aws.amazon.com/pdfs/whitepapers/latest/aws-overview/aws-overview.pdf.

Amazon Web Services, Inc.: Delivering Engaging Games at Scale Using AWS with Whatwapp, 2023a, [online] https://aws.amazon.com/solutions/case-studies/whatwapp-case-study/ (abgerufen am 12.01.2025).

Amazon Web Services, Inc.: Games Industry Lens - AWS Well-Architected Framework, 19.11.2021b, [PDF] https://docs.aws.amazon.com/pdfs/wellarchitected/latest/games-industry-lens/games-industry-lens.pdf.

Amazon Web Services, Inc.: How Supercell Runs Giant Games with Tiny Teams on AWS, 2018, [online] https://aws.amazon.com/solutions/case-studies/supercell-video/ (abgerufen am 09.01.2025).

Amazon Web Services, Inc.: MARVEL SNAP scales player matchmaking with Amazon GameLift FlexMatch, 2023b, [online] https://aws.amazon.com/solutions/case-studies/second-dinner-nuverse-case-study/ (abgerufen am 12.01.2025).

Amazon Web Services, Inc.: Start building on AWS today, o. D.b, [online] https://aws.amazon.com/ (abgerufen am 18.12.2024).

Amazon Web Services, Inc.: Supercell | Edge Services | AWS Case study, 2022, [online] https://aws.amazon.com/solutions/case-studies/supercell-edge-services-case-study/ (abgerufen am 09.01.2025).

Amazon Web Services, Inc.: Supercell Case Study, 2014, [online] https://aws.amazon.com/solutions/case-studies/supercell/ (abgerufen am 09.01.2025).

Amazon Web Services, Inc.: Supercell Leverages AWS for Seamless and Scalable Gaming Experience, 2024c, [online] https://aws.amazon.com/solutions/case-studies/supercell-aurora-case-study/ (abgerufen am 09.01.2025).

Amazon Web Services, Inc.: What is Amazon S3?, o. D.c, [online] https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html (abgerufen am 05.02.2025).

Amazon Web Services, Inc.: What is Architecture Diagramming?, o. D.d, [online] https://aws.amazon.com/what-is/architecture-diagramming/ (abgerufen am 20.01.2025).

Anantha, Mounika: AWS Reference Architecture for Asynchronous Online Gaming, in: Medium, 31.10.2024, [online] https://medium.com/@ananthamounika123/aws-reference-architecture-for-asynchronous-online-gaming-b457fe763567 (abgerufen am 28.02.2025).

Apperson, Lem: Beginning Game Development: Client-Server Architecture, in: Medium, 17.11.2024, [online] https://medium.com/@lemapp09/beginning-game-development-client-server-architecture-1b7676d80dea (abgerufen am 08.01.2025).

AWS Architektursymbolpaket: 07.02.2025, [online] https://d1.awsstatic.com/webteam/architecture-icons/q1-2025/Asset-Package_02072025.dee42cd0a6eaacc3da1ad9519579357fb546f803.zip.

AWS Cost Explorer: in: AWS Produkte, o. D., [online] https://aws.amazon.com/aws-cost-management/aws-cost-explorer (abgerufen am 28.03.2025).

AWS Fargate: in: AWS Produkte, o. D., [online] https://aws.amazon.com/fargate/ (abgerufen am 24.01.2025).

AWS Global Accelerator: in: AWS Produkte, o. D., [online] https://aws.amazon.com/global-accelerator/ (abgerufen am 11.01.2025).

AWS Lambda: in: AWS Produkte, o. D., [online] https://aws.amazon.com/lambda/ (abgerufen am 24.01.2025).

AWS Pricing Calculator: o. D., [online] https://calculator.aws/ (abgerufen am 03.03.2025).

AWS Step Functions: in: AWS Produkte, o. D., [online] https://aws.amazon.com/step-functions/ (abgerufen am 27.01.2025).

Bruce, Tim: Building a serverless multi-player game that scales, in: AWS Compute Blog, 24.02.2021, [online] https://aws.amazon.com/de/blogs/compute/building-a-serverless-multiplayer-game-that-scales/ (abgerufen am 13.01.2025).

BSI (Bundesamt für Sicherheit in der Informationstechnik): Cloud Computing Grundlagen, o. D., [online] https://www.bsi.bund.de/dok/6622124 (abgerufen am 12.12.2024).

Chan, Tak Yu: Route 53 vs Cloudfront vs Global Accelerator, in: Medium, 06.01.2022, [online] https://franky-46708.medium.com/route-53-vs-cloudfront-vs-global-accelerator-96277a65b61f (abgerufen am 26.03.2025).

Clark, Jack: How Amazon exposed its guts: The History of AWS's EC2, in: ZDNET, 07.06.2012, [online] https://www.zdnet.com/article/how-amazon-exposed-its-guts-the-history-of-awss-ec2/ (abgerufen am 18.12.2024).

Customer Success Stories: o. D., [online] https://aws.amazon.com/solutions/case-studies/ (abgerufen am 21.12.2024).

Elastic Load Balancing: in: AWS Produkte, o. D., [online] https://aws.amazon.com/elasticloadbalancing/ (abgerufen am 11.01.2025).

Golon, Piotr: Horizontal scaling and vertical scaling in AWS, in: BlowStack, 26.01.2024, [online] https://blowstack.com/blog/horizontal-scaling-and-vertical-scaling-in-aws (abgerufen am 28.02.2025).

Horan, Stephen: COVID-19 hat eine Gaming-Revolution gestartet und die Cloud wird den Prozess noch weiter beschleunigen: Von lokalen Spielen hin zu unendlicher Skalierbarkeit, in: Rackspace, 11.01.2021, [online] https://www.rackspace.com/de/blog/covid-kickstarted-gaming-revolution-and-now-cloud-will-accelerate-it (abgerufen am 12.12.2024).

Kratzke, Nane: A Brief History of Cloud Application Architectures, 14.08.2018, [online] doi:10.3390/app8081368.

Labes, Stine: Grundlagen des Cloud Computing – Konzept und Bewertung von Cloud Computing, Universitätsverlag der TU Berlin, Bd. 1, 11.2012, [online] https://api-depositonce.tu-berlin.de/server/api/core/bitstreams/02384947-8d6d-47dd-9dfc-163b580abed4/content.

Li, Yongkang/Yanying Lin/Yang Wang/Kejiang Ye/Chengzhong Xu: Serverless Computing: State-of-the-Art, Challenges and Opportunities, in: IEEE Transactions On Services Computing, Bd. 16, Nr. 2, 12.04.2022, [online] doi:10.1109/tsc.2022.3166553, S. 1522–1539.

Lonis, Peter: The Power of Cloud Computing in the Gaming Industry, in: Leaseweb Blog, 29.06.2023, [online] https://blog.leaseweb.com/2023/06/29/the-power-of-cloud-computing-in-the-gaming-industry/ (abgerufen am 12.12.2024).

Secchi, Marco.: Multiplayer Game Development with Unreal Engine 5, Packt Publishing, 06.10.2023.

Shrivastava, Saurabh/Neelanjali Srivastav: Solutions Architect's Handbook, 3. Aufl., Packt Publishing, 29.03.2024.

Shrivastava, Saurabh/Neelanjali Srivastav/Alberto Artasanchez/Imtiaz Sayed: AWS for Solutions Architects, 2. Aufl., Packt Publishing, 28.04.2023.

Singh, Deepak: Amazon ECS oder Amazon EKS: AWS Containerservices sinnvoll nutzen, in: AWS Germany, 22.08.2023, [online] https://aws.amazon.com/de/blogs/germany/amazon-ecs-oder-amazon-eks-sinnvoll-nutzen/ (abgerufen am 26.03.2025).

Singh, Vindeep/Sateesh K Peddoju: Container-based microservice architecture for cloud applications, IEEE, 05.2017, [PDF] doi:10.1109/ccaa.2017.8229914.

Stojanov, Z./I. Hristoski/J. Stojanov/A. Stojkov: A Tertiary Study on Microservices: Research Trends and Recommendations, in: Springer, Bd. 49, Nr. 8, 24.01.2024, [online] doi:10.1134/s0361768823080200, S. 796–821.

Synergy Research Group: Cloud Market Growth Stays Strong in Q2 While Amazon, Google and Oracle Nudge Higher, 01.08.2024, [online] https://www.srgresearch.com/articles/cloud-market-growth-stays-strong-in-q2-while-amazon-google-and-oracle-nudge-higher.

Velepucha, Victor/Pamela Flores: A Survey on Microservices Architecture: Principles, Patterns and Migration Challenges, in: IEEE Access, Bd. 11, 01.01.2023, [online] doi:10.1109/access.2023.3305687, S. 88339–88358.

A Anhang

A.1 Grundlegende Annahmen der Kostenabschätzung

Anzahl Anfragen pro Spieler pro Tag	500		N_{Req}		
Datengröße einer Anfrage	5	KB	S_{Req}		
Bearbeitungsdauer einer Anfrage	50	ms	t_{Req}		
Anzahl an Microservices für verteilte Systeme	20		N_{MS}		
Wechselkurs USD/EUR (stand 02.2025):	0,96	USD			
Anzahl an Spieler pro Tag	100	1.000	5.000	10.000	N_{Player}
Anzahl Anfragen pro Monat	1.500.000	15.000.000	75.000.000	150.000.000	$N_{RM} = N_{RD} * 30$
Anzahl Anfragen pro Tag	50.000	500.000	2.500.000	5.000.000	$N_{RD} = N_{Req} * N_{Player}$
Anzahl Anfragen pro Stunde	2.083	20.833	104.167	208.333	$N_{RH} = \frac{N_{RD}}{24}$
Anzahl Anfragen pro Sekunde	1	6	29	58	$N_{RS} = \frac{N_{RH}}{3600}$
Anzahl Anfragen pro Microservice pro Monat	75.000	750.000	3.750.000	7.500.000	$N_{ReqMS} = \frac{N_{RM}}{N_{MS}}$
ø Anfragevolumen pro Stunde in Kilobyte	10.417	104.167	520.833	1.041.667	$D_{KB} = S_{Req} * N_{RH}$
ø Anfragevolumen pro Stunde in Gigabyte	0,01	0,10	0,52	1,04	$D_{GB} = \frac{D_{KB}}{1000000}$
Datenbankgröße in Gigabyte	5	50	250	500	$Data_{GB}$

A.2 Eingaben/Ergebnisse des AWS-Kostenrechner zur Abschätzung der Betriebskosten

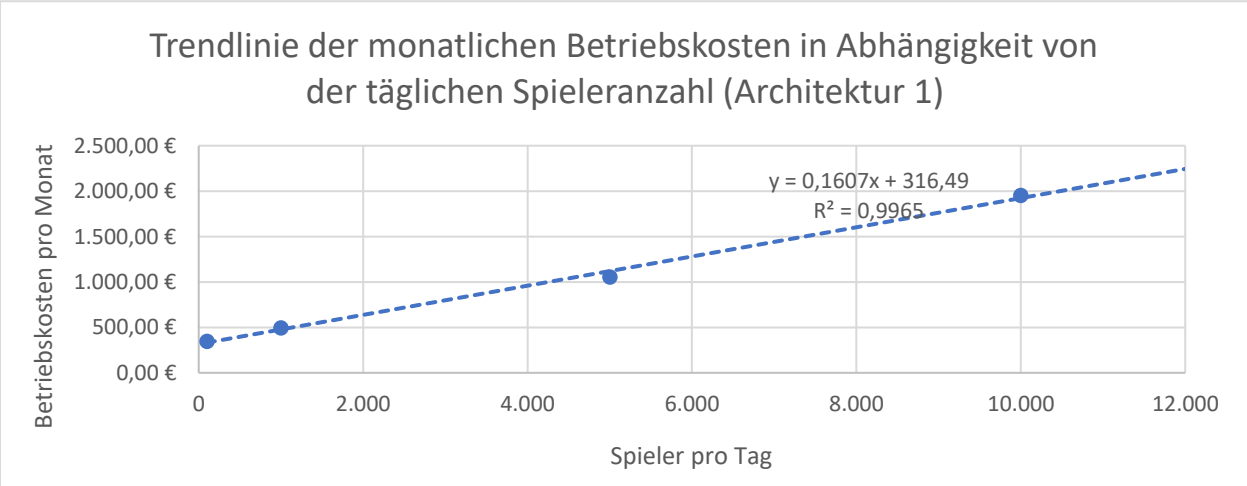
Anzahl Spieler pro Tag	100	1.000	5.000	10.000
Amazon Route 53 (genutzt in A1 & A3)				
Gehostete Zonen	2	2	2	2
Traffic Flow	1	1	1	1
Standard-Abfragen	1.500.000	15.000.000	75.000.000	150.000.000
Von AWS errechnete Kosten in USD	51,60	57,00	81,00	111,00
Amazon VPC NAT-Gateway (genutzt in A1 & A3)				
Anzahl der NAT-Gateways	2	2	2	2
Verarbeitete Daten pro NAT-Gateway (in GB)	0,01	0,1	0,52	1,04
Von AWS errechnete Kosten in USD	75,92	75,94	75,98	76,02
ELB als ALB (genutzt in A1)				
Anzahl der Application Load Balancers	2	2	2	2
Verarbeitete Byte (EC2-Instances) und IP-Adressen als Ziele (in GB)	0,01	0,1	0,52	1,04
Durchschnittliche Anzahl der neuen Verbindungen pro ALB pro Sek	1	6	29	58
Von AWS errechnete Kosten in USD	39,89	21,11	52,97	66,52
ELB als ALB (genutzt in A3)				
Anzahl der Application Load Balancers	1	1	1	1
Verarbeitete Byte (EC2-Instances) und IP-Adressen als Ziele (in GB)	0,01	0,1	0,52	1,04
Durchschnittliche Anzahl der neuen Verbindungen pro ALB pro Sek	1	6	29	58
Von AWS errechnete Kosten in USD	19,94	42,22	26,48	33,26
Amazon EC2 (genutzt in A1 & A3)				
Anzahl der Instanzen	4	4	4	4
Instanztyp	t3.small	t3.medium	t3.large	t3.xlarge
Zahlungsoptionen	On Demand	On Demand	On Demand	On Demand
Von AWS errechnete Kosten in USD	70,08	140,16	280,32	560,64
Amazon Aurora (genutzt in A1 & A3)				
Nodes	2	2	2	2
Instanztyp	db.t3.small	db.t3.medium	db.r3.large	db.r3.xlarge
Pricing Model	On Demand	On Demand	On Demand	On Demand
RDS-Proxy	Ja	Ja	Ja	Ja
Speichermenge (in GB)	5	50	250	500
IO-Spitzenrate pro Sekunde	1	6	29	58
Dauer der IO-Spitzenaktivität (in Stunden pro Monat)	730	730	730	730
Von AWS errechnete Kosten in USD	123,81	202,14	610,08	1220,15
Amazon API Gateway (genutzt in A2)				
REST-API Anforderungen	1.500.000	15.000.000	75.000.000	150.000.000
REST-API Cache-Speichergröße (GB)	1,6	6,1	13,5	28,4
Von AWS errechnete Kosten in USD	33,29	201,50	460,00	920,00
Amazon API Gateway (genutzt in A4)				
HTTP-API Anforderungen	1.500.000	15.000.000	75.000.000	150.000.000
Durchschnittliche Größe jeder Anforderung (in KB)	50	50	50	50
Von AWS errechnete Kosten in USD	1,80	18,00	90,00	180,00
AWS Lambda (genutzt in A2 & A4)				

Architektur	x86	x86	x86	x86
Anzahl der Anforderungen pro Monat	150.000	1.500.000	7.500.000	15.000.000
Dauer jeder Anforderung (in ms)	50	50	50	50
Größe des zugewiesenen Speichers (in MB)	128	256	512	1024
Von AWS errechnete Kosten für A2 in USD * 20	1,00	12,20	92,80	310,80
Von AWS errechnete Kosten für A4 in USD * 2	0,01	1,22	9,28	31,08
AWS Step Functions (genutzt in A2)				
Workflow-Anfragen pro Monat	100.000	1.000.000	5.000.000	10.000.000
Zustandsübergänge pro Workflow	5	5	5	5
Von AWS errechnete Kosten in USD	12,40	124,90	624,90	1249,90
Amazon SNS (genutzt in A2)				
Amazon Web Services Lambda Anforderungen	1.500.000	15.000.000	75.000.000	150.000.000
Von AWS errechnete Kosten in USD	0,25	7,00	37,00	74,50
Amazon SNS (genutzt in A3)				
HTTP-/HTTPS-Benachrichtigungen Anforderungen	1.500.000	15.000.000	75.000.000	150.000.000
Von AWS errechnete Kosten in USD	1,09	15,94	81,94	164,44
Amazon SQS (genutzt in A2)				
Standard-Warteschlangenanforderungen pro Monat	150.000	1.500.000	7.500.000	15.000.000
Von AWS errechnete Kosten in USD	0,11	0,29	2,69	5,69
Amazon DynamoDB (genutzt in A2 & A4)				
DynamoDB-Funktionen	On Demand	On Demand	On Demand	On Demand
Datenspeichergröße	5	50	250	500
Durchschnittliche Elementgröße (alle Attribute)	50	50	50	50
Standardschreibvorgänge in %	95	95	95	95
Transaktionale Schreibvorgänge in %	5	5	5	5
Anzahl der Schreibvorgänge in Mio. pro Monat	0,5	5	25	50
Eventuell konsistenter Prozentsatz Lesen	95	95	95	95
Sehr konsistenter Prozentsatz Lesen	5	5	5	5
Anzahl der Lesevorgänge in Mio. pro Monat	1	10	50	100
Von AWS errechnete Kosten in USD	18,94	178,11	946,79	1893,60
Amazon EKS (genutzt in A3)				
Anzahl der EKS-Cluster	1	1	1	1
Von AWS errechnete Kosten in USD	73,00	73,00	73,00	73,00
Amazon EventBridge (genutzt in A3)				
Größe der Payload in KB	50	50	50	50
Anzahl der benutzerdefinierten Ereignisse pro Monat	1.500.000	15.000.000	75.000.000	150.000.000
Anzahl der Aufrufe pro Monat	1.500.000	15.000.000	75.000.000	150.000.000
Von AWS errechnete Kosten in USD	1,86	18,60	93,00	186,00
Amazon GameLift (genutzt in A4)				
Spitzenleistung für gleichzeitige Spieler	100	1000	5000	10000
Spielsitzungen pro Instance	5	50	250	500
Spieler pro Spielsitzung	2	2	2	2
Instanz-Leerlaufpuffer %	10	10	10	10

Spot-Instance %	50	50	50	50
Instanztyp	c6g.medium	c6g.large	c6g.xlarge	c6g.2xlarge
DTO pro Spieler in KB pro Sekunde	5	5	5	5
Von AWS errechnete Kosten in USD	122,21	512,56	2030,08	3998,12
AWS Global Accelorator (genutzt in A4)				
Festpreis in USD	18,00	18,00	18,00	18,00

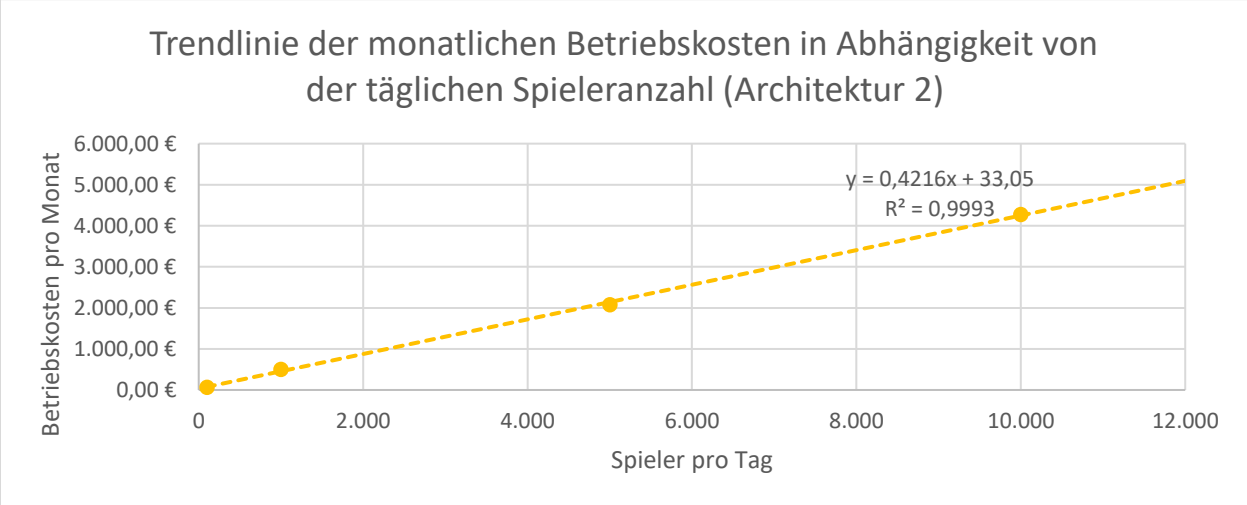
A.2.1 Kosten und Kostenfunktion von A1

Route 53					
Kosten pro Monat	in USD	51,60 USD	57,00 USD	81,00 USD	111,00 USD
Kosten pro Monat	in EUR	49,54 €	54,72 €	77,76 €	106,56 €
2 VPC NAT-Gateway					
Kosten pro Monat	in USD	75,92 USD	75,94 USD	75,98 USD	76,02 USD
Kosten pro Monat	in EUR	72,88 €	72,90 €	72,94 €	72,98 €
2 ELB als ALB					
Kosten pro Monat	in USD	39,89 USD	42,22 USD	52,97 USD	66,52 USD
Kosten pro Monat	in EUR	38,29 €	40,53 €	50,85 €	63,86 €
2x2 EC2 (On Demand)	Instanztyp	t3.small	t3.medium	t3.large	t3.xlarge
Kosten pro Monat	in USD	70,08 USD	140,16 USD	280,32 USD	560,64 USD
Kosten pro Monat	in EUR	67,28 €	134,55 €	269,11 €	538,21 €
2 Aurora DB MySQL Knoten	Instanztyp	db.t3 small	db.t3.medium	db.r3.large	db.r3.xlarge
Kosten pro Monat	in USD	123,81 USD	202,14 USD	610,08 USD	1.220,15 USD
Kosten pro Monat	in EUR	118,86 €	194,05 €	585,68 €	1.171,34 €
Totale Kosten für Architektur 1	Spieler pro Tag	100	1.000	5.000	10.000
Kosten pro Monat	in USD	361,30 USD	517,46 USD	1.100,35 USD	2.034,33 USD
Kosten pro Monat	in EUR	346,85 €	496,76 €	1.056,34 €	1.952,96 €



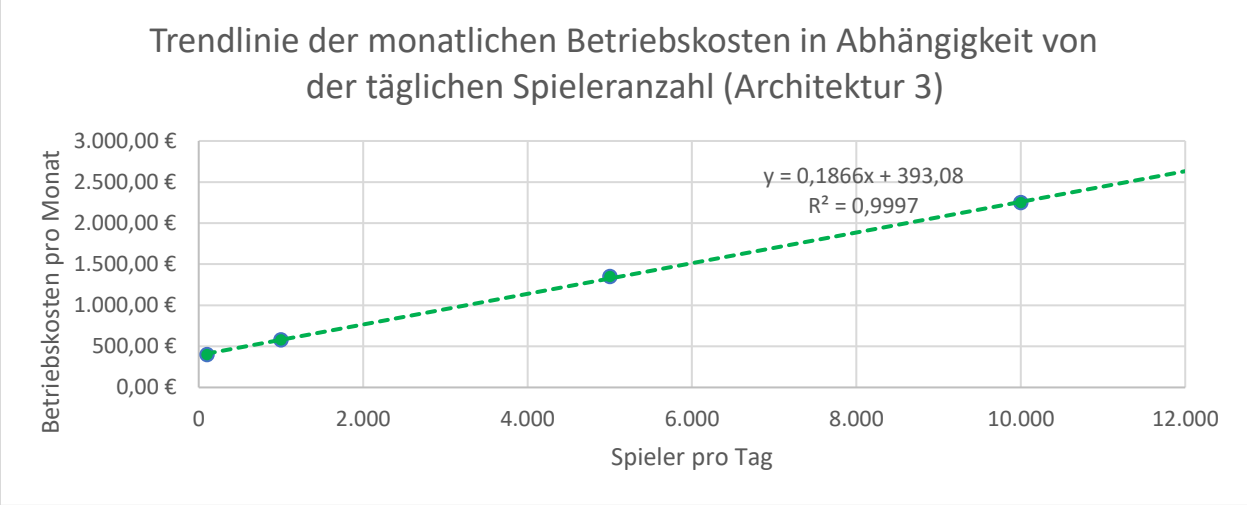
A.2.2 Kosten und Kostenfunktion von A2

API Gateway					
Kosten pro Monat	in USD	33,29 USD	201,50 USD	460,00 USD	920,00 USD
Kosten pro Monat	in EUR	31,96 €	193,44 €	441,60 €	883,20 €
20x Lambda-Funktionen					
Kosten pro Monat	in USD	1,00 USD	12,20 USD	92,80 USD	310,80 USD
Kosten pro Monat	in EUR	0,96 €	11,71 €	89,09 €	298,37 €
Step Functions					
Kosten pro Monat	in USD	12,40 USD	124,90 USD	624,90 USD	1.249,90 USD
Kosten pro Monat	in EUR	11,90 €	119,90 €	599,90 €	1.199,90 €
SNS					
Kosten pro Monat	in USD	0,25 USD	7,00 USD	37,00 USD	74,50 USD
Kosten pro Monat	in EUR	0,24 €	6,72 €	35,52 €	71,52 €
SQS					
Kosten pro Monat	in USD	0,09 USD	0,29 USD	2,69 USD	5,69 USD
Kosten pro Monat	in EUR	0,09 €	0,28 €	2,58 €	5,46 €
Dynamo DB (On Demand)					
Kosten pro Monat	in USD	18,94 USD	178,11 USD	946,79 USD	1.893,60 USD
Kosten pro Monat	in EUR	18,18 €	170,99 €	908,92 €	1.817,86 €
Totale Kosten für Architektur 2	Spieler pro Tag	100	1.000	5.000	10.000
Kosten pro Monat	in USD	65,97 USD	524,00 USD	2.164,18 USD	4.454,49 USD
Kosten pro Monat	in EUR	63,33 €	503,04 €	2.077,61 €	4.276,31 €



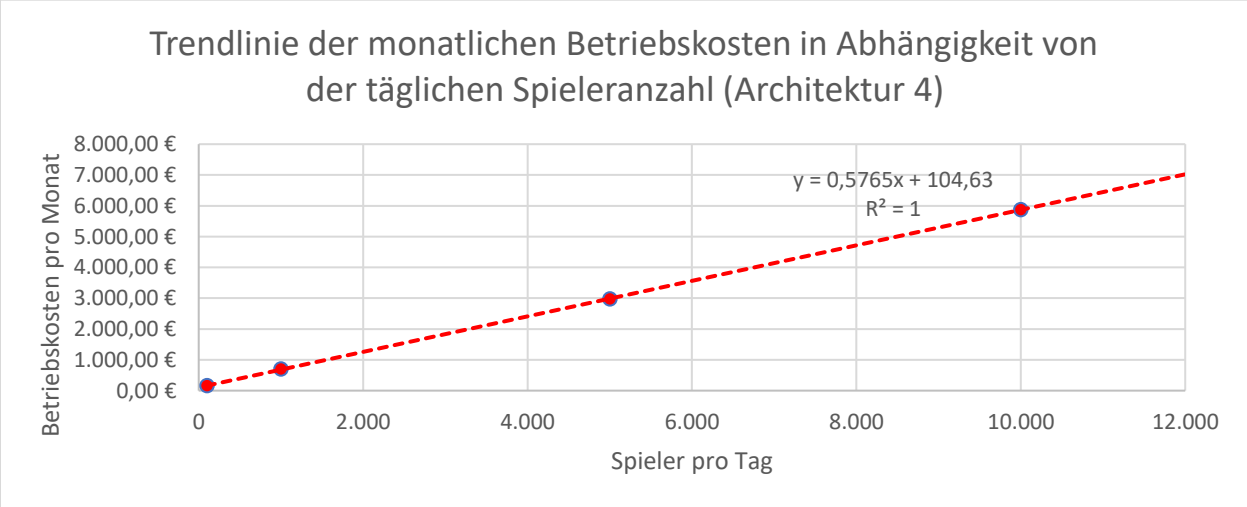
A.2.3 Kosten und Kostenfunktion von A3

Route 53					
Kosten pro Monat	in USD	51,60 USD	57,00 USD	81,00 USD	111,00 USD
Kosten pro Monat	in EUR	49,54 €	54,72 €	77,76 €	106,56 €
2 VPC NAT-Gateway					
Kosten pro Monat	in USD	75,92 USD	75,94 USD	75,98 USD	76,02 USD
Kosten pro Monat	in EUR	72,88 €	72,90 USD	72,94 USD	72,98 USD
1 ELB als ALB					
Kosten pro Monat	in USD	19,94 USD	21,11 USD	26,48 USD	33,26 USD
Kosten pro Monat	in EUR	19,14 €	20,27 USD	25,42 USD	31,93 USD
1 EKS Cluster					
Kosten pro Monat	in USD	73,00 USD	73,00 USD	73,00 USD	73,00 USD
Kosten pro Monat	in EUR	70,08 €	70,08 €	70,08 €	70,08 €
2x2 EC2 (On Demand)	Instanztyp	t3.small	t3.medium	t3.large	t3.xlarge
Kosten pro Monat	in USD	70,08 USD	140,16 USD	280,32 USD	560,64 USD
Kosten pro Monat	in EUR	67,28 €	134,55 €	269,11 €	538,21 €
EventBridge					
Kosten pro Monat	in USD	1,86 USD	18,60 USD	93,00 USD	186,00 USD
Kosten pro Monat	in EUR	1,79 €	17,86 €	89,28 €	178,56 €
SNS					
Kosten pro Monat	in USD	1,09 USD	15,94 USD	164,44 USD	81,94 USD
Kosten pro Monat	in EUR	1,05 €	15,30 €	157,86 €	78,66 €
2 Aurora DB MySQL Knoten	Instanztyp	db.t3 small	db.t3.medium	db.r3.large	db.r3.xlarge
Kosten pro Monat	in USD	123,81 USD	202,14 USD	610,08 USD	1.220,15 USD
Kosten pro Monat	in EUR	118,86 €	194,05 €	585,68 €	1.171,34 €
Totale Kosten für Architektur 3	Spieler pro Tag	100	1.000	5.000	10.000
Kosten pro Monat	in USD	417,30 USD	603,89 USD	1.404,30 USD	2.342,01 USD
Kosten pro Monat	in EUR	400,61 €	579,73 €	1.348,13 €	2.248,33 €



A.2.4 Kosten und Kostenfunktion von A4

API Gateway					
Kosten pro Monat	in USD	1,80 USD	18,00 USD	90,00 USD	180,00 USD
Kosten pro Monat	in EUR	1,73 €	17,28 €	86,40 €	172,80 €
Global Accelerator					
Kosten pro Monat	in USD	18,00 USD	18,00 USD	18,00 USD	18,00 USD
Kosten pro Monat	in EUR	17,28 €	17,28 €	17,28 €	17,28 €
GameLift Instanzen	Instanztyp	c6g.medium	c6g.large	c6g.xlarge	c6g.2xlarge
Kosten pro Monat	in USD	122,21 USD	512,56 USD	2.030,08 USD	3.998,12 USD
Kosten pro Monat	in EUR	117,32 €	492,06 €	1.948,88 €	3.838,20 €
2x Lambda Services					
Kosten pro Monat	in USD	0,10 USD	1,22 USD	9,28 USD	31,08 USD
Kosten pro Monat	in EUR	0,10 €	1,17 €	8,91 €	29,84 €
Dynamo DB (On Demand)					
Kosten pro Monat	in USD	18,94 USD	178,11 USD	946,79 USD	1.893,60 USD
Kosten pro Monat	in EUR	18,18 €	170,99 €	908,92 €	1.817,86 €
Totale Kosten für Architektur 4	Spieler pro Tag	100	1.000	5.000	10.000
Kosten pro Monat	in USD	161,05 USD	727,89 USD	3.094,15 USD	6.120,80 USD
Kosten pro Monat	in EUR	154,61 €	698,77 €	2.970,38 €	5.875,97 €



Eidesstattliche Erklärung

Ich versichere hiermit, dass ich die vorliegende Masterarbeit mit dem Thema: "Analyse und Entwicklung cloudbasierter, asynchroner Multiplayer-Spielarchitekturen mit Amazon Web Services" selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Alle Passagen, die ich wörtlich aus der Literatur oder aus anderen Quellen wie z. B. Internetseiten übernommen habe, habe ich deutlich als Zitat mit Angabe der Quelle kenntlich gemacht. Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

_____ _____

Ort, Datum Unterschrift