DHBW
Duale Hochschule
Baden-Württemberg
CAS
Center for Advanced Studies

Deckblatt wissenschaftliche Arbeit

Transferbericht

Wirtschaft

Titel der wissenschaftlichen Arbeit:

Aufgabenstellung Teil 1 (Sturm) zur Laborarbeit

Modulnummer und -name:	W3M20035.1 Cloud Infrastructures & Cloud Native Applications
Studierende*r:	D. Kleiser, P. Moosmayer, L. Schöneberger, L. Seelbach
Studiengang/-richtung:	Wirtschaftsinformatik
Studienjahrgang:	2022
Dualer Partner:	
Gutachter:	Prof. Dr. Christoph Sturm

Ich versichere, dass ich die vorliegende wissenschaftliche Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Es wurden weder die gesamte Arbeit noch Teile hieraus an anderer Stelle vorgelegt oder veröffentlicht. Darüber hinaus bestätige ich hiermit die inhaltliche Identität zwischen der eingereichten Print- und der elektronischen Version.

Abgabetermin: 10.11.2023

Heilbronn, 01.11.2023

Ort, Datum

Unterschrift

Eingangsvermerk DHBW CAS

Bewertung

Datum: …… Namenszeichen: …… Punkte bzw. Note: ……

Stand: 04.04.2022

Inhaltsverzeichnis

1. Vor- und Nachteile von Cloud-Native-Anwendungen

Cloud Computing ermöglicht den kontinuierlichen und bedarfsgerechten Zugriff auf gemeinsam genutzte Rechenressourcen, ohne direkte manuelle Interaktion (Mell und Grance, 2011, S.3). Heutzutage setzen viele relevante Infrastrukturen Cloud Computing-Modelle wie Infrastructure as a Service, Platform as a Service oder Software as a Service ein, um ihre Ressourcen zu verwalten (Bellini et al., 2015, S.1069). Unternehmen nutzen diese Technologien, um ihre Geschäftsmodelle zu transformieren und Wettbewerbsvorteile zu erzielen (Berman et al., 2011, S.1). Cloud-Anwendungen bieten sowohl Vorteile als auch Herausforderungen im Betrieb (Islam et al., 2013, S.59 f.; Almubaddel und Elmogy, 2016).

Das National Institute of Standards and Technology hebt folgende vorteilhafte Charakteristiken von Cloud-Anwendungen hervor: Automatische Bereitstellung von Rechenkapazitäten, unabhängige Nutzung von Cloud-Anwendungen über das Internet, Ressourcenpooling durch Anbieter, ortsunabhängige Skalierbarkeit, sowie die Kontrolle, Überwachung und Optimierung von Cloudsystemen auf Anbieter- und Kunden-Seite (Mell und Grance, 2011, S. 2).

Andererseits sollten bei der Verwendung von Cloud-Lösungen die folgenden Herausforderungen beachtet werden.

Cloudanwendungen müssen diverse Sicherheitseigenschaften besitzen, da die Anwendungen auf eingeschränkt beeinflussbarer und in gewisser Weise fremder Infrastruktur laufen. Es ist daher unerlässlich, angemessene Sicherheitsmaßnahmen zu ergreifen, um Datenverlust, Phishing-Angriffe und Botnet-Angriffe zu verhindern. (Ramgovind et al., 2010; Almubaddel und Elmogy, 2016; Kuyoro et al, 2011, S.252). Die klare Definition von Service Level Agreements ist entscheidend, um Betriebszeiten, Datenzugriff und Kosten zu regeln, einschließlich potenzieller Erweiterungen und Zusatzleistungen (Almubaddel und Elmogy, 2016; Weinhardt et al., 2009, S.37 f.).

Ein weiteres Problem besteht in der Integration von Cloudanwendungen in bestehende Anwendungsstrukturen und existierende Systeme.. Dabei sollte besonderer Wert auf einen reibungslosen Datentransfer und genauste Definition der Softwareanforderungen, auch von benachbarten Cloud- und On-Premise Systemen gelegt werden (Weinhardt et al., 2009, S.40 f.).

Trotz des Vorteils der Reduzierung der fixen Infrastrukturkosten, muss das Kostenmodell einer Cloudapplikation diskutiert werden. Es gibt verschiedene Preisgestaltungsmodelle, abhängig vom Produktservicetyp und dem

Ontologiekonzept. Es ist wichtig, Aspekte wie flexible Abrechnungsmodelle, multidimensionale Kostenanalysen, Supportmodelle und anwendungsorientierte Optimierungsvereinbarungen in Betracht zu ziehen (Weinhardt et al., 2009, S.40) (Li et al., 2022, S.175).

Menschliche Bedenken, wie Kontrollverlust und Jobverlust bei der Nutzung cloudbasierter Anwendungen, sind ebenfalls relevant. Cloudanbieter können diesen Bedenken mit Tools zur Versions- und Updateverwaltung sowie Backup-Maßnahmen begegnen, die die Transparenz und den Einfluss der Anwender fördern (Almubaddel und Elmogy, 2016).

Die Nutzung und Optimierung der Performance von Cloud-Computing erfordert eine gründliche Untersuchung des Performance Engineerings, abhängig vom Bereitstellungsmodell (IaaS, PaaS, SaaS). Dies beinhaltet Aspekte wie die Verfügbarkeit der Dienste, Datenengpässe, Vorhersagbarkeit und Skalierbarkeit der Performance, Datenschutz und das Risiko des Data-Lock-In gemäß den in den Service Level Agreements festgelegten Bedingungen (Paliwal, 2011, S.7; Almubaddel und Elmogy, 2016).

2. Vor- und Nachteile der Anwendung „URL-Verkürzer“ und alternative Realisierungsmöglichkeiten

Die innerhalb der vorliegenden Gruppenarbeit entwickelte Cloud-Native-Anwendung zur Verkürzung von URLs weist ebenfalls Vor- und Nachteile auf.[1] Es sollten daher auch mögliche Optimierungs- sowie alternative Umsetzungsoptionen in Erwägung gezogen und bewertet werden.

Ein wesentlicher Vorteil dieser Anwendung besteht darin, dass sie mithilfe eines Kubernetes-Netzwerks effizient in eine Vielzahl von Microservices aufgeteilt wird, die miteinander interagieren. Dies ermöglicht eine flexible und skalierbare Architektur. Die Skalierung einzelner Microservices, wie beispielsweise der Flask-Anwendung, wird mithilfe von Ingress durchgeführt. Ingress ermöglicht automatisches Load Balancing, wodurch die Last auf mehrere Instanzen verteilt wird. Dies führt zu einer verbesserten Leistung und Ausfallsicherheit der Anwendung.

[1] Eine detaillierte Anwendungsbeschreibung kann der Datei README.md im Abgabeverzeichnis entnommen werden

Docker-Container machen die Anwendung mobil und unabhängig von der Ausführungsumgebung. Diese Portabilität vereinfacht die Bereitstellung der Anwendung (Deployment) und die Entwicklung auf verschiedenen Plattformen.

Die Entscheidung für Redis als Datenbank ermöglicht eine schnelle Abfrage der Zuordnungen zwischen den verkürzten und Original-URLs, wodurch die Gesamtzuverlässigkeit der Anwendung gesteigert wird. Darüber hinaus bieten Flask als Webframework und Gunicorn als Webserver ein hohes Maß an Flexibilität für die Anpassung und Erweiterung der Anwendung.

Die Einrichtung und Verwaltung eines Kubernetes-Netzwerkes sowie die Konfiguration von Ingress und anderen Ressourcen, bringt auch einige Herausforderungen mit sich. Ein tiefes Verständnis von Konzepten wie der Kubernetes-Orchestrierung ist erforderlich, was die Bereitstellung und Wartung der Anwendung komplexer machen kann. Die Verwendung von Kubernetes und Docker kann auch ressourcenintensiv sein, insbesondere für kleinere Anwendungen ohne erhebliche Skalierungserfordernisse.

Es gibt verschiedene alternative Ansätze, die in Betracht gezogen werden können, um eine URL-Verkürzungsanwendung bereitzustellen bzw. diese weiter zu optimieren. Beispielsweise könnten Skaffold und Helm als nützliche Tools dienen, um den Bereitstellungsprozess zu vereinfachen und die Entwicklungsgeschwindigkeit zu erhöhen. Skaffold übernimmt die automatisierte Erstellung und Bereitstellung von Anwendungen in Kubernetes-Clustern, während Helm die Verwaltung von Kubernetes-Anwendungen mithilfe vordefinierter Pakete erleichtert, um die Komplexität zu reduzieren.

Alternativ zur Verwendung von Kubernetes und Containern könnte auch eine serverlose Architektur in Betracht gezogen werden, bei der der Cloud-Provider die Skalierung und Verwaltung übernimmt, was erneut die Komplexität verringern könnte.

Redis ist eine einfache und dadurch äußerst performante Datenbank. Je nach den spezifischen Anforderungen des Projekts, beispielsweise bei komplexeren Datensätzen oder wenn mehrere Tabellen benötigt werden, könnten alternative Datenbanken wie SQLite, MongoDB, MySQL oder PostgreSQL in Betracht gezogen werden.

Darüber hinaus könnten neben Flask und Gunicorn auch andere Webframeworks und Webserver in Betracht gezogen werden. Zum Beispiel steht mit Django ein voll ausgestattetes Python-Webframework mit umfassenden Funktionen zur

Verfügung, welches für größere Anwendungen deutlich besser geeignet ist. Eine weitere Option wäre Express.js, ein leichtgewichtiges und flexibles JavaScript-Webframework für Node.js. Alternativ könnten auch Ruby on Rails, ASP.NET oder Spring Boot als geeignete Alternativen betrachtet werden.

Zusammenfassend lässt sich feststellen, dass die gewählte Architektur für den URL-Verkürzer geeignet ist. Sie zeichnet sich durch ihre Schlankheit, Wartbarkeit und Skalierbarkeit aus, was insbesondere bei einer zukünftigen Steigerung des Nutzerverhaltens von Vorteil sein kann. Punktuell könnte die Anwendung durch die Nutzung von Automatisierungstools wie Skaffold und Helm noch optimiert werden. Im Falle einer Erweiterung der Anwendung in der Zukunft, wäre eine Neubewertung der Anforderungen erforderlich, um sicherzustellen, dass die Architektur den neuen Bedürfnissen gerecht wird. Hierbei könnten die bereits genannten alternativen Realisierungsmöglichkeiten sinnvoll sein.

3. Gewährleistung von Datensicherheit

Bei der Entwicklung von Cloud-native-Anwendungen ist die Gewährleistung der Datensicherheit von großer Bedeutung. Ein erster Schritt zur Sicherheit besteht in der Überprüfung der erstellten Docker-Container. Im Fall der in dieser Laborarbeit entwickelten URL-Verkürzungsanwendung ist es entscheidend, auf die Verwendung offizieller Docker-Images für Flask und Redis zu achten, um unsichere oder manipulierte Images zu vermeiden. Zudem empfiehlt es sich, unnötige Dienste und Ports zu deaktivieren, um potenzielle Sicherheitslücken zu minimieren.

Die Redis-Instanz ist für die Speicherung der Daten verantwortlich. Um die Datensicherheit zu gewährleisten, können entsprechende Volumes durch Docker Swarm oder Kubernetes verschlüsselt werden. Docker Swarm, ist ein externes Tool, das seit Version 1.12 in die Docker-Engine integriert ist. Es ermöglicht, Benutzername und Passwort für die Verbindung zu Swarm-Clustern zu konfigurieren. Dadurch wird sichergestellt, dass nur autorisierte Benutzer auf die Cluster zugreifen können. Docker Swarm unterstützt auch bei der Erstellung von Redis-Clustern mit Replikation und Failover, indem es ermöglicht, mehrere Redis-Container in einem Swarm-Dienst auszuführen. Dadurch wird sichergestellt, dass die Daten auch dann noch verfügbar bleiben, wenn ein Container oder Host ausfällt (*Docker security*, o. J.; *Swarm mode overview*, o. J.).

Die Implementierung von „Transport Layer Security“ (TLS) zur Verschlüsselung des Datenverkehrs zwischen Redis-Containern und Clients trägt ebenfalls zur Sicherheit bei. Dabei werden, selbstsignierte Zertifikate sowohl für den Redis-Server als auch optional für die Client-Authentifizierung erstellt. Die Zertifikate sollten in den Redis-Containern verfügbar sein, entweder durch Volume Binding oder durch Hinzufügen zum Container-Image. Die Redis-Konfigurationsdatei muss anschließend innerhalb der Container bearbeitet werden, um TLS zu aktivieren und die Zertifikate zu verwenden. Zuletzt ist sicherzustellen, dass die Client-Anwendung die TLS-Verschlüsselung aktiviert und die entsprechenden Zertifikate verwendet, um eine Verbindung mit Redis herzustellen (Jian & Chen, 2017, S. 144; Moravcik & Kontsek, 2020, S. 476f.; *Why and How to Secure Docker Containers Using TLS Certificates* -, 2023).

Zusätzlich ist es sinnvoll, Sicherheitsgruppen zu verwenden, um den Datenverkehr zwischen Containern auf spezifische Regeln zu beschränken. Docker bietet hierfür eine Netzwerkisolation, welche sicherstellt, dass nur berechtigte Container auf die Redis-Instanz zugreifen können. Es kann beispielsweise festgelegt werden, dass ausschließlich die Flask-Anwendung über einen definierten Port auf die Redis-Instanz zugreifen darf. In der „Docker-Compose.yml-Datei“ können mithilfe von Docker Compose Netzwerkregeln definiert werden, um den Datenverkehr zwischen Containern zu steuern. Beispielweise kann ein eigenständiges Netzwerk erstellt werden, um spezifische Netzwerkregeln für den Datenverkehr zwischen der Flask-Anwendung und der Redis-Instanz festzulegen. Obwohl sich diese Container immer noch im gleichen Netzwerk befinden, ermöglicht die Einrichtung eines separaten Netzes eine klarere Strukturierung und Steuerung des Datenverkehrs. In der Anwendung könnte der Zugriff auf den Redis-Port (standardmäßig 6379) von der Flask-Anwendung auf den Redis-Container beschränkt werden. Um unerwünschten Datenverkehr zu verhindern, sollten alle anderen Ports blockiert werden. Eine Aktualisierungsstrategie ist letztendlich hilfreich, um die Anwendungscontainer und die Redis-Instanz auf dem aktuellen Stand zu halten (Abhishek & Rajeswara Rao, 2021, S. 153f.; Jian & Chen, 2017, S. 143f.; Patra et al., 2022, S. 4).

4. Relevanz der DSGVO für die Anwendung

Die Datenschutz-Grundverordnung (DSGVO) ist eine europäische Verordnung, die die Verarbeitung personenbezogener Daten reguliert und darauf abzielt, den Schutz der Privatsphäre von Einzelpersonen innerhalb der Europäischen Union sicherzustellen.

Derzeit verarbeitet die URL-Verkürzungsanwendung keine personenbezogenen Daten, wie etwa Namen, E-Mail-Adressen oder andere identifizierbare Informationen. Die von Benutzern eingegebenen URLs, die in der Redis-Instanz gespeichert werden, sind nicht als personenbezogene Daten zu betrachten, da sie nicht direkt oder indirekt einer natürlichen Person zugeordnet werden können.

Sofern personenbezogene Daten tatsächlich gespeichert werden, erfordert die DSGVO, dass diese nur für den notwendigen Verarbeitungszweck aufbewahrt werden dürfen. Es ist zu gewährleisten, dass personenbezogenen Daten klar und präzise gespeichert und rechtzeitig gelöscht werden. Hierzu sind geeignete technische und organisatorische Maßnahmen erforderlich, um die personenbezogenen Daten vor unbefugtem Zugriff, Verwendung, Offenlegung, Verlust oder Zerstörung zu schützen.

Sollte der betrachtete URL-Verkürzer in Zukunft erweitert werden und personenbezogene Daten verarbeiten, müssten bestimmte Vorkehrungen, wie Verschlüsselung, Zugriffskontrolle, Datensicherheitsmaßnahmen und die Meldung von Datenschutzverletzungen, getroffen werden. Konkret könnten folgende Schritte in Betracht gezogen werden:

- Verwendung von robusten Passwörtern und Authentifizierungsmechanismen für die Redis-Instanz.
- Verschlüsselung der Einträge vor der Speicherung in der Redis-Instanz.
- Regelmäßige Sicherung der Redis-Instanz.

Diese Maßnahmen sollten nur mit ausdrücklicher Zustimmung der Nutzer in Erwägung gezogen werden, die ordnungsgemäß eingeholt und dokumentiert werden muss (Maier et al., 2019, S. 226ff.).

Literaturverzeichnis

Abhishek, M. K., & Rajeswara Rao, D. (2021). Framework to Secure Docker Containers. 2021 Fifth World Conference on Smart Trends in Systems Security and Sustainability (WorldS4), 152–156. https://doi.org/10.1109/WorldS451998.2021.9514041

Bellini, P.; Cenni, D.e; Nesi, P.(2015): A Knowledge Base Driven Solution for Smart Cloud Management. In: 2015 IEEE 8th International Conference on Cloud Computing. 2015 IEEE 8th International Conference on Cloud Computing (CLOUD). New York City, NY, USA, 27.06.2015 - 02.07.2015: IEEE, S. 1069–1072.

Berman, S.; Kesterson-Townes, L.; Marshall, A.; Srivathsa, R. (2011): The power of cloud. Driving business model innovation. Online verfügbar unter https://www.ibm.com/cloud-computing/us/en/assets/power-of-cloud-for-bus-model-innovation.pdf, zuletzt geprüft am 24.10.2023.

Docker security. (o. J.). Docker Documentation. Abgerufen 24. Oktober 2023, von https://docs.docker.com/engine/security/

Islam, A., Irfan, M.; Mohiuddin, K.; Al-Kabashi, H. (2013): Cloud: The Global Transformation. In: 2013 International Conference on Cloud & Ubiquitous Computing & Emerging Technologies. 2013 International Conference on Cloud & Ubiquitous Computing & Emerging Technologies (CUBE). Pune, India, 15.11.2013 - 16.11.2013: IEEE, S. 58–62.

Jian, Z., & Chen, L. (2017). A Defense Method against Docker Escape Attack. Proceedings of the 2017 International Conference on Cryptography, Security and Privacy, 142–146. https://doi.org/10.1145/3058060.3058085

Kuyoro S. O., Ibikunle Fl., Awodele O. (2011): Cloud Computing Security Issues and Challenges (International Journal of Computer Networks (IJCN), Volume (3) : Issue (5) : 2011). Online verfügbar unter https://www.researchgate.net/publication/285011991_Cloud_Computing_Security_Issues_and_Challenges, zuletzt geprüft am 24.10.2023.

Li, F.; Wu, G.; Lu, J.; Jin, M.; An, H.; Lin, J.(2022): SmartCMP: A Cloud Cost Optimization Governance Practice of Smart Cloud Management Platform. In: 2022 IEEE 7th International Conference on Smart Cloud (SmartCloud). 2022 IEEE 7th International Conference on Smart Cloud (SmartCloud). Shanghai, China, 08.10.2022 - 10.10.2022: IEEE, S. 171–176.

Maier, N., Lins, S., Teigeler, H., Roßnagel, A., & Sunyaev, A. (2019). Die Zertifizierung von Cloud-Diensten nach der DSGVO. Datenschutz und Datensicherheit - DuD, 43(4), 225–229. https://doi.org/10.1007/s11623-019-1097-3

Mell, P.; Grance, T. (2011): The NIST Definition of Cloud Computing. Recommendations of the National Institute of Standards and Technology (NIST Special Publication 800-145). Online verfügbar unter http://faculty.winthrop.edu/domanm/csci411/Handouts/NIST.pdf, zuletzt geprüft am 24.10.2023.

Moravcik, M., & Kontsek, M. (2020). Overview of Docker container orchestration tools. 2020 18th International Conference on Emerging ELearning Technologies and Applications (ICETA), 475–480. https://doi.org/10.1109/ICETA51985.2020.9379236

Paliwal, S. (2011): Performance Challenges in Cloud. Online verfügbar unter https://www.cmg.org/wp-content/uploads/2014/03/1-Paliwal-Performance-Challenges-in-Cloud-Computing.pdf.

Patra, M. K., Kumari, A., Sahoo, B., & Turuk, A. K. (2022). Docker Security: Threat Model and Best Practices to Secure a Docker Container. 2022 IEEE 2nd International Symposium on Sustainable Energy, Signal Processing and Cyber Security (ISSSC), 1–6. https://doi.org/10.1109/iSSSC56467.2022.10051481

Ramgovind S.; Eloff MM.; Smith E. (2010): The Management of Security in Cloud Computing. In: Information Security for South Africa (ISSA), 2010.

Swarm mode overview. (o. J.). Docker Documentation. Abgerufen 24. Oktober 2023, von https://docs.docker.com/engine/swarm/

Weinhardt, C.; Anandasivam, A.; Blau, B.; Stosser, J. (2009): Business Models in the Service World. In: IT Prof. 11 (2), S. 28–33. DOI: 10.1109/MITP.2009.21.

Why and How to secure docker containers using TLS certificates -. (2023, Mai 17). https://www.benchworktechnologies.com/blog/why-and-how-to-secure-docker-containers-using-tls-certificates/