Entwicklung einer Cloud Native App

Laborabeit in Cloud Infrastructures and Cloud Native Applications

Im Master Studiengang Informatik an der

DHBW-CAS

von

Linus Baumann, Alexander Dixon, Cosima Dotzauer, Dennis Hilgert, Jonas Werner

28. September 2025

Matrikelnummern	xxxxxxx, xxxxxxx, 3750882, xxxxxxx, xxxxxxx
Kurs	W3M20035
Semester	SoSe 2025
Begutachter	Prof. Dr. Christoph Sturm

Inhaltsverzeichnis

1 Vorstellung der Cloud-Native-Anwendung

Die in der Laborarbeit entwickelte Cloud-Native-Anwendung hat das Ziel, Nutzern wie Energieunternehmen oder Grundstückseigentümern bei der Suche nach geeigneten Standorten für erneuerbare Energien zu unterstützen. Hierfür werden sowohl historische als auch aktuelle Wetterdaten sowie geographische Gegebenheiten analysiert, um potenzielle Standorte für Wind- oder Solaranlagen zu identifizieren und umfassend zu bewerten.

Die Bewertung dieser Standorte erfolgt anhand verschiedener Kriterien. Neben geeigneten Wetterbedingungen wird auch die Entfernung des Standorts zu bestehenden Stromleitungen berücksichtigt, da dies entscheidend für die wirtschaftliche Rentabilität einer möglichen Anlage ist. Zudem wird geprüft, ob sich der Standort in einem Naturschutzgebiet, im Wald oder in einem Wohngebiet befindet, da diese Faktoren Einfluss auf die Genehmigungschancen für den Bau einer Anlage haben.

Die Anwendung bietet den Nutzern eine intuitive Benutzeroberfläche zur Visualisierung dieser Informationen, die bei der Entscheidungsfindung unterstützt und somit zur Planung von Wind- und Solaranlagen beiträgt.

1.1 Architektur der Cloud-Native-Anwendung

Die Architektur der Anwendung ist in zwei Hauptservices unterteilt:

Weather-Microservice: Dieser Service ist verantwortlich für die Verarbeitung und Bereitstellung von Wetterdaten. Er bezieht seine Informationen über die OpenMeteo-APIs, die sowohl historische als auch aktuelle Wetterbedingungen bereitstellen. Die Wetterdaten werden in einer PostgreSQL-Datenbank gespeichert, um eine effiziente Abfrage und Analyse zu ermöglichen. Zudem beinhaltet der Weather-Microservice ein Modell zur Wettervorhersage, das auf historischen Wetterdaten basiert und Prognosen für zukünftige Wetterbedingungen erstellt.

Geo-Microservice: Der Geo-Microservice liefert Informationen zur vorherrschenden Infrastruktur, einschließlich der Überprüfung der Entfernung zu bestehenden Stromleitungen und der Lage in Bezug auf Naturschutzgebiete, Wald oder Wohngebiete. Die Daten für diesen Service werden aus der Overpass-API von OpenStreetMap bezogen.

Beide Microservices sind über REST-APIs erreichbar und in Containern isoliert, was eine flexible und skalierbare Architektur gewährleistet.

Zur Orchestrierung der Container wird Kubernetes eingesetzt, das die Automatisierung von Deployments, die Skalierung und das Management der Container ermöglicht. Die Anwendung verwendet das

App-of-Apps-Muster in Kubernetes, um eine hierarchische Struktur von Anwendungen zu schaffen, die eine effiziente Verwaltung der Microservices und ihrer Abhängigkeiten ermöglicht.

1.2 Automatisierung des Entwicklungsprozesses

Für die Automatisierung der Infrastruktur- und Anwendungsbereitstellung kommt Ansible zum Einsatz. Ansible ermöglicht es, wiederholbare und skalierbare Deployment-Prozesse zu erstellen, die die Konfiguration und den Betrieb des Kubernetes-Clusters sowie der Container orchestrieren. Zusätzlich wird ein CI/CD-Workflow implementiert, der die kontinuierliche Integration und Bereitstellung der Anwendung ermöglicht. Dies umfasst die automatisierte Erstellung von Docker-Images, das Testen der Anwendung und die Bereitstellung in der Produktionsumgebung. GitHub Actions werden verwendet, um den Build- und Deployment-Prozess zu automatisieren.

2 Vorteile und Nachteile der Cloud-Native-Realisation

In dem folgenden Kapitel werden die Vorteile und Nachteile der hier vorliegenden Cloud-Native-Realisierung im Vergleich zu anderen Ansätzen diskutiert sowie mögliche alternative Realisierungsmöglichkeiten aufgezeigt.

2.1 Vorteile und Nachteile der Cloud-Native-Realisation

Die Realisierung der Anwendung zur Unterstützung der Standortfindung für erneuerbare Energien als Cloud-Native-Anwendung bietet mehrere Vorteile:

Skalierbarkeit und Elastizität: Bei steigender Nachfrage, beispielsweise wenn viele Nutzer gleichzeitig potenzielle Standorte für Wind- oder Solaranlagen abfragen, können zusätzliche Instanzen der Microservices schnell bereitgestellt werden, um die Last zu bewältigen. Dies wird durch die Containerisierung und Orchestrierung mit Kubernetes ermöglicht.

Unabhängige Entwicklung von Microservices: Die Nutzung von Microservices erlaubt es verschiedenen Teams, parallel an spezifischen Komponenten der Anwendung zu arbeiten. So kann ein Team beispielsweise an der Wetterdatenverarbeitung arbeiten, während ein anderes Team gleichzeitig den Geo-Microservice weiterentwickelt. Dies beschleunigt die Einführung neuer Funktionen, wie die Integration zusätzlicher Wetterdatenquellen, ohne dass dies die gesamte Anwendung beeinträchtigt.

Resilienz und Verfügbarkeit: Durch den Einsatz von Kubernetes-Primitiven wie Liveness- und Readiness-Probes wird sichergestellt, dass fehlerhafte Instanzen automatisch erkannt und neu gestartet werden. Gibt es zum Beispiel Fehler in der Wetterdatenabfrage, führen diese nicht zu einem Ausfall der gesamten Anwendung. Dies ist besonders wichtig, um die Verfügbarkeit der Anwendung für die Standortbewertung sicherzustellen.

Automatisierung in der Bereitstellung: Die Implementierung von CI/CD-Pipelines zur Automatisierung des Entwicklungs- und Bereitstellungsprozesses reduziert menschliche Fehler und beschleunigt die Zeit von der Entwicklung bis zur Produktion. Dies ist entscheidend, um zeitnah auf Änderungen in den regulatorischen Anforderungen für erneuerbare Energien zu reagieren oder neue Datenquellen schnell zu integrieren. Tools wie Ansible erleichtern zudem das Infrastrukturmanagement, was die Effizienz der gesamten Anwendung steigert.

Trotz dieser Vorteile gibt es auch einige **Nachteile.**

Herausforderungen bei der Netzwerkkommunikation: Die Kommunikation zwischen den Microservices erfolgt über Netzwerke, was potenzielle Latenzen und Sicherheitsrisiken mit sich bringt. Wenn beispielsweise der Geo-Microservice aufgrund von Netzwerkproblemen nicht erreichbar ist, kann dies die gesamte Anwendung beeinträchtigen und den Zugriff auf wichtige Informationen für die Standortbewertung verzögern. Dies gilt ebenso für den Zugriff auf externe APIs. Wenn eine externe API aufgrund von Netzwerkproblemen oder Serverausfällen nicht verfügbar ist, beeinträchtigt dies die Anwendung erheblich.

Erhöhter Overhead durch Containerisierung: Jeder Microservice läuft in einem eigenen Container, was zusätzliche Ressourcen benötigt und die Infrastrukturverwaltung komplizierter gestaltet.

Herausforderungen bei der Netzwerkkommunikation: Die Sicherheit und Effizienz der Datenübertragung zwischen dem Weather-Microservice und der PostgreSQL-Datenbank können Schwierigkeiten bereiten. Hier muss auf den Punkt Zugriffskontrolle geachtet werden.

2.2 Alternative Realisierungsmöglichkeiten

Alternativen zur Cloud-Native-Architektur sind die monolithische Architektur und traditionelle serverbasierte Anwendungen. Jede dieser Alternativen bietet spezifische Vor- und Nachteile, die im Kontext der entwickelten Anwendung zur Beratung für nachhaltige Energiegewinnungsstandorte berücksichtigt werden sollten.

2.2.1 Monolithische Architektur

Eine monolithische Architektur integriert alle Funktionen der Anwendung in einer einzigen, großen Codebasis. Dies kann die Komplexität reduzieren und die Verwaltung erleichtern, da alle Komponenten als eine Einheit entwickelt, getestet und bereitgestellt werden.

Für die vorliegende Anwendung würde dies bedeuten, dass sowohl die Verarbeitung der Wetterdaten als auch die geospatialen Analysen in einer einzigen Anwendung zusammengefasst wären. Dies könnte von Vorteil sein, da so alle Informationen zur Bewertung eines Standortes auf einmal gesammelt werden und nicht später zusammengeführt werden müssen.

Allerdings bringt eine monolithische Architektur auch erhebliche Nachteile mit sich. Die Flexibilität und Skalierbarkeit würden stark eingeschränkt, da die gesamte Anwendung als Einheit skaliert werden müsste. Bei einem Anstieg der Nutzerzahlen, beispielsweise wenn viele Grundstückseigentümer gleichzeitig potenzielle Standorte abfragen, wäre es nicht möglich, nur die wetterbezogenen Funktionen zu skalieren. Dies könnte zu Performance-Problemen führen und die Reaktionsfähigkeit der Anwendung beeinträchtigen.

2.2.2 Traditionelle serverbasierte Anwendungen

Traditionelle serverbasierte Anwendungen bieten ebenfalls eine einfachere Verwaltung, da sie in der Regel auf einer einzigen Serverinstanz oder einer Gruppe von Servern betrieben werden. Diese Art der Architektur könnte die Komplexität der Bereitstellung und des Betriebs verringern, insbesondere in kleinen Teams mit begrenzten Ressourcen.

Im Kontext der Anwendung zur Beratung für nachhaltige Energiegewinnungsstandorte könnte eine traditionelle serverbasierte Anwendung alle Funktionen auf einem zentralen Server bündeln. Dies könnte den initialen Aufwand für das Hosting und die Verwaltung der Anwendung verringern. Beispielsweise könnte die Anwendung alle Wetter- und Geoinformationen direkt von einem Server abrufen, ohne dass eine komplexe Container- oder Microservices-Architektur erforderlich wäre.

Jedoch würde eine solche Architektur nicht die gleiche Agilität bieten wie Cloud-Native-Lösungen. Änderungen und Updates an einzelnen Komponenten der Anwendung könnten schwierig und zeitaufwändig sein, da jede Änderung die gesamte Anwendung betreffen würde. Zudem wäre die Skalierung in einer traditionellen serverbasierten Umgebung möglicherweise nicht so effizient, insbesondere wenn die Anwendung wachsen und viele Nutzer gleichzeitig bedienen muss.

3 Cloud-Native Patterns in der Anwendungsarchitektur

In der entwickelten Cloud-Native-Anwendung zur Beratung für nachhaltige Energiegewinnungsstandorte finden sich verschiedene Cloud-Native Patterns, die sowohl aus dem Bereich Development Design als auch Infrastructure Cloud und Operations stammen. Diese Patterns tragen zur Effizienz, Skalierbarkeit und Wartbarkeit der Anwendung bei.

3.0.1 1. Microservices-Architektur

Ein zentrales Cloud-Native Pattern ist die **Microservices-Architektur**. In dieser Architektur wird die Anwendung in mehrere unabhängige, spezialisierte Services unterteilt, die jeweils eine bestimmte Funktion erfüllen. In der vorliegenden Anwendung gibt es beispielsweise den **Weather-Microservice**, der sich auf die Verarbeitung und Bereitstellung von Wetterdaten konzentriert, sowie den **Geo-Microservice**, der Informationen zur geographischen Infrastruktur bereitstellt.

Begründung des Einsatzes: Die Microservices-Architektur bietet mehrere Vorteile für die Anwendung. Erstens ermöglicht sie eine unabhängige Entwicklung und Bereitstellung der Services. Teams können an verschiedenen Komponenten arbeiten, ohne sich gegenseitig zu behindern, was die Entwicklungsgeschwindigkeit erhöht. Zweitens kann jeder Microservice unabhängig skaliert werden, was besonders wichtig ist, wenn beispielsweise die Abfrage von Wetterdaten während bestimmter Ereignisse (z.B. starker Wind oder extreme Wetterbedingungen) zunimmt. Diese Flexibilität ist entscheidend, um den Anforderungen der Nutzer gerecht zu werden und eine hohe Verfügbarkeit zu gewährleisten.

3.0.2 2. Containerisierung

Ein weiteres wichtiges Cloud-Native Pattern ist die **Containerisierung**. Durch die Verwendung von Containern, die in Kubernetes orchestriert werden, wird die Anwendung in isolierte Umgebungen verpackt, die alle benötigten Abhängigkeiten enthalten.

Begründung des Einsatzes: Die Containerisierung bietet zahlreiche Vorteile. Sie ermöglicht eine konsistente Ausführung der Anwendung, unabhängig von der zugrunde liegenden Infrastruktur, was besonders vorteilhaft ist, wenn die Anwendung in verschiedenen Umgebungen (z.B. Entwicklung, Test, Produktion) betrieben wird. Darüber hinaus erleichtert die Containerisierung die Skalierung und das Management der Anwendung, da Container bei Bedarf schnell gestartet oder gestoppt werden können. Diese Effizienz ist besonders wichtig, um die Ressourcen optimal zu nutzen und die Betriebskosten

zu minimieren, insbesondere in einem Bereich, in dem Kosteneffizienz und Nachhaltigkeit von großer Bedeutung sind.

3.0.3 Alternative Patterns

Zusätzlich zu den oben genannten Patterns könnten auch folgende Cloud-Native Patterns in Betracht gezogen werden:

3.0.4 3. Event-Driven Architecture

Die **Event-Driven Architecture** könnte als Alternative implementiert werden, um die Kommunikation zwischen den Microservices zu optimieren. In einer solchen Architektur reagieren Microservices auf Ereignisse und kommunizieren über ein Messaging-System, anstatt direkt über REST-APIs.

Begründung des Einsatzes: Diese Architektur könnte die Entkopplung der Microservices weiter verbessern und die Reaktionsfähigkeit der Anwendung erhöhen. Beispielsweise könnten Wetterdaten als Ereignisse veröffentlicht werden, auf die andere Microservices reagieren, um Standortanalysen oder Benachrichtigungen zu aktualisieren. Dies würde die Gesamtleistung der Anwendung steigern und die Komplexität der direkten API-Interaktionen verringern.

3.0.5 4. Service Mesh

Ein weiteres relevantes Pattern ist die Implementierung eines **Service Mesh**, das eine verbesserte Verwaltung der Microservices-Kommunikation ermöglicht.

Begründung des Einsatzes: Ein Service Mesh bietet Funktionen wie Traffic Management, Sicherheit und Monitoring auf einer abstrahierten Ebene, was die Verwaltung der Microservices vereinfacht. Dadurch könnte die Anwendung robuster und sicherer werden, da Sicherheitsrichtlinien einfacher implementiert werden können, ohne dass Änderungen an den einzelnen Microservices erforderlich sind. Dies wäre besonders wichtig in einem Anwendungsbereich, der mit sensiblen Daten arbeitet, wie etwa den Standortinformationen für erneuerbare Energien.

Insgesamt tragen die gewählten Cloud-Native Patterns dazu bei, die Anwendung effizient, flexibel und wartbar zu gestalten, während alternative Patterns zusätzliche Vorteile bieten könnten, die je nach zukünftigen Anforderungen und Entwicklungen in der Anwendung in Betracht gezogen werden sollten.

4 Datensicherheit und Datenschutz in Cloud-Native-Anwendungen