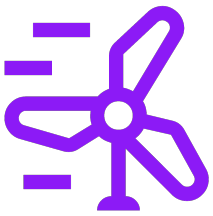

Entwicklung der Cloud Native App Green Grid Guide

Wind & Solar Potential Explorer

Laborarbeit 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

Abbildungsverzeichnis

1 Vorstellung der Cloud-Native-Anwendung

Die vorliegende Cloud-Native-Anwendung *Green Grid Guide* unterstützt Energieunternehmen und Grundstückseigentümer bei der Standortsuche für erneuerbare Energien. Sie analysiert historische und aktuelle Wetterdaten sowie geografische Gegebenheiten, um Standorte für Wind- und Solaranlagen zu bewerten. Kriterien wie Wetterbedingungen, Entfernung zu Stromleitungen und Lage in Naturschutz-, Wald- oder Wohngebieten werden hierbei berücksichtigt, da sie die Rentabilität und Genehmigungschancen beeinflussen. Eine interaktive Karte mit anpassbarem Suchradius hilft Nutzern bei der Entscheidungsfindung und Planung von Anlagen.

2 Vor- und Nachteile der Cloud-Native-Anwendung

Die Cloud-Native-Architektur der Anwendung bietet mehrere Vorteile:

Skalierbarkeit: 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 die Orchestrierung mit Kubernetes ermöglicht. Unbenötigte Instanzen können bei geringerer Nachfrage effizient heruntergefahren werden, was Ressourcen spart.

Unabhängige Entwicklung von Microservices: Die Nutzung von Microservices ermöglicht es verschiedenen Teammitgliedern, parallel an spezifischen Komponenten der Anwendung zu arbeiten. So kann ein Teammitglied beispielsweise an der Verarbeitung von Wetterdaten arbeiten, während ein anderes Teammitglied gleichzeitig den Geo-Microservice weiterentwickelt. Dies beschleunigt die Einführung neuer Funktionen, wie die Integration zusätzlicher Wetterdatenquellen, ohne die gesamte Anwendung zu beeinträchtigen.

Resilienz und Verfügbarkeit: Der Einsatz von Kubernetes-Primitiven wie Liveness- und Readiness-Probes gewährleistet, dass fehlerhafte Instanzen automatisch erkannt und neu gestartet werden. Wenn beispielsweise Fehler in der Wetterdatenabfrage in der Datenbank auftreten, führt dies nicht zu einem Ausfall der gesamten Anwendung.

Automatisierung in der Bereitstellung und einfacher Betrieb: 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 und neue Datenquellen schnell zu integrieren. Hier eingesetzte Tools wie Ansible erleichtern das Infrastrukturmanagement und steigern die Effizienz der Anwendung.

Trotz dieser Vorteile gibt es auch einige Nachteile:

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

Herausforderungen bei der Netzwerkkommunikation: Die Sicherheit und Effizienz der Datenübertragung zwischen dem Weather-Microservice und der PostgreSQL-Datenbank können Schwierigkeiten bereiten. Auch die Kommunikation des Frontends mit den Microservices über ein Netzwerk kann potenzielle Latenzen und eine komplexere Architektur mit sich bringen. 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.

3 Alternative Realisierungsmöglichkeiten

Alternative Realisierungsmöglichkeiten sind die monolithische Architektur und On-Premise-Lösungen, die jeweils spezifische Vor- und Nachteile für die vorliegende Anwendung bieten.

Eine monolithische Architektur fasst im Gegensatz zu Microservices alle Funktionen in einer einzigen Codebasis zusammen, was die Komplexität und Verwaltung erleichtert, da alles als Einheit entwickelt und bereitgestellt wird. Für die *Green Grid Guide* Anwendung bedeutet dies, dass die Wetterdatenverarbeitung und geographischen Analysen zentralisiert sind, was Netzwerkprobleme reduziert und die Komplexität verringert, da die einzelnen Informationen nicht von verschiedenen Microservices abgerufen werden müssen.
Allerdings schränkt die monolithische Architektur die Skalierbarkeit ein, da die gesamte Anwendung als Einheit skaliert werden muss. Zudem werden Entwicklung und Wartung erschwert, da unabhängige Arbeiten an spezifischen Services nicht möglich sind und Änderungen unbeabsichtigte Auswirkungen haben können.

On-Premise-Lösungen bieten der *Green Grid Guide* Anwendung Vorteile in Bezug auf Datensicherheit und Kontrolle über die Infrastruktur. Durch die Bündelung aller Funktionen auf einem zentralen Server wird die Sicherheit der Daten erhöht und Risiken, die mit externen Cloud-Diensten verbunden sind, minimiert. Diese Aspekte sind besonders wichtig, wenn sensible Informationen verarbeitet werden, was in der aktuellen Anwendung jedoch nicht der Fall ist. Zukünftige Erweiterungen könnten ein Pricing-Modell für Regionen außerhalb Deutschlands mit Zahlungsdaten und Benutzerkonten erfordern, wo diese Vorteile relevant wären.

Der Nachteil von On-Premise-Lösungen liegt in der Notwendigkeit, eigene Hardware anzuschaffen, was mit hohen Kosten verbunden ist und zudem die Skalierbarkeit einschränkt. Die Hardware muss gewartet werden, was zusätzliche Ressourcen beansprucht. Für unsere Anwendung, die derzeit keine sensiblen Daten verarbeitet, ist eine Cloud-Lösung daher sinnvoller. Sie bietet Skalierbarkeit, ist kosteneffizienter und vereinfacht die Wartung erheblich, da sie keinen physischen Hardwarebedarf hat.

4 Verwendete Cloud-Native Patterns

Die *Green Grid Guide* Anwendung nutzt verschiedene Cloud-Native Patterns, um effiziente und skalierbare Lösungen zu bieten. Zwei wesentliche Patterns aus dem Bereich „Development & Design“, die hier verwendet wurden, sind die „Microservices Architecture“ und die damit oft verbundene „Communicate Through APIs“.
Die Anwendung ist in Microservices aufgeteilt, die unabhängig entwickelt und bereitgestellt werden. Dies fördert die parallele Arbeit der Teammitglieder und ermöglicht eine modulare Entwicklung. Jeder Microservice kann hierbei die für ihn optimalen Technologien nutzen. Der Weather-Microservice verwendet beispielsweise eine PostgreSQL-Datenbank zur Speicherung von Wetterdaten. Sollte der Geo-Microservice in zukünftigen Erweiterungen ebenfalls eine Datenbank benötigen, könnte er eine PostgreSQL-Datenbank mit PostGIS-Erweiterung verwenden, die speziell für räumliche Datentypen gedacht ist und eine einfachere Suche nach Orten im Suchradius ermöglicht. Beide Microservices werden vom Frontendservice über REST-APIs angesprochen, um die Informationen den Nutzern bereitzustellen, was die Flexibilität und Wartbarkeit der Anwendung erhöht. Wenn beispielsweise der Geo-Microservice überarbeitet werden muss, bleibt das Frontend unberührt, solange die API-Schnittstelle beibehalten wird.

Aus dem Bereich „Infrastructure & Cloud“ wurden die eng miteinander verknüpften Patterns „Continuous Delivery“ und „Continuous Deployment“ eingesetzt.
Diese ermöglichen einen schnellen Entwicklungszyklus, indem Änderungen an den Services durch CI/CD-Pipelines automatisiert getestet und bereitgestellt werden. GitHub Actions führt diese Pipelines aus, überprüft den Code und erstellt Docker-Images, die in einer Container-Registry gespeichert werden. So bleibt die Anwendung stets aktuell. Dieser Workflow wird in Abbildung 4.1 graphisch dargestellt.

Abbildung 4.1: CI/CD-Pipeline mit GitHub Actions [Eigene Abbildung]

Die weitere Umsetzung des Continuous Deployment und Delivery Patterns ist in Abbildung 4.2 dargestellt.

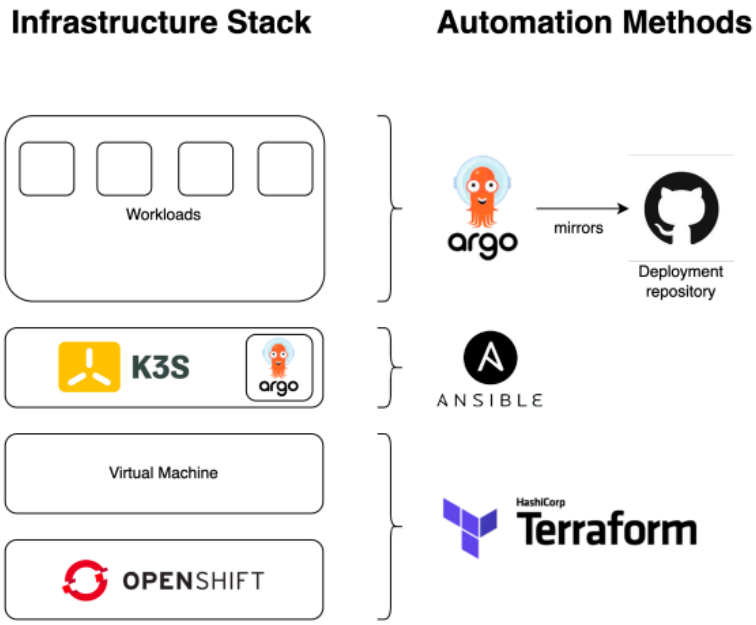

Abbildung 4.2: Infrastruktur der Anwendung und zugehörige Automatisierungsmethoden [Eigene Abbildung]

Auf der obersten Ebene befinden sich die Workloads, wie das Frontend und die Microservices. Änderungen, die durch die CI-Pipeline getestet wurden, werden in ein Deployment Repository auf GitHub hochgeladen, das von Argo CD überwacht wird. Darunter liegen K3S, eine leichte Kubernetes-Distribution, und Ansible. K3S dient als Laufzeitumgebung für die Container, während Argo CD Änderungen automatisch auf das K3S-Cluster anwendet. Ansible konfiguriert und provisioniert die Virtual Machines. Terraform wird zur Bereitstellung der Virtual Machines genutzt und verwaltet die Infrastruktur als Code, was eine konsistente und reproduzierbare Umgebung gewährleistet.
Zusammengefasst ist die gesamte Pipeline, beginnend mit der Entwicklung, über die automatisierten Tests, bis hin zur Möglichkeit, die Anwendung und Infrastruktur in eine automatisiert zu deployen, auf Continuous Deployment und Continuous Delivery ausgelegt. Dadurch wird die Qualität der Anwendung sichergestellt, da Änderungen kontinuierlich getestet werden, und eine schnelle Reaktion auf Feedback von Nutzern sowie auf sich ändernde Anforderungen ermöglicht werden.

5 Datensicherheit und Datenschutz

Derzeit spielt die Datenschutz-Grundverordnung (DSGVO) in der *Green Grid Guide* Anwendung eine geringe Rolle, da keine personenbezogenen Daten verarbeitet werden und Nutzer keinen Account für die Standortsuche in Deutschland benötigen. Daher sind bisher keine speziellen Datenschutzmechanismen implementiert. Zukünftige Erweiterungen könnten jedoch ein Pricing-Modell für internationale Standorte erfordern, das die Verarbeitung personenbezogener Daten wie Name und Zahlungsinformationen sowie die Einwilligung der Nutzer zur Datenverarbeitung umfasst. Die Sicherheit dieser Daten muss während ihrer gesamten Lebensdauer, von Erzeugung über Übertragung, Nutzung, Speicherung bis hin zur Archivierung und Löschung, gewährleistet sein. Entsprechende Sicherheitsmaßnahmen müssen sowohl auf Netzwerk- als auch auf Anwendungsebene implementiert werden, wobei der Standort der Server, auf denen die Nutzerdaten gespeichert werden, von besonderer Bedeutung ist.

Literaturverzeichnis

[1] *Pattern: Microservices Architecture*. https://www.cnpatterns.org/development-design/microservices-architecture. Accessed: 2024-09-19. URL: https://www.cnpatterns.org/development-design/microservices-architecture.

[2] *Pattern: Communicate Through APIs*. https://www.cnpatterns.org/development-design/communicate-through-apis. Accessed: 2025-09-19.

[3] *Pattern: Continuous Delivery*. https://www.cnpatterns.org/infrastructure-cloud/continuous-delivery. Accessed: 2024-09-19. URL: https://www.cnpatterns.org/infrastructure-cloud/continuous-delivery.

[4] *Pattern: Continuous Deployment*. https://www.cnpatterns.org/infrastructure-cloud/continuous-deployment. Accessed: 2024-09-19. URL: https://www.cnpatterns.org/infrastructure-cloud/continuous-deployment.