Projektantrag

Thema der Projektarbeit

Aufbau einer self-hosted Deployment-Plattform für Webanwendungen
als Alternative zu externen PaaS-Diensten

Ausbildungsberuf

Fachinformatiker Systemintegration

Auszubildender: Andreas Kaiser
Prüflingsnummer: *[PRÜFLINGSNUMMER]*
E-Mail: *[E-MAIL]*

Ausbildungsbetrieb:
AKARA Solutions GmbH
Feldstraße 97
25421 Pinneberg

Projektbetreuer: Eike-Christian Ramcke
E-Mail: info@akara-solutions.de

Geplanter Bearbeitungszeitraum

[DATUM VON] – *[DATUM BIS]*

1. Projektbezeichnung

Aufbau einer self-hosted CI/CD-Deployment-Plattform auf Basis von Hetzner Cloud, Docker und Coolify zur automatisierten Bereitstellung von Webanwendungen bei der AKARA Solutions GmbH.

2. Projektumfeld

Die AKARA Solutions GmbH ist ein IT-Dienstleistungsunternehmen mit Sitz in Pinneberg, das für eigene Produkte und Kundenprojekte mehrere Webanwendungen betreibt und kontinuierlich neue Anwendungen veröffentlicht. Bisher erfolgt das Hosting dieser Anwendungen über externe Platform-as-a-Service-Anbieter (PaaS) wie Netlify oder Vercel.

Diese Dienste sind für Einzelprojekte praktisch, stoßen jedoch bei wachsendem Anwendungsportfolio an Grenzen: steigende monatliche Kosten, eingeschränkte Konfigurierbarkeit, Abhängigkeit von Drittanbietern sowie Einschränkungen bei serverseitigen Verbindungen (z. B. SSE, Webhooks, Hintergrundprozesse).

Das Unternehmen benötigt eine eigene, kontrollierbare Infrastruktur, die beliebig viele Webanwendungen zu planbaren, günstigen Kosten betreiben kann.

3. Projektbeschreibung

3.1 Ausgangssituation (Ist-Zustand)

Aktuell werden Webanwendungen der AKARA Solutions GmbH über externe PaaS-Dienste bereitgestellt. Dabei treten regelmäßig folgende Probleme auf:

- Steigende und schwer planbare Kosten bei zunehmendem Anwendungsportfolio
- Keine vollständige Kontrolle über Server, Laufzeitumgebung und Konfiguration
- Einschränkungen bei langlebigen HTTP-Verbindungen (SSE, WebSockets) durch Edge-Function-Limits
- Vendor Lock-in: Deployments sind plattformabhängig konfiguriert
- Kein einheitlicher Deployment-Prozess über alle Projekte hinweg
- Fehlende Möglichkeit, Zugriffsschutz (z. B. Basic Auth für Staging-Umgebungen) flexibel einzurichten

3.2 Zielsetzung (Soll-Zustand)

Ziel des Projekts ist der Aufbau einer vollständig self-hosted Deployment-Plattform, die folgende Anforderungen erfüllt:

- Zentrales Hosting beliebig vieler Webanwendungen auf einem eigenen Server
- Automatisches SSL-Zertifikatsmanagement über Let’s Encrypt
- CI/CD-Integration: Push auf den Hauptbranch eines Git-Repositories löst automatisches Re-Deployment aus
- Wildcard-DNS-Konfiguration: jede neue Subdomain ist ohne zusätzliche DNS-Änderungen sofort erreichbar
- Zugriffsschutz (Basic Auth) für Staging-Umgebungen konfigurierbar
- Sicherheitskonzept: Firewall, SSH-Härtung, rollenbasierter Zugriff
- Backup-Konzept für Plattformkonfiguration und Anwendungsdaten

Konkrete messbare Ziele:

- Deployment-Zeit pro Anwendung von manuell >2 Stunden auf unter 15 Minuten reduzieren
- Mindestens 3 Webanwendungen erfolgreich auf der Plattform deployen und im Testbetrieb dokumentieren
- Monatliche Infrastrukturkosten gegenüber vergleichbaren PaaS-Tarifen um mindestens 60 % senken
- Vollständige technische Dokumentation inkl. Betriebshandbuch erstellen

3.3 Projektumfang und eigene Aufgaben

Im Rahmen der Projektarbeit übernehme ich folgende Aufgaben:

- Analyse der bestehenden Deployment-Situation und Aufnahme der Anforderungen
- Wirtschaftlichkeitsvergleich: Hetzner Cloud + Coolify vs. externe PaaS-Dienste
- Auswahl und Begründung der eingesetzten Komponenten (Server, Betriebssystem, Container-Plattform, Reverse Proxy)
- Provisionierung des Servers (Hetzner Cloud, Ubuntu 24.04)
- Konfiguration von DNS, Wildcard-Records und Subdomains
- Installation und Konfiguration von Docker, Coolify und Traefik inkl. automatischem SSL
- Einrichtung der CI/CD-Pipeline (GitHub Webhook → automatisches Re-Deployment)
- Deployment und Dokumentation von mindestens 3 Webanwendungen als Proof of Concept
- Konfiguration des Sicherheitskonzepts (Firewall, SSH-Härtung, Zugriffsschutz)
- Erstellung eines Backup-Konzepts
- Durchführung von Funktionstests und Dokumentation der Ergebnisse

3.4 Abgrenzung

Nicht Bestandteil des Projekts sind:

- Entwicklung der deploybaren Webanwendungen selbst
- Aufbau einer Hochverfügbarkeits-Infrastruktur (mehrere Server, Load Balancer)
- Einrichtung eines vollständigen Monitoring-Stacks (z. B. Prometheus/Grafana)
- Langfristiger Betrieb und Support der Plattform nach Projektabschluss

4. Ziele des Projektes im Detail

Technisch:

- Aufbau eines produktionsfähigen Servers (Hetzner CAX21, ARM64, Ubuntu 24.04)
- Containerbasierter Betrieb aller Anwendungen via Docker
- Reverse Proxy (Traefik) mit automatischem SSL über Let’s Encrypt
- Wildcard-DNS auf eigener Domain, sodass neue Subdomains ohne DNS-Eingriff sofort erreichbar sind
- Webhook-basiertes automatisches Re-Deployment bei Git-Push
- Firewall-Konfiguration: nur Ports 22, 80 und 443 öffentlich erreichbar

- SSH-Härtung: schlüsselbasierte Authentifizierung, Root-Login abgesichert
- Backup-Konzept: Hetzner Server-Snapshots, Sicherung der Coolify-Konfigurationspfade

Organisatorisch:

- Standardisierung des Deployment-Prozesses für alle zukünftigen Projekte
- Vollständige Dokumentation der Infrastruktur als Betriebshandbuch
- Nachvollziehbare Wirtschaftlichkeitsbetrachtung als Entscheidungsgrundlage

Datenschutz und Sicherheit:

- Keine Speicherung nicht erforderlicher Daten auf dem Server
- Zugriffsschutz für unveröffentlichte Anwendungen (Basic Auth)
- Verschlüsselte Kommunikation aller Anwendungen (HTTPS, Let's Encrypt)
- Rollenbasierter Zugriff auf die Verwaltungsoberfläche

5. Vorgehensmodell

Es wird das Wasserfallmodell angewendet, da die Aufgaben in klar voneinander abgrenzbare, sequenzielle Phasen unterteilt werden können. Jede Phase baut auf den Ergebnissen der vorherigen auf; die lineare Struktur ermöglicht ein systematisches Vorgehen mit fest definierten Meilensteinen.

Phasen: Analyse → Planung → Umsetzung → Test und Qualitätssicherung → Dokumentation und Abschluss

6. Projektplanung (Zeitaufwand: 40 Stunden)

Projektphase	Inhalte	Dauer (h)
Startphase	Ist-Analyse, Anforderungsaufnahme, Wirtschaftlichkeitsvergleich (PaaS vs. self-hosted)	4
Planungsphase	Komponentenauswahl und Begründung, Architekturdesign, Sicherheits- und Backup-Konzept, Ressourcenplanung	6
Durchführung (technisch)	Server-Provisionierung, DNS-Konfiguration, Docker/Coolify/Traefik-Setup, SSL, CI/CD-Integration, Deployment von 3 Anwendungen, Firewall, SSH-Härtung	20
Abschlussphase	Funktionstests, Sicherheitstest (Port-Scan, Header-Check), Soll-Ist-Vergleich, Nachkalkulation	6
Dokumentation & Reflexion	Betriebshandbuch, Installationsanleitung, persönliche Reflexion	4
Gesamtaufwand		**40**

6.1 Zeitliche Einordnung (Kalenderwochen)

- **KW *[X]*:** Startphase – Ist-Analyse, Anforderungen, Wirtschaftlichkeitsvergleich (4 h)

- **KW *[X+1]*:** Planungsphase – Komponentenauswahl, Architektur, Sicherheits- und Backup-Konzept (6 h)
- **KW *[X+2]*:** Durchführung – Server-Provisionierung, DNS, Docker/Coolify/Traefik, SSL, CI/CD, 3 App-Deployments, Firewall-Konfiguration (20 h)
- **KW *[X+3]*:** Abschlussphase – Tests, Soll-Ist-Vergleich, Nachkalkulation (6 h) + Dokumentation & Reflexion (4 h)

7. Erwartete Projektergebnisse

- Vollständig funktionsfähige self-hosted Deployment-Plattform mit automatischem SSL und CI/CD-Integration
- Mindestens 3 dokumentierte Deployments realer Webanwendungen
- Betriebshandbuch mit Installations- und Konfigurationsanleitung für zukünftige Deployments
- Wirtschaftlichkeitsvergleich mit dokumentiertem Kostenvorteil gegenüber externen PaaS-Diensten
- Sicherheitsdokumentation (Firewall-Regeln, SSH-Härtung, Backup-Konzept)
- Testdokumentation mit Vorher-/Nachher-Vergleich der Deployment-Zeiten

8. Qualitätssicherung

Während des Projekts wird eine kontinuierliche Qualitätssicherung durchgeführt:

- Funktionstest jeder deploybaren Anwendung: Erreichbarkeit, HTTPS, automatisches Re-Deployment nach Git-Push
- Sicherheitstest: Port-Scan des Servers (nur 22/80/443 offen), HTTP-Header-Prüfung (HTTPS-Weiterleitung, HSTS)
- Checkliste für jedes Deployment: SSL aktiv, Subdomain erreichbar, Webhook funktionsfähig
- Soll-Ist-Vergleich: tatsächliche Deployment-Zeit dokumentiert und mit Zielwert verglichen
- Nachkalkulation: tatsächliche Kosten vs. Kostenziel und vs. PaaS-Vergleichsangebot

9. Reflexion und Nutzen

Die Umsetzung des Projekts schafft für die AKARA Solutions GmbH eine skalierbare, kosteneffiziente Grundlage für alle zukünftigen Webprojekte. Die monatlichen Infrastrukturkosten sinken auf einen planbaren Fixbetrag – unabhängig von der Anzahl der betriebenen Anwendungen. Gleichzeitig entsteht vollständige technische Kontrolle über Laufzeitumgebung, Konfiguration und Datenhaltung. Der standardisierte Deployment-Prozess reduziert den manuellen Aufwand bei neuen Projekten dauerhaft und erhöht die Reproduzierbarkeit von Deployments. Darüber hinaus entfällt die Abhängigkeit von externen Drittanbietern, was die Datensouveränität des Unternehmens stärkt.