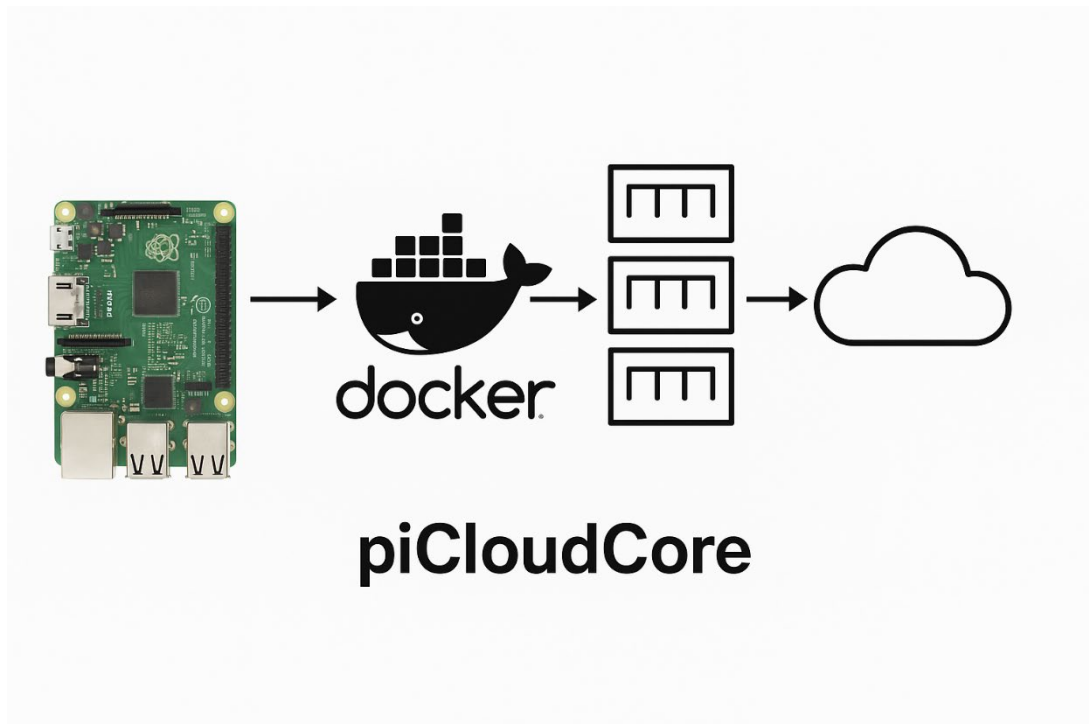


Individuelle Abschlussarbeit BLJ

piCloudcore



Autor: Jann Neururer

Unterschrift *Jann Neururer*

Datum Abgabe: 27.06.2025

Lehrfirma: EKZ Eltop

V2.3

Inhaltsverzeichnis

Änderungstabelle	6
Planung Aus Github Projects.....	7
Einleitung	8
Aufgabenstellung	8
Projektbeschreibung	8
Tagesjournal.....	9
Tagesjournal Mittwoch, 4. Juni 2025	9
Tagesjournal Donnerstag, 5. Juni 2025.....	9
Tagesjournal Freitag, 6. Juni 2025	10
Tagesjournal Mittwoch, 11. Juni 2025	10
Tagesjournal Donnerstag, 12. Juni 2025.....	11
Tagesjournal Freitag, 13. Juni 2025	11
Tagesjournal Mittwoch, 18. Juni 2025	12
Tagesjournal Donnerstag, 19. Juni 2025.....	13
Tagesjournal Freitag, 20. Juni 2025	13
Tagesjournal Mittwoch, 25. Juni 2025	14
Tagesjournal Donnerstag, 26. Juni 2025.....	15
Dienst: Raspberry Pi Grundinstallation.....	16
Ziel	16
Vorbereitung.....	16
Umsetzung	16
Ergebnis	17
Dienst: Uptime Kuma + Projektstruktur + Nginx Reverse Proxy	18
Ziel	18
Vorbereitung.....	18
Umsetzung	18
Ergebnis	20
Dienst: Uptime Kuma + Filebrowser hinter Nginx.....	21
Ziel	21
Vorbereitung.....	21
Umsetzung	21
Probleme.....	22
Ursache.....	22
Lösung	22

Ergebnis	22
Dienst: Traefik Reverse Proxy.....	23
Ziel	23
Vorbereitung.....	23
Umsetzung	23
Probleme.....	25
Lösung	25
Ergebnis	25
Dienst: Bookstack	26
Ziel	26
Vorbereitung.....	26
Umsetzung	26
Probleme.....	27
Lösung	27
Ergebnis	27
Dienst: DNS-Zugriff über Hosts-Datei & Traefik Fehlerbehebung.....	28
Ziel	28
Vorbereitung.....	28
Umsetzung	28
Probleme.....	28
Lösung	29
Ergebnis	29
Dienst: Homer Dashboard	30
Ziel	30
Vorbereitung.....	30
Umsetzung	30
Probleme.....	30
Lösung	31
Ergebnis	31
Dienst: Netdata Monitoring	32
Ziel	32
Vorbereitung.....	32
Umsetzung	32
Thema: Docker Netzwerke & zentrales Startskript	33
Ziel	33

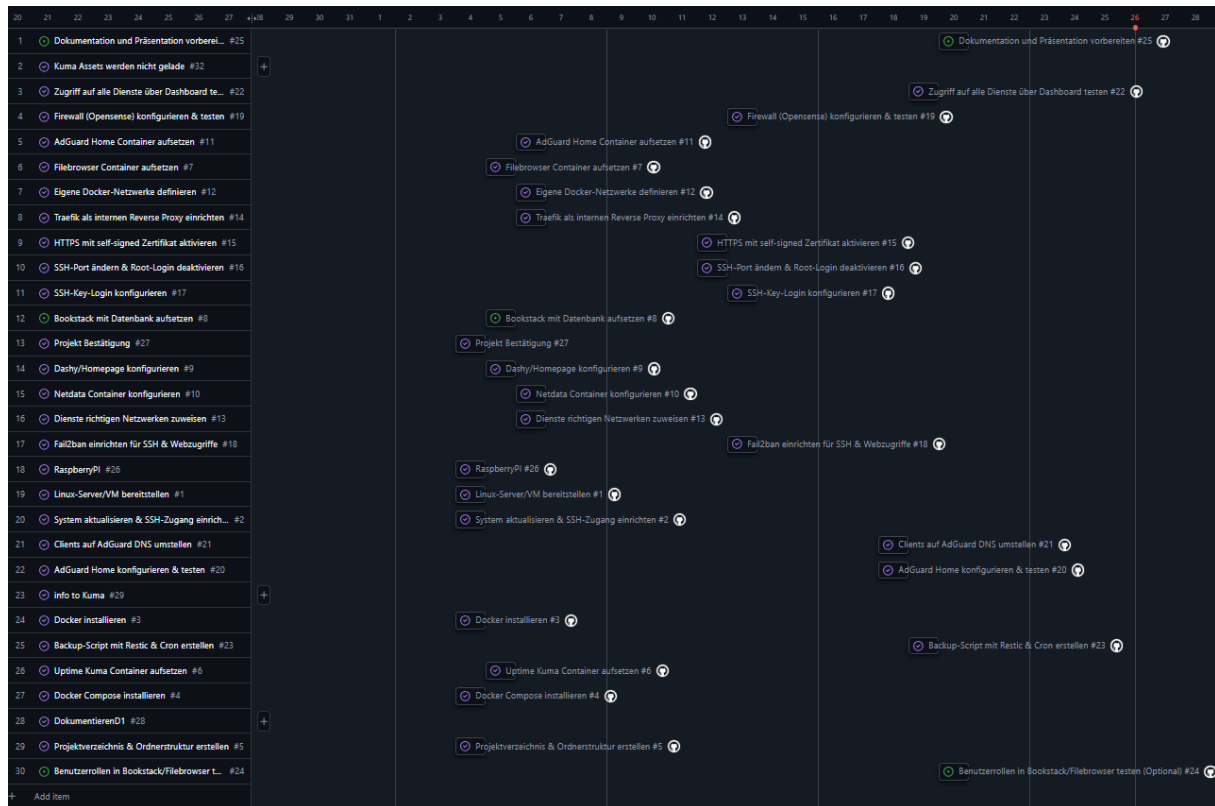
Vorbereitung.....	33
Umsetzung.....	33
Ergebnis	33
Thema: SSH absichern (Port Root Login + Passwort)	34
Ziel.....	34
Vorbereitung.....	34
Umsetzung.....	34
Probleme.....	35
Lösung	35
Ergebnis	35
Dienst: Fail2Ban	36
Ziel.....	36
Vorbereitung.....	36
Umsetzung.....	36
Dienst: AdGuard Home.....	37
Ziel.....	37
Vorbereitung.....	37
Umsetzung.....	37
Ergebnis	38
Thema: Self-Signed TLS-Zertifikat für Traefik	39
Ziel.....	39
Vorbereitung.....	39
Umsetzung.....	39
Probleme.....	40
Ergebnis	40
Dienst: UFW Firewall für Docker Netzwerke	41
Ziel.....	41
Vorbereitung.....	41
Ergebnis	41
Thema: Restic-Backup des gesamten Cloudcore-Projekts	42
Ziel.....	42
Vorbereitung.....	42
Umsetzung.....	42
Ergebnis	43
Thema: AdGuard Home Konfigurieren	44

Ziel	44
Vorbereitung.....	44
Umsetzung	44
Ergebnis	45
Fazit.....	46
Abbildungsverzeichniss	47
Anhang	48
Github Repository	48
Quellenangaben	48
Glossar.....	48
Relevante ki Chat Prompts	49
Infrastruktur:	49
Fehleranalyse, Testing:	49
Sicherheit:	49
Backup, Automation:	49
DNS, Zugriff:.....	49
Checkliste	50

Änderungstabelle

Änderungsdatum	Version	Beschreibung	Wer
04.06.2025	V1.0	Beginn Der Dokumentation	Jann Neururer
05.06.2025	V1.1	Beginn Tag 2	Jann Neururer
06.06.2025	V1.2	Tag 3	Jann Neururer
11.06.2025	V1.3	Tag4	Jann Neururer
12.06.2025	V1.4	Tag 5	Jann Neururer
13.06.2025	V1.5	Tag 6	Jann Neururer
18.06.2025	V1.6	Weiterarbeiten	Jann Neururer
19.06.2025	V2.0	Neuaufbau Doku	Jann Neururer
25.06.2025	V2.1	Grosse Änderungen am Dokument viel neuer Content	Jann Neururer
26.06.2025	V2.2	Vorletzter Tag vor Abgabe	Jann Neururer
27.06.2025	V2.3	Finale Version	Jann Neururer

Planung Aus Github Projects



Einleitung

Aufgabenstellung

Während der Abschlussprojektphase vom 2. bis 27. Juni haben wir die Aufgabe, ein selbstgewähltes IT-Projekt innerhalb von maximal 12 Arbeitstagen zu planen, umzusetzen und zu dokumentieren.

Ziel ist es, ein Thema aus dem eigenen Interesse oder dem Firmenumfeld zu wählen und daraus ein konkretes, praxisnahes Projekt zu entwickeln. Dabei sollen verschiedene Phasen durchlaufen werden: von der Themenfindung über die Planung und Umsetzung bis hin zur Präsentation.

Das Projekt soll technisch sinnvoll, realistisch umsetzbar und nachvollziehbar dokumentiert sein. Es wird erwartet, dass mindestens drei Zwischenziele (Milestones) definiert und erreicht werden. Am Schluss wird das Endergebnis in einer Live-Demo vorgestellt.

Projektbeschreibung

Im Rahmen meines Abschlussprojekts realisiere ich den Aufbau einer sicheren Selfhosting-Plattform auf Basis von Docker. Ziel ist es, verschiedene Dienste wie ein Monitoring-System (Netdata, Uptime Kuma), eine webbasierte Dateiverwaltung (Filebrowser), ein internes Wiki (Bookstack) sowie DNS-Filterung (AdGuard Home) in einer isolierten und übersichtlich verwalteten Umgebung bereitzustellen.

Die Plattform wird auf einem Linux-Server oder Raspberry Pi betrieben und mit OPNsense als zentrale Firewall abgesichert. Zusätzlich werden Mechanismen wie Firewall-Regeln, Fail2ban, Benutzerrollen, Reverse Proxy mit HTTPS und Backups integriert.

Durch dieses Projekt kann ich mein Wissen in den Bereichen Linux, Containerisierung, Netzwerksicherheit und Plattformbetrieb praxisnah anwenden und vertiefen. Die erstellte Umgebung ist modular aufgebaut und auch nach Projektabschluss privat weiter nutzbar.

Tagesjournal

Tagesjournal Mittwoch, 4. Juni 2025

Heute habe ich mit meinem Raspberry Pi Projekt gestartet. Zuerst habe ich das Image mit dem Raspberry Pi Imager auf die SD Karte geschrieben und dabei direkt wichtige Einstellungen wie Hostname, Benutzername, Passwort, Tastatur Layout und Zeitzone konfiguriert.

Danach habe ich mich per SSH auf den Pi verbunden. Anschliessend habe ich in der Datei `/etc/systemd/network/10-eth0.network` eine statische IP-Adresse gesetzt, damit mein Pi im Netzwerk zuverlässig erreichbar bleibt.

Ich habe das System mit `apt update` & `apt upgrade` auf den neusten Stand gebracht und danach Docker über das offizielle Installationsscript (get.docker.com) installiert.

Zum Schluss habe ich noch überprüft, ob Docker Compose verfügbar ist was der Fall war. Jetzt ist der Pi bereit für den Einsatz als Selfhosting Plattform.

Tagesjournal Donnerstag, 5. Juni 2025

Heute habe ich die Grundstruktur meines Projekts aufgebaut. Ich habe ein zentrales Projektverzeichnis namens `cloudcore project` erstellt und darin für jeden Dienst eigene Unterordner angelegt (z. B. `uptime-kuma`, `filebrowser`, `reverse proxy`, etc.).

Anschliessend habe ich den Container für Uptime Kuma gestartet zuerst manuell mit `docker run`, später habe ich ein sauberes `dockercompose.yml` dafür erstellt. Danach konnte ich das Admin Interface aufrufen und den ersten Benutzer anlegen.

Um Uptime Kuma übersichtlich erreichbar zu machen, habe ich Nginx als Reverse Proxy konfiguriert. Dazu habe ich eine eigene `kuma.conf` erstellt und Nginx per Docker Compose in einem eigenen Container gestartet.

Ich habe ausserdem ein eigenes Docker Netzwerk (`kumanet`) angelegt, damit Nginx und Kuma intern miteinander kommunizieren können. Zum Schluss habe ich überprüft, ob der Proxy funktioniert was erfolgreich war.

Ich habe zudem im Dashboard von Kuma den ersten Monitoring Eintrag für den Nginx Container hinzugefügt, um dessen Verfügbarkeit zu prüfen.

Insgesamt konnte ich heute das Grundsystem erfolgreich bereitstellen und vernetzen.

Tagesjournal Freitag, 6. Juni 2025

Heute habe ich versucht, Uptime Kuma und Filebrowser über einen gemeinsamen Nginx Reverse Proxy erreichbar zu machen. Dazu habe ich im Verzeichnis cloudcore project passende dockercompose.yml angelegt.

Alle Container laufen im selben Docker-Netzwerk namens Cloudnet, damit sie miteinander kommunizieren können. Ich habe die default.conf von Nginx so eingerichtet, dass Aufrufe über /kuma und /filebrowser korrekt an die jeweiligen Container weitergeleitet werden.

Für Uptime Kuma habe ich die Variable BASE_PATH=/kuma gesetzt und diesen Pfad auch direkt in der Datenbank gespeichert. Trotzdem wurden beim Aufruf über /kuma keine Styles und Skripte geladen, es kam zu 404 Fehlern. Nach längerer Fehlersuche habe ich herausgefunden, dass Uptime Kuma aktuell nicht gut mit Subpfaden klarkommt.

Tagesjournal Mittwoch, 11. Juni 2025

Heute habe ich mich hauptsächlich mit der Einrichtung von Traefik als Reverse Proxy für meine Docker Container (Uptime Kuma, Filebrowser, Bookstack) beschäftigt. Ich habe Traefik so konfiguriert, dass es als zentraler Reverse Proxy fungiert und den Traffic an die richtigen Container weiterleitet.

Ich habe für jeden Service in den dockercompose.yml Dateien die notwendigen TraefikLabels hinzugefügt, sodass Traefik weiss, wie es die Anfragen über die verschiedenen Subdomains (kuma.local, file.local, bookstack.local) weiterleiten soll. Dazu habe ich auch ein externes Netzwerk, „Cloudnet“, erstellt, damit alle Container miteinander kommunizieren können.

Nach der Konfiguration von Traefik und den Containern habe ich getestet, ob alles funktioniert. Dabei gab es zu Beginn einige Fehler wie „Bad Gateway“ und 404 Not Found, was an falschen oder fehlenden Labels lag. Diese Probleme konnte ich durch Anpassungen der Konfiguration beheben.

Tagesjournal Donnerstag, 12. Juni 2025

Heute habe ich meine Containerstruktur weiter optimiert und Traefik als zentralen Reverse Proxy erfolgreich eingerichtet. Netdata, Homer, Uptime Kuma und Filebrowser sind nun alle stabil über Subdomains im internen Netzwerk erreichbar. Damit habe ich ein funktionierendes Monitoring und Dashboard Interface umgesetzt. Für jeden Dienst habe ich die passenden Traefik Labels gesetzt und ein gemeinsames Docker Netzwerk (cloudnet) erstellt, damit alle Komponenten miteinander kommunizieren können. BookStack funktioniert noch nicht wie gewünscht beim Aufruf erscheint aktuell ein Bad Gateway Fehler. Das schaue ich mir später genauer an. Zusätzlich habe ich ein Backup des Raspberry Pi mit Win32 Disk Imager erstellt, da der Pi Imager nicht funktioniert hat.

Tagesjournal Freitag, 13. Juni 2025

Am Freitag habe ich hauptsächlich bestehende Konfigurationen überprüft, dokumentiert und kleinere Anpassungen vorgenommen. Ich habe z. B. die dockercompose.yml Dateien für Uptime Kuma, Filebrowser und Bookstack nochmals kontrolliert und überarbeitet, um eine einheitliche Struktur und Namensgebung sicherzustellen. Dabei achtete ich besonders auf saubere Netzwerkanbindung an Traefik (Labels, Ports, Netzwerkzuweisung). Zudem habe ich begonnen, Screenshots für die Dokumentation zu sammeln und korrekt zu benennen. Ein Teil des Tages ging auch dafür drauf, mit verschiedenen .local Domains zu testen, ob alle Dienste wie erwartet über Traefik erreichbar sind. Am Nachmittag habe ich mit der Beschreibung der bisherigen Projektschritte für die Abschlussdokumentation begonnen und Inhalte wie Aufbau des Dashboards, Containerstruktur und Serviceeinbindung festgehalten.

Tagesjournal Mittwoch, 18. Juni 2025

Heute habe ich viel an der Sicherheit und Infrastruktur meines Projekts gearbeitet. Am Morgen startete ich damit, den SSH-Zugang auf meinem Raspberry Pi komplett abzusichern. Ich habe den Port auf 2121 geändert, Root Login und Passwortauthentifizierung deaktiviert und stattdessen einen eigenen SSH-Schlüssel eingerichtet. Anfangs gab's ein paar Probleme mit den Dateirechten, aber nach ein paar Korrekturen funktionierte der Login per Key dann einwandfrei. Danach habe ich Fail2Ban eingerichtet, um den SSH-Zugang zusätzlich gegen Brute Force Angriffe zu schützen. Auch hier lief nicht alles sofort rund die Logdateien mussten richtig konfiguriert werden, damit Fail2Ban überhaupt starten konnte. Am Ende lief aber alles stabil, und ich konnte Fail2Ban erfolgreich testen. Ein grösserer Teil des Tages war dann für HTTPS in Traefik reserviert. Ich habe ein selbstsigniertes Zertifikat erstellt und dieses über die dockercompose.yml eingebunden. Zuerst hatte ich einen Fehler in der Syntax (Klammerformat), wodurch Traefik nicht starten wollte aber nach Anpassung ging auch das. Danach konnte ich endlich <https://traefik.local> aufrufen. Im Anschluss habe ich alle anderen Services wie Filebrowser, Uptime Kuma, Homer und Netdata ebenfalls auf HTTPS umgestellt. Dafür mussten die Labels in den Compose-Dateien erweitert werden, was nach ein paar Tests auch funktionierte. Am Schluss habe ich AdGuard Home im Container eingerichtet und konfiguriert. Ich habe die Netzwerkschnittstelle eth0 gewählt und die DNS-Ports gesetzt. Anfangs kam eine 404 Fehlermeldung nach dem Setup, die ich aber beheben konnte. Jetzt läuft AdGuard Home sauber im reverse-Netzwerk und ist unter <http://dns.local> erreichbar. Nebenbei habe ich noch meine dockercompose.yml sauber strukturiert, Start und Stop Skripte genutzt und mein README für GitHub aufgeräumt.

Tagesjournal Donnerstag, 19. Juni 2025

Heute habe ich mich voll auf die Dokumentation konzentriert und dabei entschieden, nochmal komplett von vorne anzufangen. Meine alte Doku war inzwischen ziemlich unübersichtlich, durcheinander und teils doppelt geschrieben. Deshalb habe ich ein komplett neues Worddokument erstellt und angefangen, alles sauber und einheitlich aufzubauen. Zusammen mit ChatGPT habe ich mir ein klares Schema überlegt, wie ich jeden Dienst beschreiben will immer gleich aufgebaut mit Ziel, Vorbereitung, Umsetzung, Probleme, Lösung und Ergebnis. So ist alles verständlich, technisch korrekt und jemand anderes mit dem gleichen Wissensstand könnte die Umsetzung gut nachvollziehen. Ich habe dann viele bereits umgesetzte Teile dokumentiert, unter anderem die Einrichtung vom Raspberry Pi, Docker, Uptime Kuma, den Wechsel von Nginx zu Traefik, SSH-Hardening, Fail2Ban, das Dashboard mit Homer, Netdata und mein Netzwerk-Setup. Dabei habe ich nicht nur Inhalte übernommen, sondern auch direkt verbessert, klarer formuliert und Fehlerstellen bereinigt. Ausserdem habe ich darauf geachtet, auch Probleme realistisch zu dokumentieren zum Beispiel das mit dem Reverse Proxy bei Kuma oder dass ich mich per SSH plötzlich nicht mehr verbinden konnte. Das macht die Doku ehrlicher und zeigt, was wirklich im Projekt passiert ist. Am Ende des Tages hatte ich eine saubere, verständliche und technisch saubere Basis für die restliche Projektdokumentation und endlich ein gutes Gefühl beim Dokumentieren.

Tagesjournal Freitag, 20. Juni 2025

Heute war ich im Homeoffice und habe an meiner Projektdokumentation weitergearbeitet. Nachdem ich gestern eine komplett neue Doku angefangen habe, habe ich heute viele fehlende Abschnitte ergänzt. Ich habe die Einrichtung von AdGuard Home dokumentiert – inklusive Docker-Setup, Traefik-Anbindung und Eintrag im Homer-Dashboard. Danach habe ich den Teil über das selbstsignierte HTTPS-Zertifikat für Traefik geschrieben. Dort habe ich erklärt, wie das Zertifikat erstellt und in die Konfiguration eingebunden wurde. Ausserdem habe ich den Abschnitt über die Docker-Netzwerke ergänzt und beschrieben, wie ich das Startskript aufgebaut habe, mit dem alle Container automatisch gestartet werden. Zum Schluss habe ich noch alle bisherigen Texte durchgelesen, verbessert und begonnen, den Anhang zu strukturieren (Screenshots, Docker-Files usw.).

Tagesjournal Mittwoch, 25. Juni 2025

Heute habe ich mich auf die technische Umsetzung und umfassende Dokumentation meines Projekts konzentriert. Der Schwerpunkt lag dabei auf der Inbetriebnahme von AdGuard Home als DNS-Server im Docker-Netzwerk. Ich habe sichergestellt, dass der Container korrekt auf Port 53 (TCP und UDP) lauscht, DNS-Anfragen verarbeitet und Zugriff auf das Internet hat. Dazu wurden UFW-Firewall-Regeln angepasst, eine NAT-Masquerade-Regel per iptables gesetzt und eine UFW-Route für das Netzwerk reverse eingerichtet. Danach habe ich die DNS-Funktionalität erfolgreich mit dig und nslookup getestet.

Zusätzlich habe ich mehrere Themenbereiche in der Projektdokumentation abgeschlossen oder ergänzt:

Thema: AdGuard Home konfigurieren inklusive DNS-Bindung, Upstream-DNS, Firewall-Regeln und Client-Test

Dienst: AdGuard Home allgemeine Beschreibung des Dienstes im Projektkontext

Dienst: UFW Firewall für Docker Netzwerke mit Fokus auf Netzwerksegmentierung und gezielten Zugriffskontrollen

Thema: Self Signed TLS Zertifikat für Traefik inklusive Erstellung, Einbindung und Sicherheitseinschätzung

Thema: Restic Backup des gesamten Cloudcore Projekts Konfiguration, Script, Cronjob und Restore-Möglichkeit

Ich habe darauf geachtet, alle relevanten Konfigurationen, Befehle und Tests nachvollziehbar zu dokumentieren und die Struktur der Doku weiter zu konsolidieren.

Tagesjournal Donnerstag, 26. Juni 2025

Am Donnerstag stand die finale Umsetzung der HTTPS-Verschlüsselung mit eigenen TLS-Zertifikaten im Fokus. Ich habe mich intensiv mit der Traefik-Konfiguration auseinandergesetzt, um die Zertifikate korrekt über ein separates `dynamic.yaml` einzubinden. Nach mehreren Fehlversuchen mit falscher Pfadangabe, YAML-Fehlern und Containerabstürzen konnte das Problem behoben werden. Die Zertifikate wurden erfolgreich eingebunden und geprüft – der Browser zeigt nun ein selbstsigniertes Zertifikat mit dem korrekten CN (z. B. `traefik.local`) an. Zum Schluss habe ich das Word-Dokument weiter überarbeitet: Ich habe ein automatisches Bildverzeichnis erstellt und ein Gantt-Diagramm eingefügt, um den Projektverlauf visuell darzustellen.

Tagesjournal Freitag, 27. Juni 2025

Heute habe ich den letzten Projekttag genutzt, um die Dokumentation und Demo-Präsentation abzuschliessen. Zuerst habe ich sämtliche Screenshots in den `media/`-Ordner verschoben und mit `git add`, `commit` und `push` sauber ins GitHub-Repository übertragen. Danach habe ich das `README.md` ergänzt inklusive strukturierter Projektübersicht, eingebundenen Bildern und einer vollständigen Dienstetabelle. Ich habe vor dem Mittag noch die Präsentation geübt und gemerkt, dass ich zu viel Zeit benötige, dies muss ich noch anpassen. Zusätzlich habe ich eine eigene `media.md` erstellt, in der alle verwendeten Bilder mit Beschreibung gelistet sind. Für das Wiki habe ich eine Startseite und einen Footer in Markdown geschrieben und hochgeladen. Am Nachmittag habe ich die Live-Demo als Video aufgenommen. Dabei zeigte ich die Funktion aller Container über das Homer-Dashboard, die Verschlüsselung per HTTPS mit self-signed Zertifikat, Uptime Kuma im Live-Betrieb sowie den Restic-Backup-Nachweis. Danach habe ich das Video mit Adobe Premiere Pro geschnitten und erfolgreich exportiert. Zum Abschluss habe ich nochmals alle Punkte auf der Checkliste überprüft und die Dokumentation fertiggestellt.

Dienst: Raspberry Pi Grundinstallation

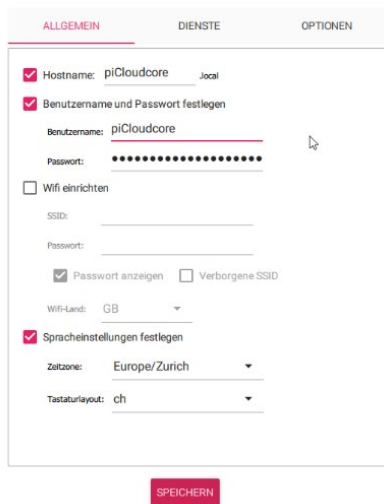
Ziel

Der Raspberry Pi soll für die spätere Selfhosting-Plattform vorbereitet werden. Dazu gehören Netzwerk-Konfiguration, SSH-Zugriff und Docker-Installation.

Vorbereitung

Raspberry Pi Imager verwendet (Raspberry Pi light version)

Einstellungen direkt im Imager gesetzt:



- Hostname: piCloudcore
- Benutzername: piCloudcore
- Passwort definiert
- Sprache: Europe/Zurich
- Tastatur: CH
- WLAN deaktiviert (Verbindung via LAN)

1Pi Grundconfig im Imager

Umsetzung

- Raspberry Pi flashen und konfigurieren mit dem Imager
- Verbindung per SSH: ssh piCloudcore@piCloudcore.local
- Statische IP-Adresse konfiguriert so das der Pi immer dieselbe IP hat das es keine komplikationen mit Netzwerkkänderungen gibt: **/etc/systemd/network/10-eth0.network**

```
[match]
Name=eth0

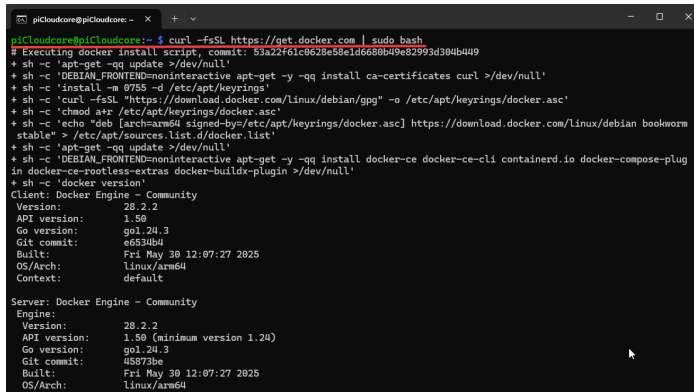
[Network]
Address=10.80.4.210/16
Gateway=10.80.4.1
DNS=10.80.4.1
```

2Static IP setzen

Docker installiert mit:

curl -fsSL https://get.docker.com | sudo bash

Docker Compose war bereits enthalten da es eine neuere Version ist muss man es nicht mehr separat Installieren.



```
piCloudcore@piCloudcore:~$ curl -fsSL https://get.docker.com | sudo bash
# Executing docker install script, commit: 53a22f61c8628e58e1d6688b49e82993d394b449
+ sh -c 'apt-get -qq update >/dev/null'
+ sh -c 'DEBIAN_FRONTEND=noninteractive apt-get -y -qq install ca-certificates curl >/dev/null'
+ sh -c 'install -m 0755 -d /etc/apt/keyrings'
+ sh -c 'curl -fsSL "https://download.docker.com/linux/debian/gpg" -o /etc/apt/keyrings/docker.asc'
+ sh -c 'chmod a+r /etc/apt/keyrings/docker.asc'
+ sh -c 'echo "deb [arch=arm64 signed-by=/etc/apt/keyrings/docker.asc] https://download.docker.com/linux/debian bookworm
stable" > /etc/apt/sources.list.d/docker.list'
+ sh -c 'apt-get -qq update >/dev/null'
+ sh -c 'DEBIAN_FRONTEND=noninteractive apt-get -y -qq install docker-ce docker-ce-cli containerd.io docker-compose-plug
in docker-ce-rootless-extras docker-buildx-plugin >/dev/null'
+ sh -c 'docker version'
Client: Docker Engine - Community
 Version: 28.2.2
 API version: 1.50
 Go version: go1.24.3
 Git commit: e6534b4
 Built: Fri May 30 12:07:27 2025
 OS/Arch: linux/arm64
 Context: default

Server: Docker Engine - Community
 Engine:
  Version: 28.2.2
  API version: 1.50 (minimum version 1.24)
  Go version: go1.24.3
  Git commit: 45873be
  Built: Fri May 30 12:07:27 2025
  OS/Arch: linux/arm64
```

3Docker Installation

Ergebnis

Der Raspberry Pi ist vollständig vorbereitet:

- feste IP-Adresse
- per SSH erreichbar
- Docker & Compose einsatzbereit für die Selfhosting-Plattform.

Dienst: Uptime Kuma + Projektstruktur + Nginx Reverse Proxy

Ziel

Einrichtung einer modularen Selfhosting-Plattform mit Docker. Als ersten Dienst soll Uptime Kuma zur Überwachung laufen und über einen Reverse Proxy erreichbar sein.

Vorbereitung

Zentrales Verzeichnis cloudcore-project erstellt

Unterordner für jeden Dienst angelegt (z. B. uptime-kuma/, reverse-proxy/)

Ziel: klare Struktur & einfache Wartung mit docker-compose

Reverse Proxy: Nginx

Uptime Kuma soll später per `http://<IP>/kuma` erreichbar sein

Umsetzung

Projektstruktur:

Kopieren Bearbeiten

cloudcore-project/

├─ uptime-kuma/

├─ reverse-proxy/nginx/

└─ ...

```
drwxr-xr-x 2 root root 4096 Jun 5 09:06 adguard
drwxr-xr-x 2 root root 4096 Jun 5 09:06 bookstack
drwxr-xr-x 2 root root 4096 Jun 5 09:06 dashy
drwxr-xr-x 2 root root 4096 Jun 5 09:06 filebrowser
drwxr-xr-x 2 root root 4096 Jun 5 09:06 netdata
drwxr-xr-x 3 root root 4096 Jun 5 09:07 reverse-proxy
drwxr-xr-x 2 root root 4096 Jun 5 09:05 uptime-kuma
```

4Projektstruktur

Uptime Kuma Container gestartet mit:

`docker run -d --restart=unless-stopped \`

`-p 3001:3001 -v ./data:/app/data \`

`--name uptime-kuma louislam/uptime-kuma:1`

Danach Umstellung auf docker-compose.yml für den Service.

```
GNU nano 7.2 docker-compose.yml
version: '3.8'

services:
  uptime-kuma:
    image: louislam/uptime-kuma:1
    container_name: uptime-kuma
    restart: unless-stopped
    ports:
      - "3001:3001"
    volumes:
      - uptime-kuma-vol:/app/data

volumes:
  uptime-kuma-vol:
```

5Kuma Compose File

Admin-Konto erstellt beim ersten Web-Zugriff auf <http://10.80.4.210:3001>

Benutzername: piCloudcore

Passwort: Zli12345

Nginx Reverse Proxy konfiguriert:

- Datei kuma.conf im Pfad reverse-proxy/nginx/ erstellt
- Ziel: Weiterleitung von /kuma an den Container

```
GNU nano 7.2
server {
    listen 80;
    server_name uptime.local;

    location / {
        proxy_pass http://uptime-kuma:3001;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
        proxy_set_header Host $host;
    }
}
```

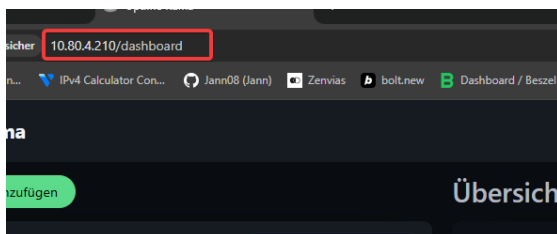
6Nginx Proxy einstellung

Gemeinsames Netzwerk kuma-net erstellt: **docker network create kuma-net**

Beide Container Kuma + Proxy ins Netzwerk eingebunden

- Kommunikation via Hostnamen (uptime-kuma)

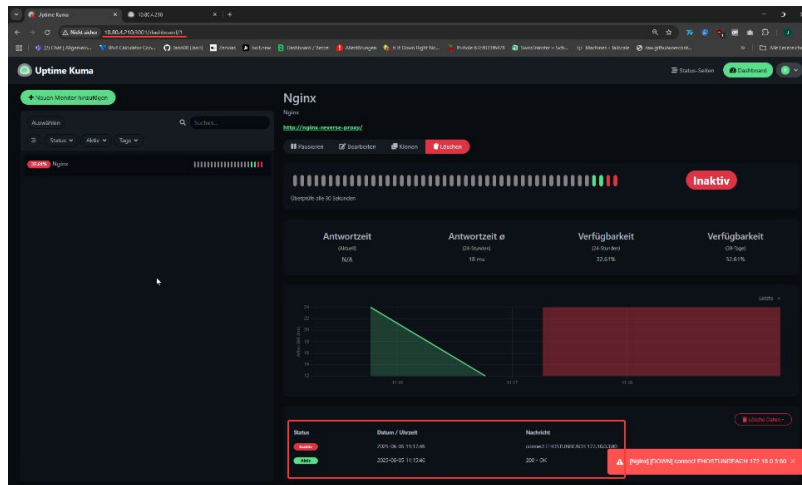
Dashboard erfolgreich über Proxy erreichbar



7Dashboard kuma erreichen

Monitoring-Test:

- Nginx-Container in Kuma als erster Monitor hinzugefügt
- Statusanzeige bestätigt: Kommunikation & Verfügbarkeit korrekt



8Kuma Dashboard

Ergebnis

- Selfhosting-Projektstruktur steht
- Uptime Kuma läuft stabil
- Erreichbar über den Reverse Proxy
- Monitoring aktiv erster Dienst einsatzbereit

Dienst: Uptime Kuma + Filebrowser hinter Nginx

Ziel

Zwei Webdienste (Uptime Kuma + Filebrowser) sollen über denselben Reverse Proxy (Nginx) erreichbar sein, jeweils unter einem eigenen Subpfad.

Vorbereitung

Gemeinsames Docker-Netzwerk Cloudnet erstellt:

docker network create --attachable --driver bridge Cloudnet

Für beide Dienste eigene docker-compose.yml erstellt

Reverse Proxy Nginx so konfiguriert, dass:

- `http://10.80.4.210/kuma` → Uptime Kuma
- `http://10.80.4.210/filebrowser` → Filebrowser

```
server {
    listen 80;

    location = /kuma {
        return 301 /kuma/;
    }

    location /kuma/ {
        proxy_pass http://uptime-kuma:3001/kuma/;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
    }

    location /filebrowser {
        proxy_buffers 8 32k;
        proxy_buffer_size 64k;
        client_max_body_size 75M;

        proxy_pass http://filebrowser:8080;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;

        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
        proxy_read_timeout 999999999;
    }
}
```

10Reverse Proxy kuma einstellen

```
services:
  uptime-kuma:
    image: louislam/uptime-kuma:1
    container_name: uptime-kuma
    restart: unless-stopped
    ports:
      - "3001:3001"
    volumes:
      - uptime-kuma-vol:/app/data
    networks:
      - Cloudnet
    environment:
      - BASE_PATH=/kuma/

volumes:
  uptime-kuma-vol:

networks:
  Cloudnet:
    external: true
```

9Kuma Netzwerk und basepath

```
services:
  filebrowser:
    image: hurlenko/filebrowser
    container_name: filebrowser
    user: "1000:1000"
    environment:
      - FB_BASEURL=/filebrowser
    volumes:
      - ./data:/data
      - ./config:/config
    networks:
      - Cloudnet
    restart: unless-stopped

networks:
  Cloudnet:
    external: true
```

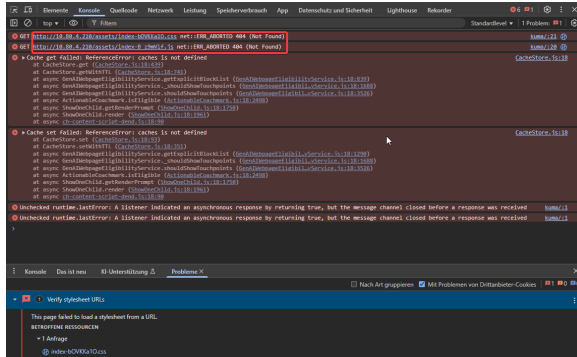
11Filebrowser yaml file

Umsetzung

1. filebrowser über ein bestehendes GitHub-Repo eingerichtet (Docker Compose + Nginx config übernommen)
2. Beide Container in das Netzwerk Cloudnet eingebunden
3. Nginx-Konfigdatei angepasst, sodass beide Pfade korrekt an die internen Container weiterleiten
4. Uptime Kuma mit `BASE_PATH=/kuma` gestartet
5. Zusätzlich `basePath=/kuma` direkt in `kuma.db` gesetzt
6. Neues Volume angelegt, um alte Konfiguration zu entfernen
7. Browserzugriff getestet

Probleme

- Uptime Kuma lädt unter /kuma nur weissen Bildschirm
- Fehler: Assets werden fälschlicherweise unter /assets/ statt /kuma/assets/ gesucht



12 Google Chrome error logs Kuma

Ursache

- Uptime Kuma lädt Frontend-Pfade nicht korrekt relativ zum BASE_PATH es wird meine Pfad Angabe ignoriert und den Standardpfad benutzt.
- Assets bleiben auf absolute Pfade
- Trotz korrekter Konfiguration in ENV-Variable und DB lädt Kuma falsch

```
location /kuma/ {
    proxy_pass http://uptime-kuma:3001/kuma/;
}
```

13 Location der Kuma Assets

Lösung

Trotz folgender Versuche keine Besserung:

- BASE_PATH korrekt gesetzt (ENV + Datenbank)
- Volume neu angelegt
- Zugriff nur über /kuma/
- Nginx-Redirect location = /kuma { return 301 /kuma/; }
- Debugging mit Hilfe von ChatGPT

Fazit: Uptime Kuma ist nicht reverse-proxy-tauglich mit Subpfaden, problematisch bei statischen Assets

Ergebnis

- Filebrowser funktioniert über /filebrowser problemlos
- Uptime Kuma funktioniert hinter Nginx auf Subpfad /kuma **nicht stabil**
- Entscheidung: Proxy-Setup später mit Traefik neu aufbauen

Dienst: Traefik Reverse Proxy

Ziel

Einrichtung von Traefik als zentralem Reverse Proxy für alle Docker-Dienste. Ziel ist eine zuverlässige, flexible Weiterleitung per Subdomain (z. B. kuma.local) statt fehleranfällige Subpfade wie bei Nginx.

Vorbereitung

Nginx hatte Probleme mit BASE_PATH z. B. bei Uptime Kuma

Entscheidung zur frühzeitigen Umstellung auf Traefik fiel während des Projekts

Ziel: sauberes, containerbasiertes Routing über Labels

Umsetzung

1. Traefik-Container mit docker-compose.yml aus der offiziellen Doku eingerichtet

Quelle: <https://doc.traefik.io/traefik/>

2. Reverse Proxy lauscht auf Port 80 (entrypoint web) und 443 (entrypoint websecure)
3. Traefik-Konfiguration nutzt statisches traefik.yml + dynamisches Label-Routing

```
version: "3.8"

services:
  traefik:
    image: traefik:v3.0
    container_name: traefik
    labels:
      - "traefik.enable=true"
      - "traefik.http.routers.api.rule=Host('traefik.local')"
      - "traefik.http.routers.api.service=api@internal"
      - "traefik.http.routers.api.entrypoints=web"
    command:
      - "--api.dashboard=true"
      - "--providers.docker=true"
      - "--providers.docker.exposedbydefault=false"
      - "--entrypoints.web.address=:80"
    ports:
      - "80:80"
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock:ro
    networks:
      - Cloudnet

networks:
  Cloudnet:
    external: true
```

14Traefik Compose file

Für alle Dienste wurden entsprechende Labels hinzugefügt:

```
labels:
  - "traefik.http.routers.kuma.rule=Host(`kuma.local`)"
  - "traefik.http.services.kuma.loadbalancer.server.port=3001"
  - "traefik.enable=true"
```

15 Labels um Traefik zu nutzen

```
services:
  uptime-kuma:
    image: louislam/uptime-kuma:1
    container_name: uptime-kuma
    labels:
      - "traefik.enable=true"
      - "traefik.docker.network=Cloudnet"
      - "traefik.http.routers.kuma.rule=Host(`kuma.local`)"
      - "traefik.http.routers.kuma.entrypoints=web"
      - "traefik.http.services.kuma.loadbalancer.server.port=3001"
    restart: unless-stopped
    volumes:
      - uptime-kuma-vol:/app/data
    networks:
      - Cloudnet
    environment:
      - BASE_PATH=
volumes:
  uptime-kuma-vol:
networks:
  Cloudnet:
    external: true
```

16 kuma mit labels

```
services:
  filebrowser:
    image: hurlenko/filebrowser
    container_name: filebrowser
    labels:
      - "traefik.enable=true"
      - "traefik.docker.network=Cloudnet"
      - "traefik.http.routers.filebrowser.rule=Host(`file.local`)"
      - "traefik.http.routers.filebrowser.entrypoints=web"
      - "traefik.http.services.filebrowser.loadbalancer.server.port=8080"
    user: "1000:1000"
    environment:
      - FB_BASEURL=/
    volumes:
      - ./data:/data
      - ./config:/config
    networks:
      - Cloudnet
    restart: unless-stopped
networks:
  Cloudnet:
    external: true
```

17 Filebrowser mit labels

Netzwerk cloudnet verwendet, in dem auch alle anderen Dienste verbunden sind

Dienste wurden nach und nach an Traefik angebunden

Probleme

- Initiale Umstellung von Nginx zu Traefik erforderte Änderung aller docker-compose.yml
- Einige Dienste hatten falsche Labels oder fehlende Netzwerkzuweisungen.
- TLS musste manuell per Self-Signed-Zertifikat eingerichtet werden (später gelöst)

Lösung

- Unterstützung durch ChatGPT bei Label Fehlern und Struktur
- Fehlerhafte Labels und Netzwerknamen korrigiert
- TLS-Zertifikate eingebunden via Traefik-Konfiguration
- Nginx dienst entfernt Traefik vollständig übernommen

Ergebnis

- Traefik funktioniert stabil als Reverse Proxy
- Alle Services sind über individuelle Subdomains erreichbar
- Keine weiteren BASE_PATH-Probleme wie bei Nginx

Dienst: Bookstack

Ziel

Einrichtung von Bookstack als internes Wiki auf der Selfhosting-Plattform. Der Dienst soll über Traefik per Subdomain erreichbar sein und seine Daten in einer SQL-Datenbank speichern.

Vorbereitung

Container basiert auf linuxserver/bookstack

Bookstack benötigt eine angebundene Datenbank (MySQL/MariaDB)

Konfiguration erfolgt über Umgebungsvariablen und persistente Volumes

Traefik soll als Reverse Proxy verwendet werden

Umsetzung

1. docker-compose.yml geschrieben auf Basis des offiziellen Images
2. Volume-Mounts definiert, damit Datenbank und Bookstack-Daten erhalten bleiben
3. Datenbankverbindung automatisch über das Dockerfile eingerichtet
4. Labels für Traefik konfiguriert mit Hilfe von ChatGPT;

```
version: '3.8'

services:
  bookstack-db:
    image: mysql:8
    container_name: bookstack-db
    restart: unless-stopped
    environment:
      MYSQL_ROOT_PASSWORD: Z1i12345
      MYSQL_DATABASE: bookstack
      MYSQL_USER: piCloudcore
      MYSQL_PASSWORD: Z1i12345
    volumes:
      - db_data:/var/lib/mysql
    networks:
      - Cloudnet

  bookstack:
    image: lscr.io/linuxserver/bookstack:latest
    container_name: bookstack
    labels:
      - "traefik.enable=true"
      - "traefik.docker.network=Cloudnet"
      - "traefik.http.routers.bookstack.rule=Host(`bookstack.local`)"
      - "traefik.http.routers.bookstack.entrypoints=web"
      - "traefik.http.services.bookstack.loadbalancer.server.port=80"
    restart: unless-stopped
    environment:
      - DB_HOST=bookstack-db
      - DB_USER=piCloudcore
      - DB_PASS=Z1i12345
      - DB_DATABASE=bookstack
      - APP_URL=http://bookstack.local
    volumes:
      - bookstack_data:/config
    networks:
      - Cloudnet
    depends_on:
      - bookstack-db

volumes:
  db_data:
  bookstack_data:

networks:
  Cloudnet:
    external: true
```

18Bookstack yml file

5. Container gestartet → Volume und Containerstruktur wurden korrekt erstellt

```
[+] Running 4/4
✔ Volume "bookstack_bookstack_data" Created
✔ Volume "bookstack_db_data" Created
✔ Container bookstack-db Started
✔ Container bookstack Started
piCloudcore@piCloudcore:~/cloudcore-project/bookstack $
```

19Bookstack richtig erstellt

Probleme

- Beim Zugriff über <http://book.local> erscheint **Bad Gateway**
- Bookstack-Dienst scheint zwar zu laufen, aber der Webserver antwortet nicht wie erwartet
- Ursache bisher unklar (möglicherweise DB-Verbindung, interner Port, Healthcheck)

Lösung

Noch keine Lösung gefunden. Debugging hat bis jetzt nichts gebracht.

Ergebnis

- Container läuft, Traefik-Weiterleitung ist konfiguriert
- Bookstack ist derzeit nicht erreichbar
- Fehlerbehebung wird bei voriger zeit später durchgeführt.

Dienst: DNS-Zugriff über Hosts-Datei & Traefik

Fehlerbehebung

Ziel

Alle Services sollen über sprechende Subdomains wie kuma.local, file.local, book.local erreichbar sein ohne externes DNS. Zusätzlich sollen interne Netzwerk- und Routing-Probleme gelöst werden.

Vorbereitung

Auf dem lokalen Client wurde die Datei C:\Windows\System32\drivers\etc\hosts angepasst

```
# Custom local reverse proxy entries  
10.80.4.210 kuma.local file.local bookstack.local traefik.local
```

20Inhalt der Hosts Datei auf Client

So lassen sich alle Services direkt über den Browser aufrufen

Umsetzung

1. Hosts-Datei bearbeitet
2. Container-Namen & Labels in Docker überprüft
3. Netzwerkdefinitionen kontrolliert

Probleme

- **404 Page Not Found** und **Bad Gateway** bei fast allen Diensten
- Ursache 1: In Docker Compose wurde das Netzwerk einmal als Cloudnet (gross) und einmal als cloudnet (klein) benannt, keine Kommunikation
- Ursache 2: Labels für Traefik waren inkorrekt oder unvollständig
- Ursache 3: BASE_PATH war an manchen Stellen noch aktiv (veraltet durch Traefik)

Lösung

- Netzwerknamen in allen docker-compose.yml korrigiert und einheitlich als Cloudnet neu erstellt
- Alle Labels überarbeitet, mit Hilfe von ChatGPT
- BASE_PATH-Variablen entfernt, da Routing über Traefik erfolgt

Ergebnis

- Alle Subdomains funktionieren lokal über die Hosts-Datei
- Traefik leitet Anfragen korrekt weiter
- Keine 404- oder Gateway-Fehler mehr ausgenommen Bookstack

Dienst: Homer Dashboard

Ziel

Ein zentrales Web-Dashboard, das alle Selfhosting-Services (Uptime Kuma, Bookstack) übersichtlich auflistet und verlinkt. Zugriff soll über <http://homer.local> erfolgen.

Vorbereitung

Neuer Ordner homer/ erstellt

Docker Compose Setup mit Labels für Traefik vorbereitet

Konfiguration (config.yml) soll aus ./homer/assets geladen werden

Umsetzung

1. docker-compose.yml in homer/ erstellt
2. Traefik-Labels gesetzt
3. config.yml geschrieben mit zwei Kategorien:
 - **Monitoring** (Uptime Kuma, netdata)
 - **Tools** (Bookstack, filebrowser)

```
version: '3.8'

services:
  homer:
    image: b4bz/homer:latest
    container_name: homer
    volumes:
      - ./homer/assets:/www/assets
    labels:
      - "traefik.enable=true"
      - "traefik.http.routers.homer.rule=Host(`homer.local`)"
      - "traefik.http.routers.homer.entrypoints=web"
    networks:
      - traefik
    restart: unless-stopped

networks:
  traefik:
    external: true
```

21Homer Dashboard yaml

Dashboard verlinkt alle Dienste über ihre Subdomains

Docker Volume bzw. Bind Mount definiert:

./homer/assets:/www/assets

```
title: Projekt Dashboard
subtitle: Übersicht über alle Dienste
theme: default
logo: ""

services:
  - name: Monitoring
    icon: fas fa-chart-line
    items:
      - name: Uptime Kuma
        url: http://kuma.local
        icon: uptime-kuma.png
      - name: Traefik
        url: http://traefik.local
        icon: traefik.png
      - name: Netdata
        url: http://netdata.local
        icon: netdata.png
  - name: Tools
    icon: fas fa-toolbox
    items:
      - name: Bookstack
        url: http://bookstack.local
        icon: bookstack.png
      - name: Filebrowser
        url: http://file.local
        icon: filebrowser.png
```

22Homer Conf für dienste

Probleme

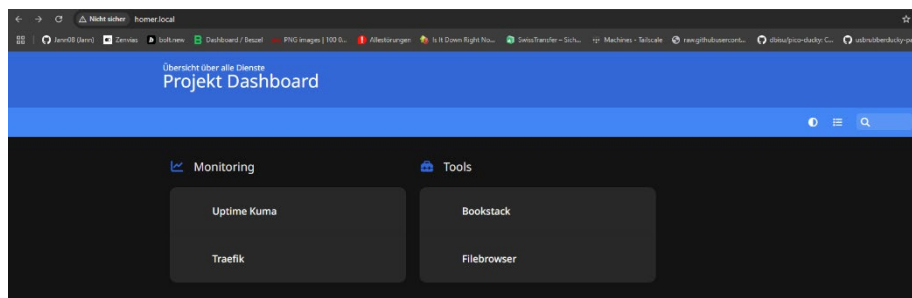
- Beim ersten Start wurde nur die Fehlermeldung **No configuration found** angezeigt
- Ursache: config.yml lag im falschen Verzeichnis (nicht in /www/assets)

Lösung

- Datei korrekt nach `./homer/assets/config.yml` verschoben
- Container neu gestartet → Dashboard geladen

Ergebnis

- Homer-Dashboard ist über `http://homer.local` erreichbar
- Alle Dienste sind in Gruppen kategorisiert und direkt anklickbar
- Übersichtliche Steuerzentrale für alle Selfhosted-Services erfolgreich eingerichtet



Dashboard von Homer funktioniert

Dienst: Netdata Monitoring

Ziel

Systemmonitoring auf dem Raspberry Pi via Netdata in Echtzeit. Der Dienst soll Metriken wie CPU-Auslastung, Netzwerk, Prozesse und Containerdaten anzeigen ohne auf separate Konfigurationen angewiesen zu sein.

Vorbereitung

Netdata wird direkt im host-Modus betrieben, um vollständigen Zugriff auf Systemdaten zu erhalten

Container läuft dauerhaft und startet automatisch neu

Zugriff erfolgt über <https://netdata.local>

Umsetzung

1. Docker-Compose-Datei erstellt:

```
version: '3.8'

services:
  netdata:
    image: netdata/netdata:latest
    container_name: netdata
    pid: host
    network_mode: host
    cap_add:
      - SYS_PTRACE
    security_opt:
      - apparmor:unconfined
    volumes:
      - /etc/passwd:/host/etc/passwd:ro
      - /etc/group:/host/etc/group:ro
      - /proc:/host/proc:ro
      - /sys:/host/sys:ro
      - /etc/os-release:/host/etc/os-release:ro
    restart: unless-stopped
```

24Netdata yaml

2. Container gestartet mit:

docker compose up -d

3. Erste Verbindung im Browser geöffnet → Login-Screen erscheint

4. Um die Session zu verbinden, wurde folgender

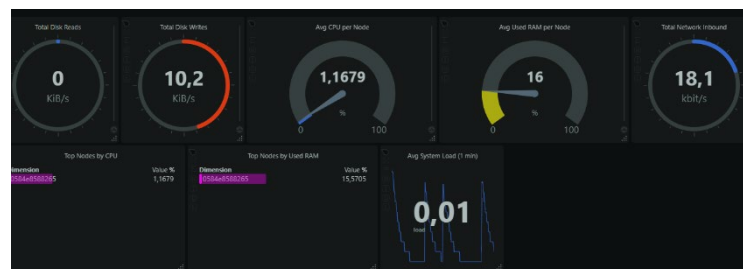
Befehl im Terminal ausgeführt:

docker exec netdata cat /var/lib/netdata/netdata_random_session_id

Den angezeigten **Private Key** in das Feld im UI eingefügt → Zugriff freigeschaltet

Ergebnis

- Netdata läuft stabil im Host-Modus
- Zugriff auf alle relevanten Metriken des Raspberry Pi möglich
- Echtzeitüberwachung über <http://netdata.local>



25Netdata Monitoring

Thema: Docker Netzwerke & zentrales Startskript

Ziel

Alle Container sollen in einem gemeinsamen, logisch getrennten Docker-Netzwerk laufen. Zusätzlich soll ein Skript alle Container geordnet und automatisch starten können.

Vorbereitung

Zwei Docker-Netzwerke manuell erstellt:

reverse Für Dienste mit Webzugriff über Traefik (Homer, Filebrowser, Netdata)

internal nur für interne Kommunikation, Bookstack zu Datenbank

Umsetzung

1. Netzwerkzuweisung in jeder dockercompose.yml sichergestellt

networks:

Reverse

Internal

NAMES	NETWORKS
filebrowser	reverse
uptime-kuma	reverse
bookstack	reverse, internal
bookstack-db	internal
homer	reverse
netdata	reverse
traefik	reverse

26Netzwerke

Bookstack verwendet beide Netzwerke (reverse, internal), um Webzugriff und DBAnbindung zu ermöglichen

```
set -e # Instant Shutdown on error

cd "$(dirname "$0")" # changes to the directory

cd reverse-proxy/traefik && docker compose up -d
cd ../../netdata && docker compose up -d
cd ../homer && docker compose up -d
cd ../bookstack && docker compose up -d
cd ../uptime-kuma && docker compose up -d
cd ../filebrowser && docker compose up -d
```

27Dockercontainer Start script

Bash Skript erstellt, um alle Dienste nacheinander zu starten und zu stoppen:

Dieses Script ist Ausbau fähig da es nur zuverlässig ist wen die Ordnerstruktur nicht verändert wird oder das Script am richtigen Ort ausgeführt wird.

Ergebnis

- Alle Container sind korrekt in die vorgesehenen Netzwerke eingebunden
- Reverse Dienste erreichbar über Traefik
- Start Skript vereinfacht das Hochfahren der gesamten Plattform erheblich

Thema: SSH absichern (Port Root Login + Passwort)

Ziel

Der SSH Zugang zum Raspberry Pi soll sicherer gemacht werden:

- Kein Root Login
- Keine Passwort Authentifizierung
- Verbindung nur über SSH Schlüssel
- SSH Port von Standard 22 auf 2121 geändert

Vorbereitung

Bestehende SSH Verbindung testen

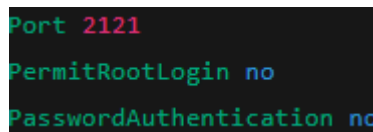
SSH Key wird auf dem Client generiert

Server Konfiguration wird angepasst

Umsetzung

1. SSH Config ändern auf Pi im file:

```
sudo nano /etc/ssh/sshd_config
```



```
Port 2121
PermitRootLogin no
PasswordAuthentication no
```

28SSHD config

2. SSH Key generieren auf dem Client:

```
ssh-keygen -t ed25519 -C piCloudcore
```

3. Key auf dem Pi hinterlegen im file:

```
nano ~/.ssh/authorized_keys
```

Key sollte ca so aussehen:

```
ssh-ed25519
```

```
AAAAC3NzaC1lZDI1NTE5AAAAIKt0qkxuikXAAP5O0PdbsmrZNvoy22DOSL5f2ZuVWX  
Kd piCloudcore
```

4. Dateiberechtigungen setzen:

```
chmod 700 ~/.ssh
```

```
chmod 600 ~/.ssh/authorized_keys
```

```
chown piCloudcore:piCloudcore ~/.ssh/authorized_keys
```

5. Wenn alles richtig eingerichtet ist kann auf den RaspberryPi zugegriffen werden ohne ein Passwort einzugeben und zwar mit diesem Command:
- ssh -i ~/.ssh/piprojectSSH -p 2121 piCloudcore@10.80.4.210**

Probleme

- Der SSHmKey funktionierte anfangs nicht
- Grund: authorized_keys war im Besitz von root, nicht von piCloudcore daher konnte der User nicht darauf zugreifen.

Lösung

- Besitzer mit chown korrigiert
- Danach funktionierte der Login über Port 2121 sofort via SSHmKey

Ergebnis

- SSH ist abgesichert
- Nur noch Login mit ed25519 Key über Port 2121 erlaubt
- Root Zugang und Passwort Anmeldung sind deaktiviert

Dienst: Fail2Ban

Ziel

Zusätzlicher Schutz gegen Brute Force Angriffe auf den SSH Dienst. Fail2Ban soll verdächtige Login Versuche automatisch blockieren.

Vorbereitung

SSH bereits gehärtet (kein Root Login, nur Key Zugriff)

Fail2Ban wird zusätzlich installiert und für SSH aktiviert

Umsetzung

1. Installation:

```
sudo apt install fail2ban -y
```

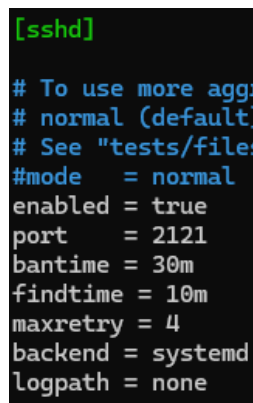
Konfiguration aktivieren:

2. Originaldatei kopieren:

```
sudo cp /etc/fail2ban/jail.conf /etc/fail2ban/jail.local
```

Dann bearbeiten unter [sshd]:

```
sudo nano /etc/fail2ban/jail.local
```



```
[sshd]
# To use more aggressive action
# normal (default)
# See "tests/files"
#mode = normal
enabled = true
port = 2121
bantime = 30m
findtime = 10m
maxretry = 4
backend = systemd
logpath = none
```

29Fail2ban settigs

Fail2Ban neu starten:

```
sudo systemctl restart fail2ban
```

Dienst: AdGuard Home

Ziel

Ein DNS-basiertes Werbeblockingsystem lokal betreiben. AdGuard Home soll über Traefik erreichbar sein und im Homer-Dashboard als zentraler Eintrag gelistet werden.

Vorbereitung

Zugriff über dns.local

Traefik übernimmt das Routing

Konfig & Daten werden persistent gespeichert

Visualisierung im Homer-Dashboard

Umsetzung

1. Ordner adguard-home/ erstellt
2. Docker-Compose-Datei mit Traefik-Labels eingerichtet:

```
services:
  adguard:
    image: adguard/adguardhome
    container_name: adguard
    restart: unless-stopped
    networks:
      - reverse
    volumes:
      - ./conf:/opt/adguardhome/conf
      - ./work:/opt/adguardhome/work
    labels:
      - "traefik.enable=true"
      - "traefik.docker.network=reverse"
      - "traefik.http.routers.adguard.rule=Host(`dns.local`)"
      - "traefik.http.routers.adguard.entrypoints=websecure"
      - "traefik.http.routers.adguard.tls=true"
      - "traefik.http.services.adguard.loadbalancer.server.port=80"

networks:
  reverse:
    external: true
```

30Adguard yaml

3. Hostdatei anpassen im host file unter C:\Windows\System32\drivers\etc

```
# Custom local reverse proxy entries
10.80.4.210 kuma.local file.local bookstack.local traefik.local homer.local netdata.local dns.local
```

31Hostdatei

4. Container starten:
Docker compose up -d

5. Webinterface über `http://dns.local` öffnen und auf Funktion testen.
6. Homer-Eintrag hinzugefügt in `config.yml`:

```
- name: Security
  icon:
  items:
    - name: AdGuard
      url: https://dns.local
```

32Homer eintrag

Ergebnis

AdGuard Home läuft im Reverse-Proxy-Netz reverse

Zugriff via `dns.local` funktioniert stabil

Dienst erscheint korrekt im Homer-Dashboard unter Security

Thema: Self-Signed TLS-Zertifikat für Traefik

Ziel

Traefik soll alle Services über HTTPS ausliefern. Dafür wird ein selbstsigniertes Zertifikat verwendet, das lokal generiert und manuell eingebunden wird.

Vorbereitung

Ziel: HTTPS für lokale Domains (kuma.local, dns.local)

Zertifikat & Key sollen lokal gespeichert und dauerhaft eingebunden werden

Keine öffentliche CA notwendig – Zertifikat wird in Browser als Ausnahme akzeptiert

Umsetzung

1. Ordnerstruktur für Zertifikate angelegt mit:

```
mkdir -p ~/certs
```

```
cd ~/certs
```

2. Zertifikat & Key erzeugt mit:

```
openssl req -x509 -newkey rsa:4096 -nodes \  
-keyout self.key -out self.crt -days 365 \  
-subj "/CN=piCloudcore.local"
```

3. Ergebnis:

- self.crt: öffentliches Zertifikat
- self.key: privater Schlüssel

4. Dateien nach Traefik verschoben mit:

```
cp ~/certs/self.* ~/cloudcore-project/reverse-proxy/traefik/certs/
```

5. traefik.yml erweitert mit certs Volume und dem File Directory certs:

```
services:
  D Run Service
  traefik:
    image: traefik:v3.0
    container_name: traefik
    labels:
      - "traefik.enable=true"
      - "traefik.http.routers.api.rule=Host(`traefik.local`)"
      - "traefik.http.routers.api.service=api@internal"
      - "traefik.http.routers.api.entrypoints=websecure"
      - "traefik.http.routers.api.tls=true"
    command:
      - "--entrypoints.web.address=:80"
      - "--entrypoints.websecure.address=:443"
      - "--entrypoints.websecure.http.tls=true"
      - "--api.dashboard=true"
      - "--providers.docker=true"
      - "--providers.docker.exposedbydefault=false"
      - "--providers.file.directory=/certs"
      - "--providers.file.watch=true"
    ports:
      - "80:80"
      - "443:443"
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock:ro
      - ./certs:/certs:ro
    networks:
      - reverse
  networks:
    reverse:
      external: true
```

33Traefik anpassen

6. In allen Services wurde entrypoints=websecure gesetzt wie hier im Screenshot:

```
services:
  D Run Service
  uptime-kuma:
    image: louislam/uptime-kuma:1
    container_name: uptime-kuma
    labels:
      - "traefik.enable=true"
      - "traefik.docker.network=reverse"
      - "traefik.http.routers.kuma.rule=Host(`kuma.local`)"
      - "traefik.http.routers.kuma.entrypoints=websecure"
      - "traefik.http.routers.kuma.tls=true"
      - "traefik.http.services.kuma.loadbalancer.server.port=3001"
```

34Änderungen der services mit endpoint

7. Zertifikat im Browser akzeptieren sobald man auf die Webseiten verbindet.

Probleme

Anfangs wurden einige Dienste noch über HTTP (entrypoints=web) erreicht → alle auf websecure umgestellt

In manchen Browsern muss das Zertifikat manuell freigegeben werden

Ergebnis

Alle Dienste laufen jetzt über HTTPS mit dem self-signed Zertifikat

Verbindung ist verschlüsselt, Traefik verwendet lokal generiertes TLS

Dienst: UFW Firewall für Docker Netzwerke

Ziel

Ursprünglich war geplant, den Netzwerkverkehr zwischen den Docker-Netzwerken mit einer eigenen OPNsense VM zu regeln. Ich habe den Plan verworfen da das anschliessen mehrerer Docker Netzwerke an Virtuelle Netzwerke zu vielen Fehlern und Ausfällen führen könnte. Stattdessen nehme ich eine einfachere, aber effektive Lösung mit UFW auf dem Raspberry Pi.

Vorbereitung

1. Docker wurde so konfiguriert, dass es iptables nicht mehr benötigt indem iptables einfach ausgeschaltet wird im file **daemon.json** im Verzeichnis **/etc/docker/**

```
{
  "iptables": false
}
```

35 iptables ausschalten

2. Subnetze der Netzwerke mit **docker network inspect (Netzwerk)** Herausfinden:

- reverse: 172.22.0.0/16
- internal: 172.20.0.0/16

3. UFW aktiviert und Basiszugriffe erlaubt:

sudo ufw allow 2121/tcp für SSH

sudo ufw allow 80,443/tcp Für Traffic

sudo ufw enable

To	Action	From
---	-----	----
2121/tcp	ALLOW IN	Anywhere
80,443/tcp	ALLOW IN	Anywhere
172.22.0.0/16	DENY IN	172.20.0.0/16
2121/tcp (v6)	ALLOW IN	Anywhere (v6)
80,443/tcp (v6)	ALLOW IN	Anywhere (v6)

36 Firewall rules

4. Zugriff von Intern Netzwerk auf Reverse verbieten

sudo ufw deny from 172.20.0.0/16 to 172.22.0.0/16

Ergebnis

OPNsense wurde aus technischen Gründen verworfen

UFW steuert den Verkehr zwischen Docker-Netzwerken zuverlässig

reverse bleibt für Webdienste offen, internal ist geschützt

System bleibt wartbar, leicht erweiterbar und performant

Thema: Restic-Backup des gesamten Cloudcore-Projekts

Ziel

Das gesamte Verzeichnis /home/piCloudcore/cloudcore-project inklusive aller Docker-Dienste, Konfigurationsdateien und Skripte soll täglich automatisch gesichert werden. Alte Backups sollen automatisch auf 7 tägliche Snapshots begrenzt werden.

Vorbereitung

1. Restic installieren
sudo apt update
sudo apt install restic
2. Backup-Ziel erstellen
sudo mkdir -p /mnt/backup/docker
sudo chown -R piCloudcore:piCloudcore /mnt/backup/docker
3. Restic Repository initialisieren
export RESTIC_REPOSITORY=/mnt/backup/docker
export RESTIC_PASSWORD=geheimespasswort
restic init

Umsetzung

1. Backup-Script erstellen

Pfad: /home/piCloudcore/cloudcore-project/scripts/backup_containers.sh

```
#!/bin/bash
export RESTIC_REPOSITORY=/mnt/backup/docker
export RESTIC_PASSWORD=geheimespasswort

restic backup /home/piCloudcore/cloudcore-project

restic forget --keep-daily 7 --prune
```

37backup Script

2. Script ausführbar machen mit:
`chmod +x /home/piCloudcore/cloudcore-project/scripts/backup_containers.sh`
3. Cronjob einrichten mit **crontab -e** falls nach einer Nummer gefragt wird dann nano auswählen.
4. Das Einfügen das jeden Tag um 11:45 ein Backup gemacht wird: **45 11 * * ***
/home/piCloudcore/cloudcore-project/scripts/backup_containers.sh >>
/var/log/restic_backup.log 2>&1
5. Wenn das Backup geladen werden sollte also restored kann dieser command verwendet werden: **restic restore latest --target /tmp/restic-restore**

Ergebnis

Restic ist erfolgreich installiert und konfiguriert

Das gesamte Projektverzeichnis /home/piCloudcore/cloudcore-project wird täglich um 11:45 Uhr automatisch gesichert, sofern Pi läuft

Ältere Snapshots als 7 Tage werden, automatisch gelöscht

Das Backup Script ist modular und leicht erweiterbar

Thema: AdGuard Home Konfigurieren

Ziel

AdGuard Home soll als zentraler DNS-Server im Docker-Netzwerk reverse laufen und DNS-Anfragen aus dem lokalen Netzwerk beantworten. Der Container soll korrekt ins Internet routen und über Port 53 (TCP/UDP) erreichbar sein.

Vorbereitung

Container adguard lief bereits über Traefik auf Subdomain dns.local

Docker verwendet iptables: false Firewall Routing manuell

IP des Raspberry Pi: 10.80.4.210

Netzwerk: reverse mit Subnetz 172.22.0.0/16

UFW aktiviert

Umsetzung

1. docker-compose.yml um DNS-Ports erweitert im Adguard yaml file:

```
services:
  > Run Service
  adguard:
    image: adguard/adguardhome
    container_name: adguard
    restart: unless-stopped
    networks:
      - reverse
    ports:
      - "53:53/tcp"
      - "53:53/udp"
```

38Adguard yaml

2. UFW angepasst, damit DNS durchgelassen wird
`sudo ufw allow 53/tcp`
`sudo ufw allow 53/udp`
`sudo ufw reload`
3. Masquerade-Regel für NAT gesetzt erlaubt 172.22.0.0/16 den Zugriff auf Ethernet Interface eht0.
`sudo iptables -t nat -A POSTROUTING -s 172.22.0.0/16 -o eth0 -j MASQUERADE`

4. AdGuard DNS konfiguriert
DNS-Dienst aktiviert
Bind-Adresse: 0.0.0.0:53
Upstream-DNS-Server: 1.1.1.1, 8.8.8.8, 9.9.9.9
5. Forwarding-Regel für Docker und Host gesetzt
`sudo ufw route allow in on br-3bb6630096a9 out on eth0`

Ergebnis

AdGuard DNS antwortet erfolgreich auf Port 53 (IPv4 + IPv6)

UFW blockiert nicht mehr

Container hat vollständige Internetverbindung

Der Windows-Client kann AdGuard als DNS-Server nutzen

Setup ist stabil und produktionsbereit

Fazit

In den letzten 12 Tagen habe ich viel gelernt und geschafft. Ich habe meinen Raspberry Pi so eingerichtet, dass mehrere Dienste darauf sauber laufen, zum Beispiel ein Dashboard, Monitoring, Ad-Blocker und ein sicherer Zugang über HTTPS. Auch gelungen ist mir die saubere Docker-Struktur und die sichere Konfiguration wie bei dem SSH-Zugang nur mit Schlüssel. Auch wenn es zwischendurch Fehler gab, konnte ich diese jedes Mal lösen. Ausser beim Bookstack Dienst dieses Problem bleibt ungelöst. Das Projekt hat mir gezeigt, wie viel ich in kurzer Zeit erreichen kann, wenn ich Schritt für Schritt arbeite. Es war spannend viele neue Dinge zu lernen, mehr über Docker Container zu erfahren und zu lernen.

Abbildungsverzeichnis

1Pi Grundconfig im Imager.....	16
2Static IP setzen.....	16
3Docker Installation	17
4Projektsruktur.....	18
5Kuma Compose File	19
6Nginx Proxy einstellung.....	19
7Dashboard kuma erreichen.....	19
8Kuma Dashboard.....	20
9Kuma Netzwerk und basepath.....	21
10Reverse Proxy kuma einstellen	21
11Filebrowser yaml file.....	21
12Google Chrome error logs Kuma.....	22
13Location der Kuma Assets.....	22
14Traefik Compose file	23
15Lables um Traefik zu nutzen	24
16kuma mit lables.....	24
17Filebrowser mit lables.....	24
18Bookstack yaml file.....	26
19Bookstack richtig erstellt.....	26
20Inhalt der Hosts Datei auf Client.....	28
21Homer Dashboard yaml	30
22Homer Conf für dienste.....	30
23Dashboard von Homer Funktioniert.....	31
24Netdata yaml.....	32
25Netdata Monitoring.....	32
26Netzwerke	33
27Dockercontainer Start script	33
28SSHD config	34
29Fail2ban settigs	36
30Adguard yaml	37
31Hostdatei	37
32Homer eintrag	38
33Traefik anpassungen	40
34Änderungen der services mit entrypt.....	40
35Iptables ausschalten	41
36Firewall rules.....	41
37backup Script.....	42
38Adguard yaml	44

Anhang

Github Repository

[Abschlussarbeit Main](#)

[Abschlussarbeit Wiki](#)

Quellenangaben

https://doc.traefik.io/traefik/	Traefik
https://hub.docker.com/	Docker
https://github.com/louislam/uptime-kuma	Uptime kuma
https://github.com/filebrowser/filebrowser	Filebrowser
https://www.bookstackapp.com/	Bookstack
https://docs.netdata.cloud/	Netdata
https://github.com/AdguardTeam/AdGuardHome	AdGuardHome
https://restic.readthedocs.io/	Restic
https://www.fail2ban.org/	Fail2ban
https://ubuntu.com/server/docs/security-firewall	Firewall

Glossar

Container: Isolierte Umgebung zum Ausführen von Services.

Traefik: Moderner Reverse Proxy, um HTTP und HTTPS weiterzuleiten.

EntryPoint: Konfiguration in Traefik für Ports wie 80 oder 443

Self Signed Zertifikat: Vom Benutzer erstelltes TLS-Zertifikat ohne externes Zertifikat.

UFW: Uncomplicated Firewall ist ein Tool Zur Regelung von Netzwerktraffik.

Restic: Open-Source Backup Tool mit Datenverschlüsselung.

Relevante ki Chat Prompts

Infrastruktur:

„Wie binde ich ein self.crt und self.key in Traefik richtig ein?“

„Warum lädt Traefik immer noch das Default-Zertifikat?“

Fehleranalyse, Testing:

„Warum funktioniert HTTPS auf kuma.local nicht trotz korrekter Labels?“

„Was bedeutet: field not found, node: tls in Traefik?“

„Wie prüfe ich, ob Docker Volumes korrekt in den Container gemountet wurden?“

Sicherheit:

„Wie sichere ich meinen SSH-Zugang auf dem Raspberry Pi richtig ab?“

„Wie setze ich UFW-Regeln für Docker-Netzwerke sinnvoll ein?“

Backup, Automation:

„Wie begrenze ich Restic auf die letzten 7 Snapshots?“

DNS, Zugriff:

„Wie konfiguriere ich meine Hosts-Datei, um local-Domains erreichbar zu machen?“

Chat GPT wurde verwendet um diese Dokumentation zu Korrigieren.

Checkliste

Wichtige Hinweise	Dokumentation ist für Fachpersonen verständlich	X
	Eigenleistung und Unterstützungen sind klar deklariert	X
Allgemein	Kopfzeile Projektname Titel	X
	Fusszeile Datum, AutorIn, Seitenzahl	X
	Seitenlayout: Keine Überlappung Seitenränder, Quer/Hochformat	X
	Beschriftung der Bilder/Grafiken	X
	Einsatz von aussagekräftiger Grafiken (Netzwerkplan, DB Schema)	X
Titelblatt	Klar und übersichtlich gestaltet	X
	Überbegriff: Individuelle Abschlussprojekt BLJ	X
	Projektname (aussagekräftig / max 20 Zeichen)	X
	Name, Abgabedatum, Name der Lehrfirma	X
	Version des Dokuments	X
Inhaltsverzeichnis	Kapitel nummeriert (z.B.: 2.1; 2.1.1 usw.)	X
	Seitenzahlen korrekt angegeben	X
	Formatierung überprüft	X
Einleitung	Änderungstabelle/Versionierung (tabellenform)	X
	Aufgabenstellung und Projektbeschreibung	X
	Mögliche Risiken vor Projektbeginn	0
Planung	Terminplan (Gantt Diagramm oder Screenshot Github Project) vorhanden	X
	Entscheidungswege und Möglichkeiten (Entscheidungsmatrix)	X
Hauptteil	Detaillierte Beschreibung des Vorgehens und der Zwischenschritte	X
	Ergebnisse der Arbeit	X
	Entscheidungen sind formuliert	X
	Arbeitsjournal vorhanden	X
	Testplan/Testfälle mit Ergebnisse	X
	Persönliches Fazit	X
Anhang	Quellenangaben und Literaturverzeichnis vorhanden	X
	Glossar/Begriffserklärungen vorhanden (<i>Github / Dokument</i>)	X
	Bildverzeichnis vorhanden	X
	Programm-Code, Scripts, Foto-Dokumentation (<i>Github / Dokument</i>)	X
	Relevante KI-Chat-Prompts/Auszüge	X
	Testplan/Testfälle (optional)	0
	Ausgefüllte Checkliste	X
Code/Konfigurationen (Dateien bevorzugt auf Github)	Link zum Github Repository vorhanden	X
	Programm-Code folgt Clean-Code Richtlinien	X
	Wichtige Module, Klassen, Funktionen sind kommentiert	X
	Aussagekräftige Namen für Dateien, Klassen, Funktionen, DB Felder, ...	X
	Programm-Dateien Header mit Autor, Datum, Version, Beschreibung	X