

# Pi-Hole DNS Threat Intelligence-gestütztes NIDS

07.09.2025

CHRISTIAN SCHMID

# I. Management Summary

Dieses Projekt beschreibt den Aufbau einer zentralisierten, energieeffizienten und wartungsarmen Heim- bzw. Lab-Infrastruktur auf Basis eines Raspberry Pi, in der wesentliche Netzwerkdienste, Monitoring-Tools und Cloud-Integrationen in einer einzigen Plattform gebündelt werden.

Kern der Lösung ist ein Raspberry Pi, auf dem mehrere isolierte Docker-Container laufen. Diese beinhalten unter anderem:

- Pi-hole als DNS-Filter und Werbeblocker
- Unbound als sicherer, unabhängiger DNS-Resolver
- NGINX Proxy Manager zur zentralen Verwaltung und Absicherung des Zugriffs
- Uptime Kuma für Service- und Verfügbarkeitsüberwachung
- Dashy als zentrales Dashboard
- Watchtower für automatische Container-Updates
- AWS CLI zur direkten Interaktion mit Cloud-Diensten

Die Plattform ist mit AWS S3 und AWS Lambda verknüpft, um Logdaten und Backups automatisiert zu verarbeiten. Benachrichtigungen werden in Echtzeit per Discord Webhook an den Betreiber gesendet.

## Nutzen:

- Zentrale Verwaltung aller Dienste über eine einzige Hardwareplattform
- Automatisierte Wartung und Updates
- Verbesserte Netzwerksicherheit durch DNS-Filterung und lokale Auflösung
- Hohe Transparenz durch Monitoring und Echtzeit-Benachrichtigungen
- Skalierbarkeit für zusätzliche Dienste und Integrationen

Mit diesem Ansatz wird ein stabiler, flexibler und kosteneffizienter Betrieb erreicht, der sowohl für ambitionierte Home-Lab-Umgebungen als auch für kleine Unternehmen geeignet ist.

## II. Inhaltsverzeichnis

I. Management Summary .....	I
II. Inhaltsverzeichnis .....	II
1. Problem, Fragestellung, Vision.....	1
2. Stand der Praxis .....	2
3. Ideen und Konzepte.....	3
4. Methodik .....	5
5. Realisierung.....	6
5.1 Komponenten .....	6
5.1 Installationen .....	6
5.1.1 Raspberry Pi 4 4GB .....	6
5.1.2 Docker .....	7
5.1.3 Pi-Hole .....	8
5.1.4 Unbound .....	9
5.1.5 Uptime Kuma .....	10
5.1.6 Dashy .....	11
5.1.7 Watchtower .....	12
5.1.8 Nginx Proxy Manager.....	12
5.1.9 AWS CLI .....	14
5.2 AWS .....	15
5.2.1 IAM-User (pi-log-uploader).....	15
5.2.2 S3-Bucket .....	17
5.2.3 AWS Lambda .....	20
5.2.4 IAM Policy.....	21
5.3 Discord Webhook .....	25
5.4 N8n .....	26
6. Evaluation und Validation .....	28
7. Ausblick .....	29
8. Anhänge .....	30
8.1 Links.....	30
8.2 Skripte .....	31
8.3 Abbildungsverzeichnis .....	34
8.4 Tabellenverzeichnis .....	35

# 1. Problem, Fragestellung, Vision

## **Problem:**

In Heim- und Laborumgebungen fehlt oft eine zentrale, übersichtliche und automatisierte Infrastruktur für Netzwerküberwachung, DNS-Filterung, Service-Management und Cloud-Integration. Viele Systeme sind verteilt, schwer zugänglich und erfordern manuelle Wartung, was zu Sicherheitslücken, unnötigem Zeitaufwand und unklaren Abhängigkeiten führt.

## **Fragestellung:**

Wie kann eine kompakte, energieeffiziente Plattform aufgebaut werden, die zentrale Netzwerkdienste, Monitoring und Cloud-Anbindung integriert?

Wie lässt sich sicherstellen, dass diese Dienste zuverlässig, wartungsarm und jederzeit erreichbar sind – sowohl lokal als auch aus der Cloud?

Wie kann die Lösung so gestaltet werden, dass sie mit minimalem Hardwareeinsatz auch erweiterbar bleibt?

## **Vision:**

Eine vollständig integrierte Raspberry-Pi-basierte Plattform, auf der alle wichtigen Netzwerk- und Monitoringdienste in Docker-Containern zentral betrieben werden.

Die Plattform soll:

- als DNS-Resolver und Werbeblocker (Pi-hole + Unbound) fungieren
- über ein zentrales Dashboard (Dashy) und Monitoring (Uptime Kuma) die Verfügbarkeit aller Dienste überwachen
- Cloud-Speicher (AWS S3) und Automatisierungen (AWS Lambda) einbinden
- Benachrichtigungen in Echtzeit per Discord Webhook senden
- sich automatisch aktualisieren (Watchtower) und einfach über NGINX Proxy zugänglich sein

Ziel ist ein stabiles, selbstwartendes und erweiterbares System, das sowohl für Home-Lab- als auch für kleine Unternehmensumgebungen einsetzbar ist.

## 2. Stand der Praxis

### Technologische Basis:

- Raspberry Pi als stromsparender, multifunktionaler Server
- Docker zur isolierten Ausführung und einfachen Verwaltung mehrerer Dienste
- Pi-hole zur DNS-Filterung und Werbung-/Tracking-Blockierung
- Unbound als rekursiver DNS-Resolver für mehr Sicherheit und Unabhängigkeit von externen DNS-Anbietern
- NGINX Proxy Manager für Zugriffsbündelung und SSL-Management
- Uptime Kuma als Open-Source-Monitoringlösung
- Dashy als konfigurierbares Service-Dashboard
- Watchtower zur automatischen Containeraktualisierung
- AWS CLI für direkte Interaktion mit AWS-Diensten
- N8n als konfigurierbares Automatisierungstool (GUI)

### Cloud-Integration:

- AWS S3 als zentraler Speicher für Logs, Backups oder Konfigurationsdaten
- AWS Lambda zur Event-basierten Datenverarbeitung und Automatisierung
- Discord Webhooks für Benachrichtigungen in Echtzeit

### Stand der Praxis:

Solche Raspberry-Pi-gestützten All-in-One-Lösungen sind im Home-Lab-Bereich verbreitet, werden aber oft nur teilweise umgesetzt. Häufig fehlt die Kombination aus:

1. DNS-Sicherheit
2. Service-Monitoring
3. Automatisierter Cloud-Anbindung
4. Echtzeit-Benachrichtigung
5. Selbstwartung

Die hier dargestellte Architektur geht über einfache Pi-hole-Installationen hinaus, indem sie mehrere Services unter einem gemeinsamen Proxy bündelt, Automatisierungen integriert und Cloud-Funktionalitäten einbindet.

### 3. Ideen und Konzepte

Um richtig aufzuzeigen, wie das Homelab aufgebaut ist, wird ein (erweiterbarer) Plan erstellt. Dies wurde mithilfe einer Figma-Vorlage gemacht.

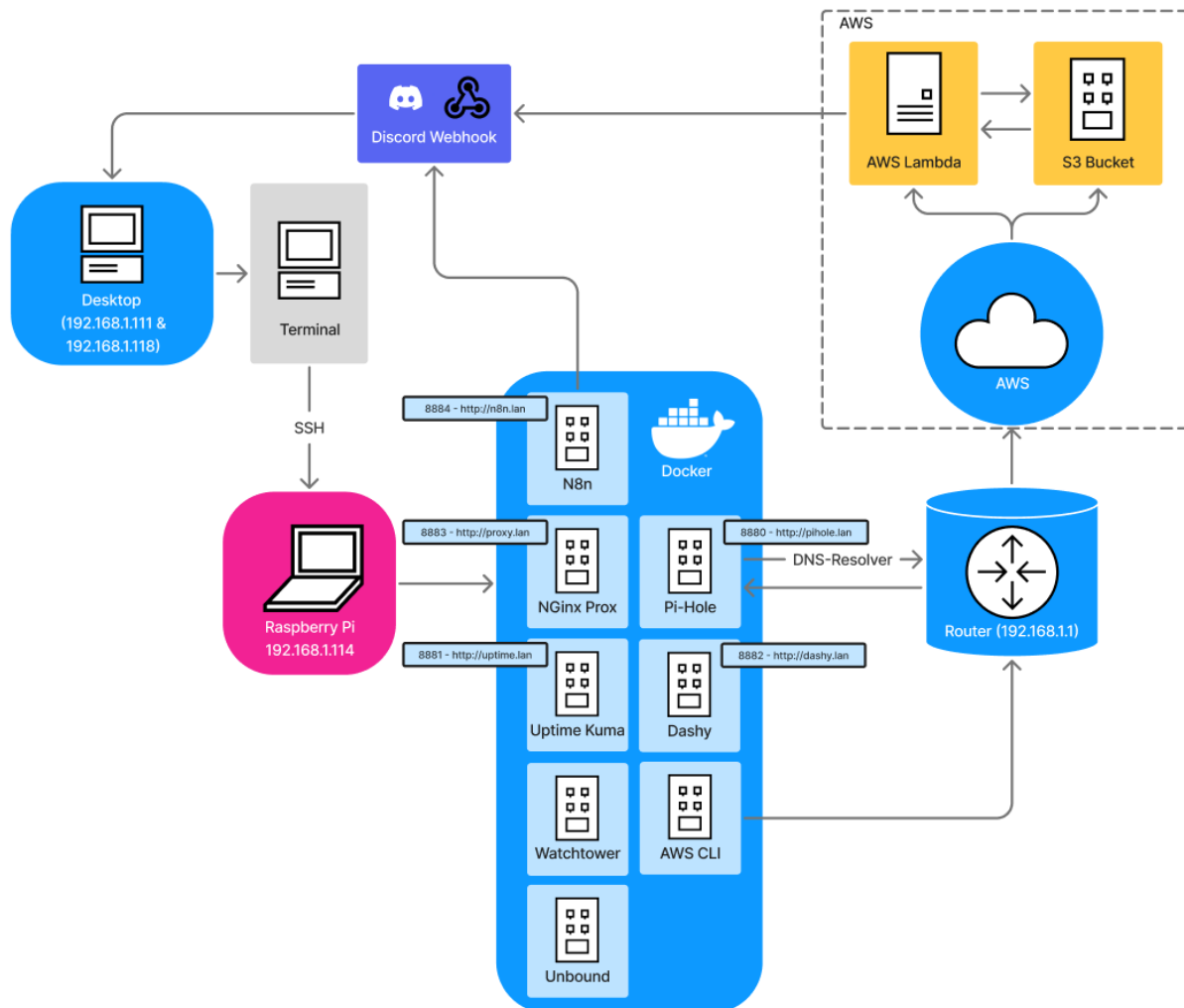


Abbildung 1: Konzeptplan

#### Desktop-Computer

- Zwei IPs: 192.168.1.111 & 192.168.1.118
- Dient als Hauptzugriffspunkt
- Verbindung via Terminal (SSH) zum Raspberry Pi

#### Raspberry Pi (192.168.1.114)

- Läuft als zentrale Plattform für verschiedene Dienste in Docker-Containern.
- Enthaltene Container:
  - **Pi-hole** – DNS-Filter / Werbeblocker, erreichbar unter `http://pihole.local` oder Port 8880.
  - **Uptime Kuma** – Monitoring-Tool, erreichbar unter `http://uptime.local` oder Port 8881.
  - **Dashy** – Dashboard für Dienste, erreichbar unter `http://dashy.local` Port oder 8882.
  - **NGINX Proxy** – Reverse Proxy, erreichbar unter `http://proxy.local` oder Port 8883.

- **Unbound** – Lokaler DNS-Resolver.
- **Watchtower** – Automatische Updates der Docker-Container.
- **AWS CLI** – Zugriff auf AWS-Dienste über Kommandozeile.
- **N8n** – Automatisierungstool für Cybernews, erreichbar unter <http://n8n.local> oder Port 8884.

### **Router (192.168.1.1)**

- Leitet DNS-Anfragen an den Pi-hole (DNS-Resolver) weiter.
- Dient als Brücke zwischen Netzwerk und

### **AWS-Integration**

- AWS CLI auf dem Raspberry Pi interagiert mit AWS S3 (Speicher) und AWS Lambda (Serverless-Funktionen).
- Daten/Logs werden z. B. vom Raspberry Pi hochgeladen.
- AWS Lambda verarbeitet Daten und kann Discord Webhooks auslösen.

### **Discord Webhook**

- Sorgt für Benachrichtigungen (z. B. Statusmeldungen oder Alerts) direkt in einen Discord-Kanal.

Die Struktur ist in keinstenweise fertig, es können immer wieder Änderungen vorkommen und der Plan kann erweitert werden.

## 4. Methodik

Die Umsetzung des Projekts erfolgte in mehreren aufeinander abgestimmten Schritten. Grundlage bildete die Auswahl des Raspberry Pi als stromsparende Hardwareplattform. Darauf wurde ein Linux-Basisbetriebssystem installiert und für den stabilen Dauerbetrieb konfiguriert.

Im Anschluss wurde Docker eingerichtet, um die einzelnen Dienste isoliert und unabhängig voneinander in Containern zu betreiben. Die Container wurden über Docker-Compose definiert, sodass sich die gesamte Infrastruktur reproduzierbar und leicht verwalten lässt.

Dabei wurden die Dienste schrittweise integriert und konfiguriert.

Parallel erfolgte die Einbindung von AWS S3 für die Speicherung von Logs und Backups sowie AWS Lambda für eventbasierte Prozesse. Für die Benachrichtigung wurde ein Discord Webhook implementiert.

Die Systemintegration wurde iterativ getestet: Nach jedem Schritt wurden Funktion, Erreichbarkeit und Interaktion der Dienste geprüft. Abschliessend wurde ein Gesamttest durchgeführt, bei dem Szenarien wie DNS-Auflösung, Service-Ausfälle, automatisierte Backups und Benachrichtigungen simuliert wurden.



## 5. Realisierung

### 5.1 Komponenten

Für dieses Projekt wurden folgende Komponenten und Dienste verwendet:

#### Raspberry Pi 4 4GB

- Docker (Plattform, um andere Dienste/Applikationen via Container zu starten)
- PiHole (DNS-Filter/Adblocker auf L3-Ebene (Netzwerk))
- Unbound (Eigener DNS-Resolver)
- Uptime-Kuma (Monitoring-Tool für Dienste & Server)
- Dashy (All-In-One-Dashboard)
- Nginx Proxy Manager (Auflösen von IPs)
- N8n (Automation)

#### Amazon AWS

- AWS S3 Bucket (Speichern der Logs, stündlich)
- AWS Lambda (Ausführen von Funktionen – Vergleich mit IoC-Listen)

#### Sonstiges

- Discord (Alarmierung)

### 5.1 Installationen

#### 5.1.1 Raspberry Pi 4 4GB

Aufgesetzt wurde der Raspberry mithilfe dem Raspberry Pi Imager v1.8.5 dem Raspberry Pi Lite OS 64-Bit, da es sich um ein ARM 64-Bit System handelt. Dies wurde auf einer Mikro-SD Karte frisch installiert.

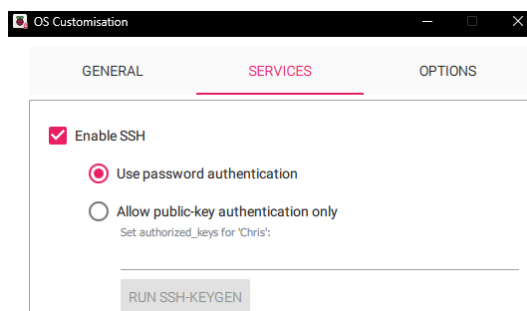


Abbildung 2: Installation - Raspberry (OS Customisation)

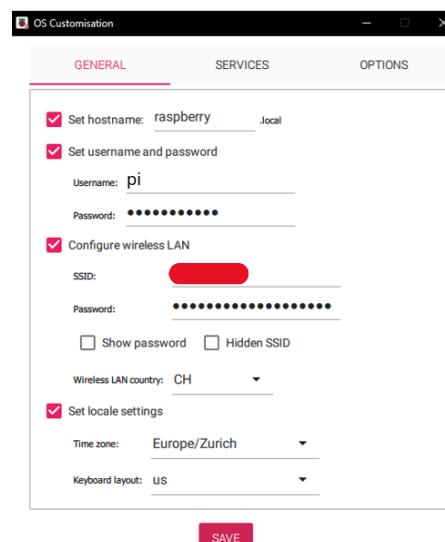


Abbildung 3: Installation - Raspberry (Konfiguration)

SSH wurde ebenso aktiviert, um einfachen Zugriff per Desktop zu haben via der Raspberry-IP. Als Best-Practice sollte SSH mit Passwort deaktiviert werden und durch SSH-Keys gesetzt werden, darauf wurde aber einfachheitshalber verzichtet.

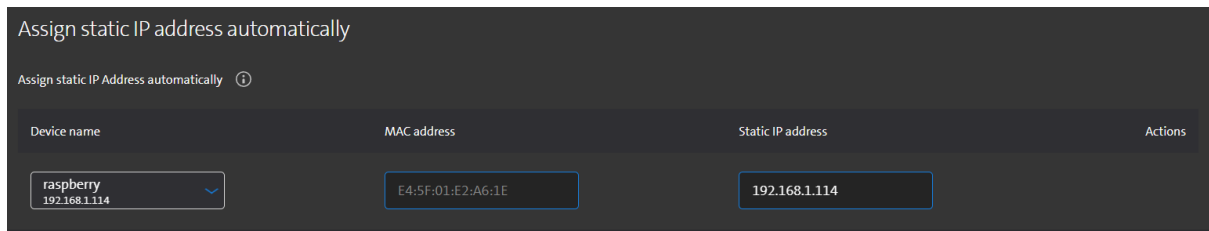


Abbildung 4: Raspberry - statische IP setzen

Damit die IP des Rasperrys stets die gleiche bleibt, wurde eine statische IP (192.168.1.114) auf dem Router (=Swisscom) gesetzt:

### 5.1.2 Docker

Nachdem die Mikro-SD Karte eingeführt und das System aktualisiert wurde, war Docker an der Reihe. Die Installation von Docker Engine, CLI & Daemon wurde mittels der Docker Bedienungsanleitung installiert:

```
curl -fsSL https://get.docker.com -o get-docker.sh
sudo sh get-docker.sh
```

➤ Anschliessend muss der Nutzer zur Docker Gruppe hinzugefügt werden:

```
sudo usermod -aG docker $USER
```

Nach der Installation muss geprüft werden, ob Docker installiert ist:

```
pi@raspberry:~$ docker version
Client: Docker Engine - Community
 Version:      28.1.1
 API version:  1.49
 Go version:   go1.23.8
 Built:        Fri Apr 18 09:52:08 2025
 OS/Arch:      linux/arm64
 Context:      default

Server: Docker Engine - Community
 Engine:
  Version:      28.1.1
  API version:  1.49 (minimum version 1.24)
  Go version:   go1.23.8
  Built:        Fri Apr 18 09:52:08 2025
  OS/Arch:      linux/arm64
  Experimental: false
 containerd:
  Version:      1.7.27
  runc:
  Version:      1.2.5
  docker-init:
  Version:      0.19.0
```

Abbildung 5: Docker - Version

### 5.1.3 Pi-Hole

Das Pi-Hole ist eine Software, welches auf Netzwerk-Level Werbung blockieren und überwachen kann. Durch bereitgestellte Listen wird eine grosse Menge von Werbung und schädlichen Webseiten blockiert. Zusätzlich stellt das Pi-Hole auch einen optionalen DHCP-Server dar, welcher für dieses Projekt aber nicht benutzt wurde.

Die Installation wurde mit der Pi-Hole Dokumentation auf GitHub gemacht.

Ein Ordner und das `docker-compose.yml` muss erstellt werden. Darin kann das Standardconfig eingefügt werden.

```
mkdir -p ~/pihole && cd ~/pihole
vim docker-compose.yml
```

- Durch das "restart: unless-stopped" startet Pi-Hole automatisch, wenn der Raspberry eingeschaltet wird.

Durch `docker compose up -d` wird der Container gestartet.

```
pi@raspberrypi:~/pihole $ vim docker-compose.yml
pi@raspberrypi:~/pihole $ docker compose up -d
[+] Running 1/12
  ! pihole [..#..] Pulling
    d13a3fff434d Downloading [=====] 1.396MB/... 3.9s
    d2e94adaca89 Downloading [=====] 3.582MB/... 3.9s
    4f4fb70ef54 Download complete 0.5s
    b8f82a2297b3 Downloading [=====] 409.6kB/... 3.9s
    171fdabd6c87 Waiting 3.9s
    fc9a60886657 Waiting 3.9s
    e2aa21b1fefe Waiting 3.9s
    c9acb12deaa4 Waiting 3.9s
    ce78ed14825f Waiting 3.9s
    d228e9827753 Waiting 3.9s
    67708873166b Waiting 3.9s
```

Abbildung 6: Pi-Hole - Container start

Nun ist Pi-hole mit dem Browser standardmässig mit der IP des Raspberrys erreichbar:

Username: admin

Passwort (Standard, aber nun entsprechend abgeändert!): 'correct horse battery staple'.

Eine Blockliste ist von Pi-Hole Gravity schon vorinstalliert, es handelt sich um eine Liste von StevenBlack.

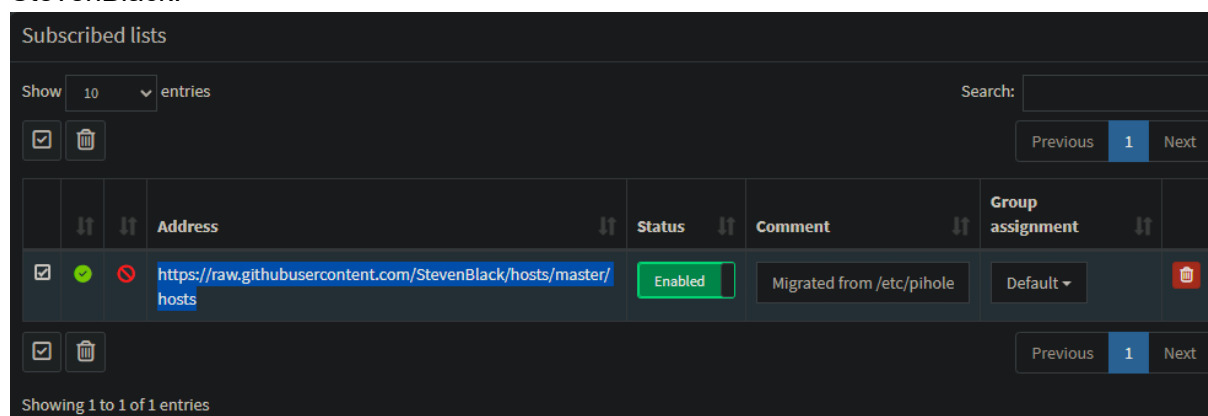


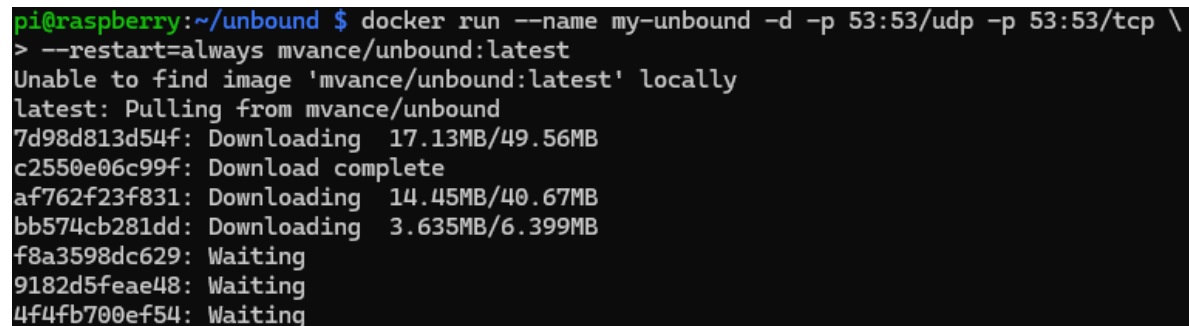
Abbildung 7: Pi-Hole - Blockliste

### 5.1.4 Unbound

Unbound ist ein DNS-Resolver, welcher rekursiv arbeitet. Wenn Pi-Hole die IP-Adresse nicht weiss, muss auf Cloudflare/Google zurückgegriffen werden (8.8.8.8/1.1.1.1). Mithilfe Unbound wird bei einem Ausfall von Pi-Hole ein eigener DNS-Server entstehen. Somit werden keine Spuren bei öffentlichen DNS-Servern hinterlassen.

Auch hier wurde mit der Unbound GitHub-Dokumentation gearbeitet.

```
Mittels docker run -name my-unbound -d -p 53:53/udp -p 53:53/tcp \
--restart=always mvance/unbound:latest
```



```
pi@raspberrypi:~/unbound $ docker run --name my-unbound -d -p 53:53/udp -p 53:53/tcp \
> --restart=always mvance/unbound:latest
Unable to find image 'mvance/unbound:latest' locally
latest: Pulling from mvance/unbound
7d98d813d54f: Downloading 17.13MB/49.56MB
c2550e06c99f: Download complete
af762f23f831: Downloading 14.45MB/40.67MB
bb574cb281dd: Downloading 3.635MB/6.399MB
f8a3598dc629: Waiting
9182d5feae48: Waiting
4f4fb700ef54: Waiting
```

Abbildung 8: Unbound - Installation

Da kein Webserver oder ähnliches existiert, gibt es auch kein Webinterface. Unbound läuft somit komplett im Hintergrund.

### 5.1.5 Uptime Kuma

Uptime Kuma ist ein „Watcher“, welcher regelmässig jegliche Dienste (Pi-Hole, Unbound, Router, HTTP/S Server etc.) überwacht. Genau wie bei den anderen beiden Diensten, wird auch hier mit der GitHub-Dokumentation gearbeitet. Es wurde lediglich der Port angepasst. Nach der Installation ist Uptime Kuma mit der Raspberry IP und dem entsprechenden Port aufrufbar. (<http://192.168.1.114:8881>)

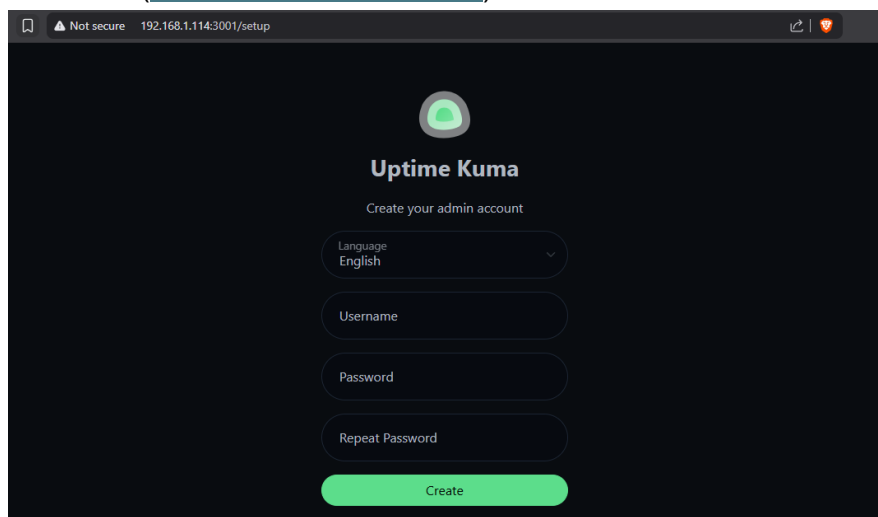


Abbildung 9: Uptime Kuma - Admin Account

Nach dem Aufruf muss ein Admin account erstellt werden. Standardmässig wurde hier der Benutzername admin gewählt.

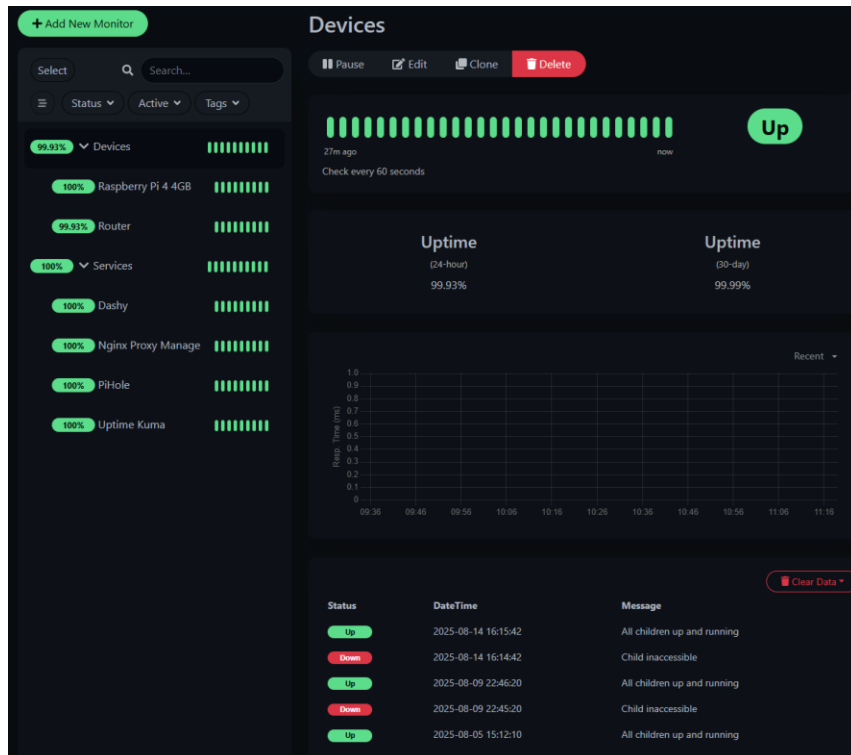


Abbildung 10: Uptime Kuma - Startseite

Es wurden danach via „Add New Monitor“ Services und Devices hinzugefügt.

## 5.1.6 Dashy

Dashy ist ein All-In-One Dienst, welcher dynamisch Inhalte von API-Services anzeigen kann. Es eignet sich super, um alles auf einem Blick zu haben. Für die Installation wurde mit der GitHub-Dokumentation von Dashy gearbeitet. Nach der Installation ist Dashy über den angegebenen Port erreichbar (in meinem Fall <http://192.168.1.114:8883>).



Abbildung 11: Dashy - Startseite

Die Startseite von Dashy sieht recht leer aus. Durch ein paar Anpassungen kann es sehr lebhaft aussehen. Neben den gängigen Services wurden noch die meist besuchten Seiten hinzugefügt.

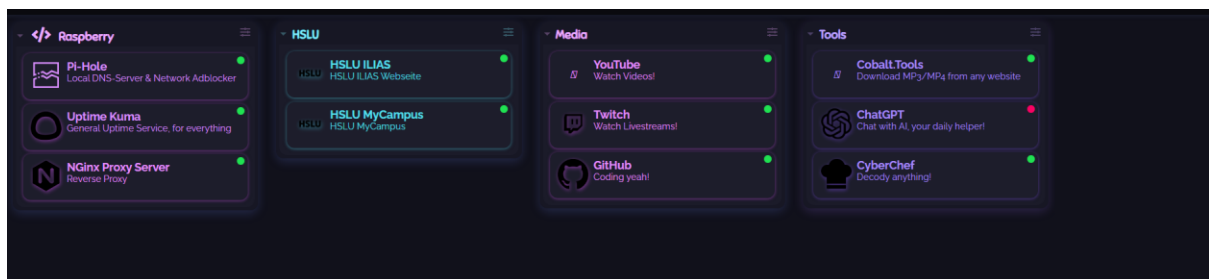


Abbildung 12: Dashy - Startseite (Modifiziert)

### 5.1.7 Watchtower

Watchtower ist ein Dienst, um die laufenden Container stets up-to-date zu haben. Es zieht alle 24 Stunden die aktuellste Datei herunter und installiert es direkt. Gearbeitet wurde auch hier mit der Watchtower GitHub-Dokumentation. Dieser Dienst läuft wie Unbound im Hintergrund. Somit hat es keinen Webservice.

### 5.1.8 Nginx Proxy Manager

**Disclaimer: Anstatt „.local“ wurde „.lan“ verwendet, da der Swisscom Router zuerst in die mDNS eingreift.**

Der Proxy Manager ist unter anderem für das einfache umwandeln Adressen. Mithilfe des Pi-Holes könnte man auch einzelne Adressen setzen, diese sind jedoch nur pro Device gedacht. Bei mehreren Dienste (= mehreren Ports) braucht es daher eine Reverse Proxy. Hierfür ist der Nginx Proxy Manager.

Der Nginx Proxy Manager wurde über Docker über die GitHub-Dokumentation von Nginx Proxy Manager installiert.

Nach dem Setup ist das NPM-Adminpanel für den Proxy Manager unter dem angegebenen Port erreichbar. Die Standardcredentials sind [admin@example.com](mailto:admin@example.com) & changeme. Die Credentials wurden angepasst.

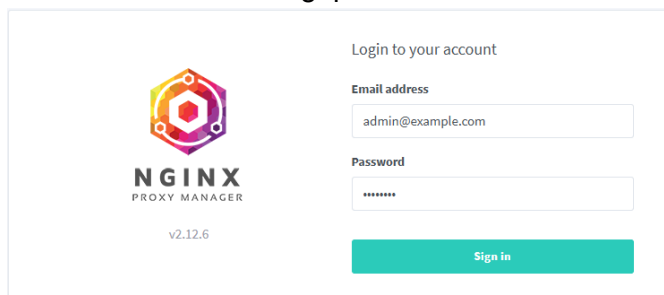
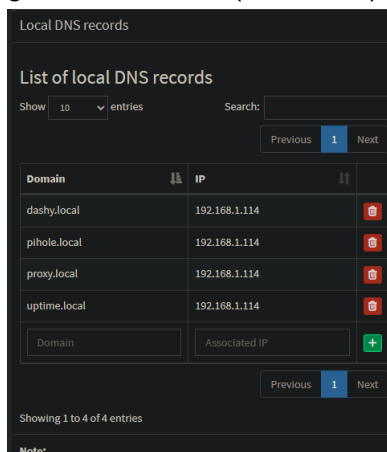


Abbildung 13: Nginx Proxy Manager - Anmelden

Das eigentliche Auflösen der IPs muss nachwievor über den DNS-Server (dem Pi-Hole) gemacht werden (siehe Bild). Dabei wurden folgende Einträge gemacht:



Dashy → dashy.local (192.168.1.114)  
PiHole → pihole.local (192.168.1.114)  
Nginx Proxy → proxy.local (192.168.1.114)  
Uptime Kuma → uptime.local (192.168.1.114)

➔ **NEU: überall .lan!**

Abbildung 14: Pi-Hole - local DNS records

Auf dem Nginx Proxy Manager wurden dann die Einträge weitergeleitet. Dies muss via Hosts → Proxy Hosts gemacht werden.

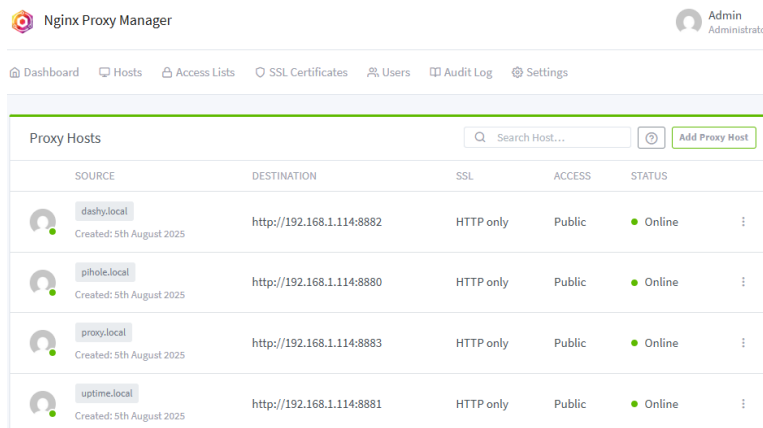


Abbildung 15: Nginx Proxy Manager - Proxy Einträge

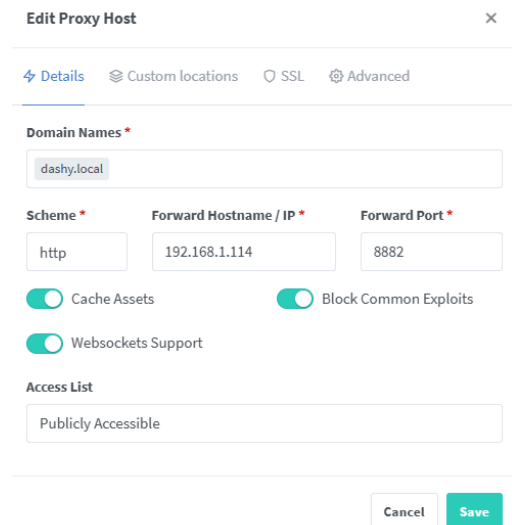


Abbildung 16: Nginx Proxy Manager - Einzelner Proxy Eintrag

Die Optionen Cache Assets, Block Common Exploits und Websockets Support wurde Best-Practice-Halber aktiviert. Nun können die einzelnen Dienste durch normale, einfache URLs aufgerufen werden.

Dienst (Port)	Ansprechbar via
PiHole (8880)	<a href="http://pihole.local">http://pihole.local</a> (neu <a href="http://pihole.lan">http://pihole.lan</a> )
Uptime Kuma (8881)	<a href="http://uptime.local">http://uptime.local</a> (neu <a href="http://uptime.lan">http://uptime.lan</a> )
Dashy (8882)	<a href="http://dashy.local">http://dashy.local</a> (neu <a href="http://dashy.lan">http://dashy.lan</a> )
Nginx Proxy Manager (8883)	<a href="http://proxy.local">http://proxy.local</a> (neu <a href="http://proxy.lan">http://proxy.lan</a> )
N8n (8884)	<a href="http://n8n.local">http://n8n.local</a> (neu <a href="http://n8n.lan">http://n8n.lan</a> )

Tabelle 1: Auflösung Dienste



### 5.1.9 AWS CLI

Das AWS CLI (Amazon Web Services Client Interface) ist für die nahtlose Kommunikation zwischen dem Raspberry Pi und mit der Cloud. In diesem Fall ist es für das nahtlose Hochladen der Pi-Hole Logs. Die Installation wurde mit der GitHub-Dokumentation gemacht.

Commands:

```
sudo apt update
curl "https://awscli.amazonaws.com/awscli-exe-linux-aarch64.zip" -o "awscliv2.zip"
unzip awscliv2.zip
sudo ./aws/install
aws configure
```

Nach dem Installieren musste mit `aws configure` den AWS Access Key und den AWS Access Secret Key sowie die Region angeben. Dieser wird im nächsten Schritt erstellt.

## 5.2 AWS

### 5.2.1 IAM-User (pi-log-uploader)

Damit der Raspberry Pi die Pi-Hole Logs hochladen kann, muss ein User erstellt werden. Dieser User muss für das Hochladen der Logs berechtigt sein. Kurz gesagt heisst dies, dass man einen IAM User erstellen sollte. Da sich das Interface auf AWS stets ändert, sehen die Schritte ungefähr so aus:

1. Burgermenü → All Services → Access Management → IAM → Users → Create User
2. Optionen: Siehe Screenshot (Username, Passwort etc.)

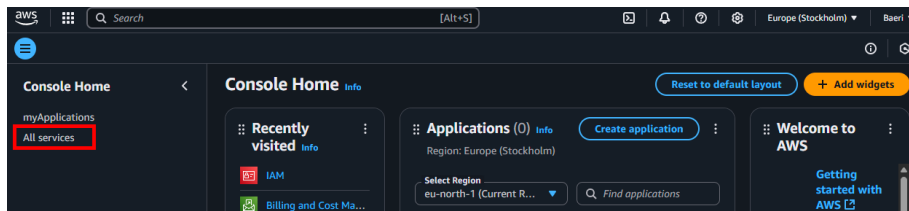


Abbildung 19: AWS - All Services:

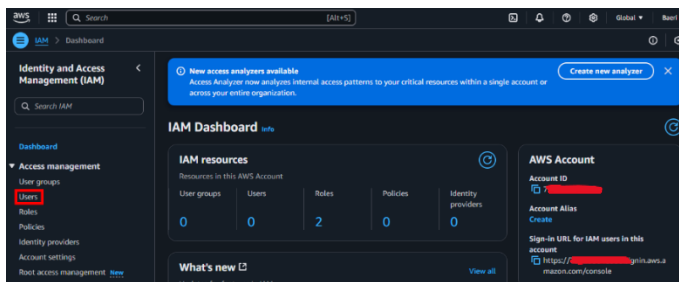


Abbildung 17: AWS - IAM Dashboard

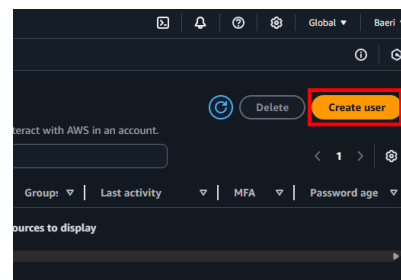


Abbildung 18: AWS - IAM Create User



Abbildung 20: AWS - pi-log-uploader IAM Optionen

Die Berechtigungen für den User wurden extern via einer Policy hinzugefügt. Dabei wurde sich für S3 entschieden, da es günstiger als zB. Cloudwatch ist.

AWS besitzt über 1'500 verschiedene Policies. Ich habe jedoch keine Policy spezifisch für diesen Anwendungsfall gefunden. Es wurde ein JSON welches den S3-Bucket erlaubt, die einzelnen Logs zu uploaden hochgeladen (*s3-upload-pihole-uploader.json*):

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:PutObject",
        "s3:PutObjectAcl"
      ],
      "Resource": "arn:aws:s3:::pi-logs-bucket/*"
    }
  ]
}
```

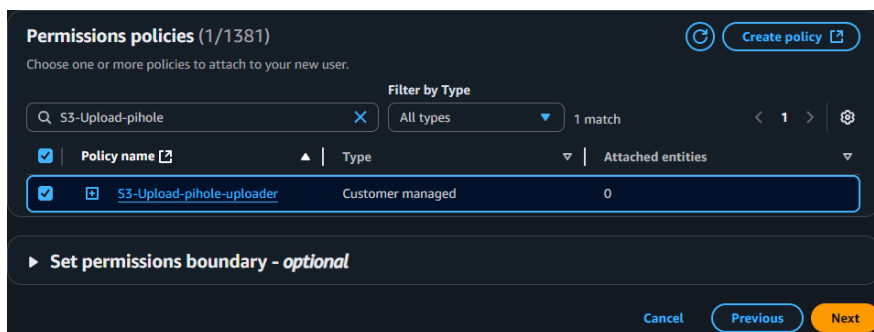
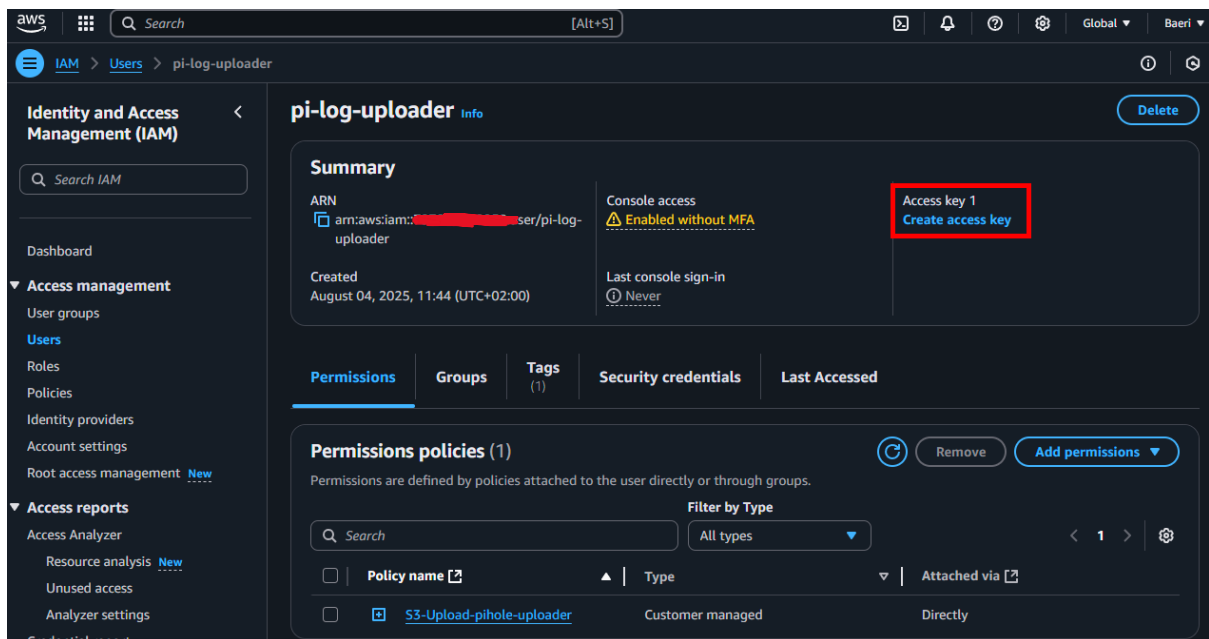


Abbildung 21: AWS - pi-log-uploader Permissions

Anschliessend muss noch ein AWS Access Key & AWS Secret Access Key erstellt werden. Durch die beiden Access Keys wird ein „Login“ erstellt für die AWS CLI.



Via `aws configure` auf dem Raspberry Pi und den AWS Keys kann man sich nun verbinden.

## 5.2.2 S3-Bucket

Damit die Logs gespeichert werden können, muss ein Storage auf der AWS Cloud erstellt werden. Die Lösung – ein AWS S3-Bucket. Der Vorgang ist: Burgermenü → All Services → Storage → S3 → Create Bucket

**Create bucket** [Info](#)

Buckets are containers for data stored in S3.

**General configuration**

**AWS Region**  
Europe (Stockholm) eu-north-1

**Bucket type** [Info](#)

☒ **General purpose**  
Recommended for most use cases and access patterns. General purpose buckets are the original S3 bucket type. They allow a mix of storage classes that redundantly store objects across multiple Availability Zones.

☐ **Directory**  
Recommended for low-latency use cases. These buckets use only the S3 Express One Zone storage class, which provides faster processing of data within a single Availability Zone.

**Bucket name** [Info](#)  
pi-logs-bucket-baeri

Bucket names must be 3 to 63 characters and unique within the global namespace. Bucket names must also begin and end with a letter or number. Valid characters are a-z, 0-9, periods (.), and hyphens (-). [Learn More](#) [?](#)

**Copy settings from existing bucket - optional**  
Only the bucket settings in the following configuration are copied.

[Choose bucket](#)

Format: s3://bucket/prefix

**Object Ownership** [Info](#)  
Control ownership of objects written to this bucket from other AWS accounts and the use of access control lists (ACLs). Object ownership determines who can specify access to objects.

☒ **ACLs disabled (recommended)**  
All objects in this bucket are owned by this account. Access to this bucket and its objects is specified using only policies.

☐ **ACLs enabled**  
Objects in this bucket can be owned by other AWS accounts. Access to this bucket and its objects can be specified using ACLs.

**Object Ownership**  
Bucket owner enforced

Abbildung 22: AWS - S3 Bucket Konfigurationen

Da die PiHole Logs jeweils immer in `/home/pi/pihole/etc-pihole/logs/pihole.log` gespeichert werden, können diese nun mittels Cronjob regelmässig auf AWS gesendet werden. Der «Base-Command» würde wiefolgt aussehen:

```
aws s3 cp /home/pi/pihole/etc-pihole/logs/pihole.log s3://xxxxxxxxxxx/pihole-$(date +%F-%H).log
```

Dieser Command sendet aber immer das gesamte Log auf den S3 Bucket, was sehr schnell sehr lang wird. Daher wurde ein Skript verwendet, was inkrementell neue Einträge hochlädt. Dieses Skript kann mit Cronjob kontinuierlich ausgeführt werden. Einfachheitshalber habe ich das Skript in den Ordner `/etc/cron.hourly` verschoben. Somit wird es jede Stunde ausgeführt.

Das Skript (*upload-pihole-logs*):

```
#!/bin/bash

LOG="/home/pi/pihole/etc-pihole/logs/pihole.log"
STATE="/tmp/pihole_log_last_pos"
BUCKET="s3://pi-logs-xxxxxxxxxxxx/pihole"
DATE=$(date +%F)
TIME=$(date +%H%M)
OUTFILE="/tmp/pihole-${DATE}-${TIME}.log"

POS=$(cat "$STATE" 2>/dev/null || echo 0)
SIZE=$(stat -c%s "$LOG")

[ "$POS" -gt "$SIZE" ] && POS=0

if [ "$SIZE" -gt "$POS" ]; then
    dd if="$LOG" bs=1 skip="$POS" of="$OUTFILE" status=none
    aws s3 cp "$OUTFILE" "$BUCKET/$DATE/pihole-${DATE}-${TIME}.log"
    echo "$SIZE" > "$STATE"
    rm "$OUTFILE"
fi
```

Die Struktur auf dem S3 Bucket sieht nun wie folgt aus:

```
s3://pi-logs-bucket-baeri/pihole/
├── 2025-08-04/
│   ├── pihole-2025-08-04-1600.log
│   └── pihole-2025-08-04-1700.log
├── 2025-08-05/
└── ...
```

Damit es nun nicht unendliche Logs gibt und der S3 Bucket eine bescheidene Grösse hat, wird der Bucket noch mit einer Lifecycle Rule versehen. Dies bedeutet, dass nach x Tagen, Stunden oder Minuten die ältesten Dateien gelöscht werden.

Gemacht wurde das auf dem S3 Bucket unter Management → Lifecycle Rules

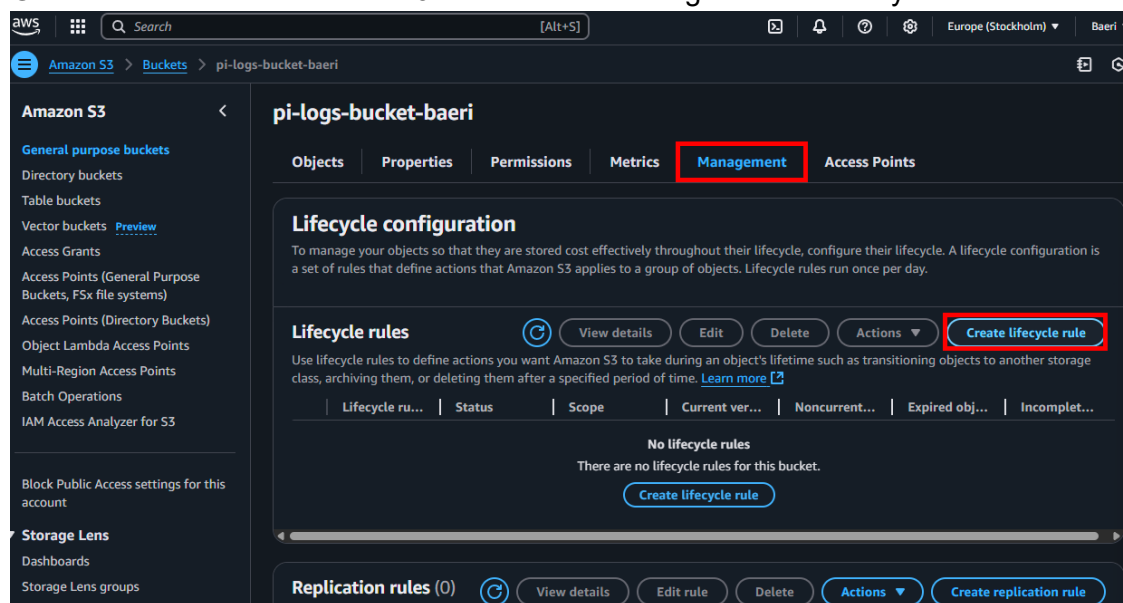


Abbildung 23: AWS - S3 Bucket Lifecycle Management

Die Einstellungen sind entsprechend selbsterklärend, da nach 7 Tagen die Einträge gelöscht werden:

**Create lifecycle rule** [Info](#)

**Lifecycle rule configuration**

**Lifecycle rule name**

DeleteOldPiHoleLogs

Up to 255 characters

**Choose a rule scope**

☒ Limit the scope of this rule using one or more filters

☐ Apply to all objects in the bucket

**Filter type**

You can filter objects by prefix, object tags, object size, or whatever combination suits your usecase.

**Prefix**

Add filter to limit the scope of this rule to a single prefix.

pihole/

Don't include the bucket name in the prefix. Using certain characters in key names can cause problems with some applications and protocols. [Learn more](#)

**Object tags**

You can limit the scope of this rule to the key/value pairs added below.

[Add tag](#)

**Object size**

You can limit the scope of this rule to apply to objects based on their size. [Learn more](#)

☐ Specify minimum object size

☐ Specify maximum object size

**Lifecycle rule actions**

Choose the actions you want this rule to perform.

☐ Transition current versions of objects between storage classes  
This action will move current versions.

☐ Transition noncurrent versions of objects between storage classes  
This action will move noncurrent versions.

☒ Expire current versions of objects

☐ Permanently delete noncurrent versions of objects

☐ Delete expired object delete markers or incomplete multipart uploads  
These actions are not supported when filtering by object tags or object size.

**Expire current versions of objects**

For version-enabled buckets, Amazon S3 adds a delete marker and the current version of an object is retained as a noncurrent version. For non-versioned buckets, Amazon S3 permanently removes the object. [Learn more](#)

**Days after object creation**

7

Abbildung 24: AWS - S3 Bucket Lifecycle Rule

### Exception Rules (AWS Lambda)

Es wird eine Exception Rule definiert, welche dann den Zugriff auf AWS Lambda explizit erlaubt (bzw. Lambda auf S3-Bucket). Hierbei definiert man einfach eine separate Bucket Policy (*bucket-exception-rule.json*):

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowLambdaRoleReadAccess",
      "Effect": "Allow",
      "Principal": { "AWS": "*" },
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::xxxxxxx/*",
      "Condition": {
        "StringEquals": {
          "aws:PrincipalArn": "arn:aws:iam::xxxxxxx:role/Lambda-pihole-analyzer"
        }
      }
    }
  ]
}
```

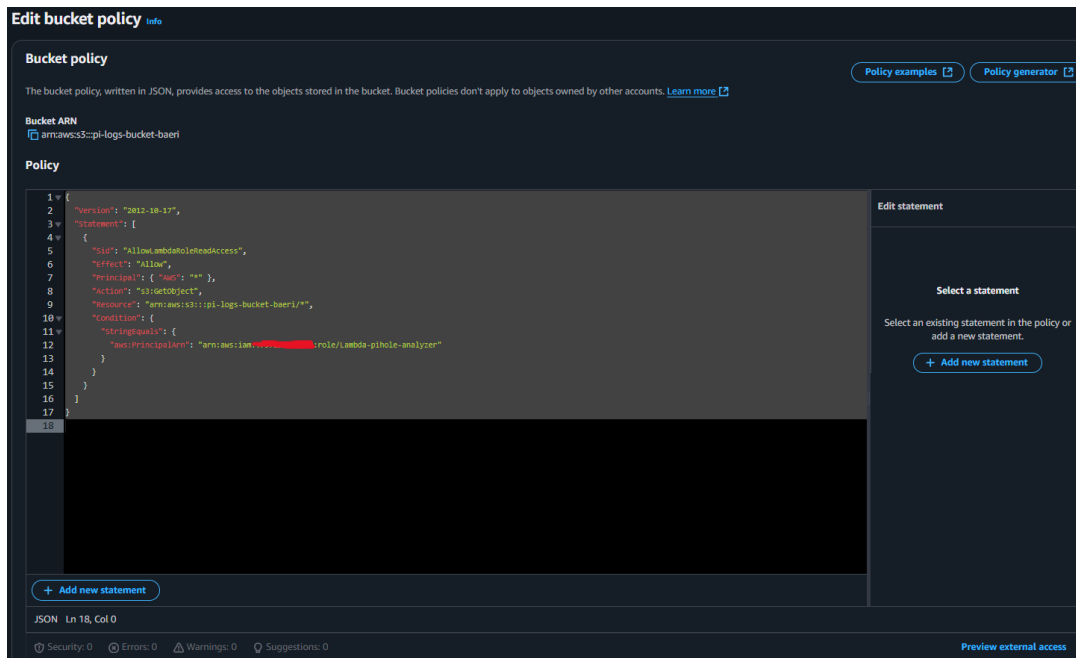


Abbildung 25: AWS - S3 Bucket Exception Policy

Momentan wird alles geloggt und entsprechend auf dem Bucket sortiert. Nun muss das ganze ausgewertet werden.

## 5.2.3 AWS Lambda

Mittels AWS Lambda kann eine Funktion einmalig definiert und ausgeführt werden. Ohne jeglichem Overhead von Servern oder Ähnlichem. Das Setup von AWS Lambda ähnelt sehr dem Erstellen vom S3-Bucket.

Burgermenü → All Services → Compute → Lambda → Create Function

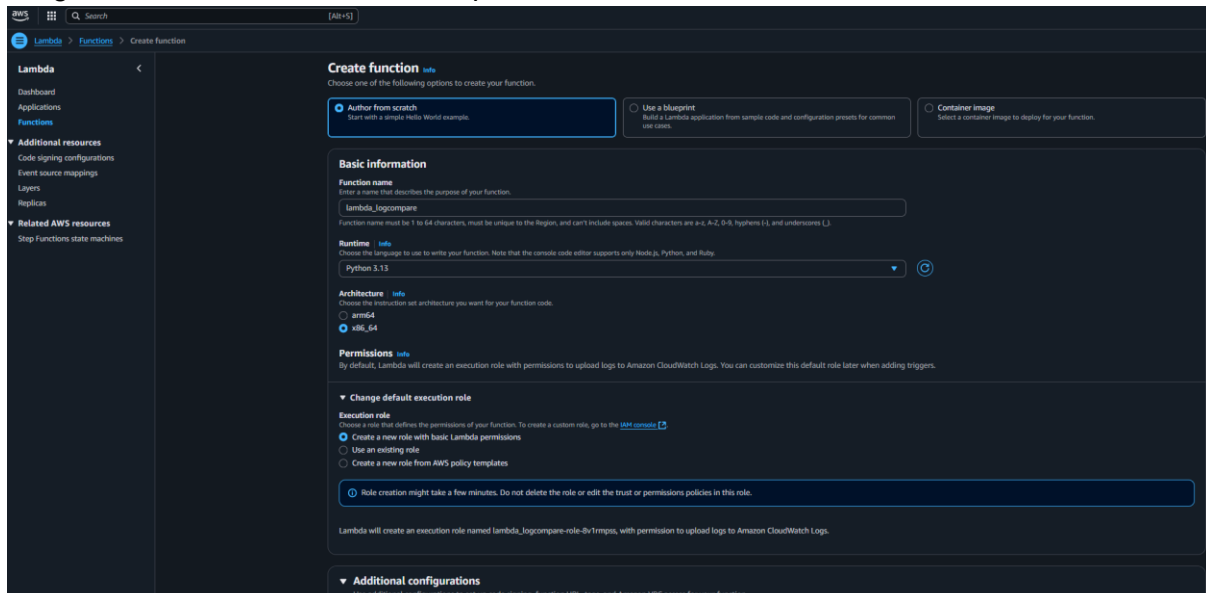


Abbildung 26: AWS - Lambda Create function

Als Lambda Sprache wurde Python 3.13 verwendet, rein aus Bequemlichkeit. Für die Nutzung muss eine separate Rolle und eine separate Policy erstellt werden.

## 5.2.4 IAM Policy

Als Erstes wurde eine Policy für Lambda unter IAM → Policies → Create Policy erstellt.

**Review and create** [info](#)  
Review the permissions, specify details, and tags.

**Policy details**

**Policy name**  
Enter a meaningful name to identify this policy.  
  
Maximum 128 characters. Use alphanumeric and "+-./@\_:" characters.

**Description - optional**  
Add a short explanation for this policy.  
  
Maximum 1,000 characters. Use alphanumeric and "+-./@\_:" characters.

**Permissions defined in this policy** [info](#) [Edit](#)  
Permissions defined in this policy document specify which actions are allowed or denied. To define permissions for an IAM identity (user, group, or role), attach a policy to it.

Allow (2 of 447 services) [Show remaining 445 services](#)

Service	Access level	Resource	Request condition
CloudWatch Logs	Limited: Write	All resources	None
S3	Limited: Read	BucketName] string like [s3-logs-bucket, ObjectPath] string like [All]	None

Abbildung 27: AWS - Lambda IAM Policy

Das JSON zu dieser Policy sieht wie folgt aus (*lambda-iam-policy.json*):

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowS3ReadAccess",
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:ListBucket"
      ],
      "Resource": "arn:aws:s3:::pi-logs-bucket/*"
    },
    {
      "Sid": "AllowLogging",
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents"
      ],
      "Resource": "*"
    }
  ]
}
```

Mit dieser Policy konnte dann anschliessend eine Rolle erstellt werden, welche unter IAM → Roles → Create Role passiert. Die Policy, wie auch die Rolle wurde „Lambda-pihole-analyzer“ genannt.



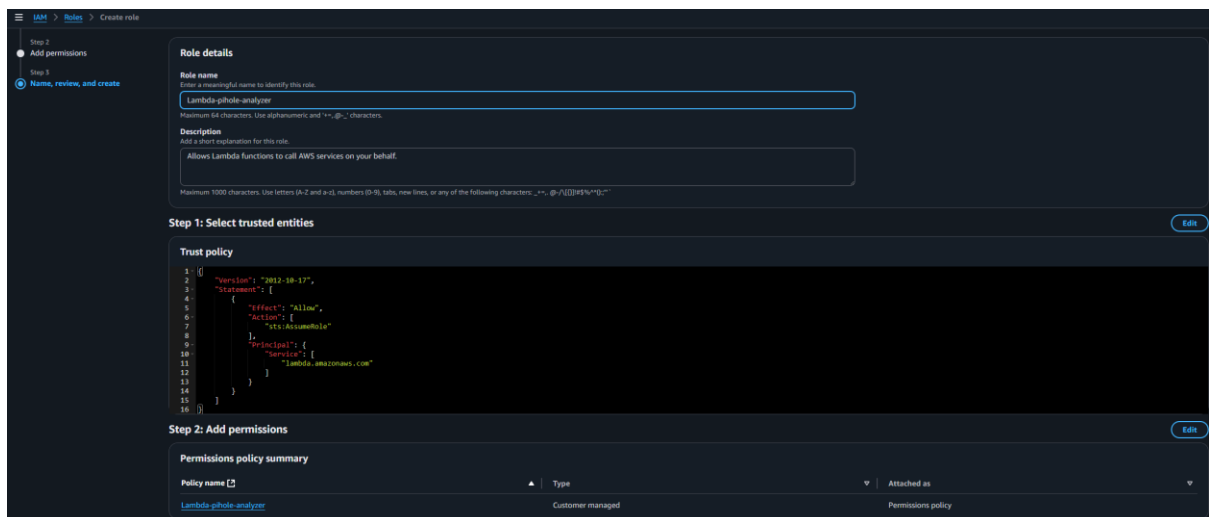


Abbildung 28: AWS - IAM Policy Lambda-pihole-analyzer

## Trigger

Damit die Funktion auslöst, braucht es einen Trigger. Hierfür eignet sich ein S3-Bucket-Trigger sehr gut. Dies bedeutet, dass bei einem Update im S3-Bucket die Funktion (=die Analyse) sofort gestartet wird.

Das „leere“ Template sieht so aus:

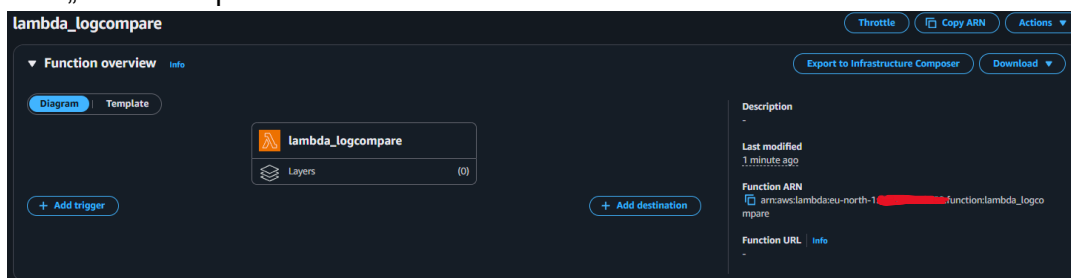


Abbildung 29: AWS - Lambda leeres Template

Der Trigger passiert bei jeglichen Updates im Ordner `pihole/` im vorher erstellten Bucket. Es wurde der Unterordner `pihole/` genommen um Rekursion zu vermeiden, da jeglicher weiterer Output NICHT in `pihole/` gesetzt wird.

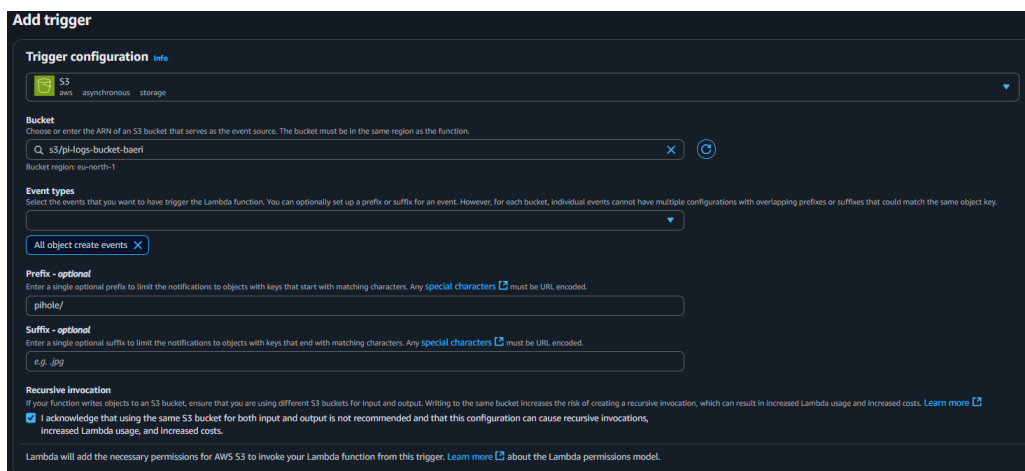


Abbildung 30: AWS - Lambda Trigger konfiguration

Damit sind die Konfigurationen für Lambda eigentlich gemacht. Es wird eine Discord Webhook verwendet, welche im nächsten Kapitel beschrieben wird. Das Skript für die Analyse bzw. den Vergleich ist im Anhang zu finden.

### Layer

Damit die externen Module funktionieren, müssen die Python-Module extern erstellt werden und via Lambda → Layers → Create Layer hochgeladen werden. Die einzelnen Module müssen per .zip hochgeladen werden. Das eigentliche Erstellen der externen Module funktioniert mit dem Command `pip install <MODULE> -t ..`. Damit der Layer hinzugefügt werden kann, müssen compatible runtimes und architectures definiert werden.

The screenshot shows the 'Create layer' page in the AWS Lambda console. Under 'Layer configuration', the 'Name' field is 'python\_modules'. The 'Upload a zip file' option is selected, and a file named 'python.zip' (17.03 MB) is shown. The 'Compatible architectures' dropdown is set to 'x86\_64'. At the bottom, a table lists the created layer:

Merge order	Name	Layer version	Compatible runtimes	Compatible architectures	Version ARN
1	python_modules	4	python3.13	x86_64	arn:aws:lambda:eu-north-1:123456789012:layer:python_modules:4

Abbildung 32: AWS - Lambda Layers

Anschliessend kann der Layer mit den Python Modulen (=python\_modules) auf die Lambda-Funktion gepackt werden. Somit sind die Module installiert.

The screenshot shows the 'Add layer' page in the AWS Lambda console. Under 'Function runtime settings', the 'Runtime' is 'Python 3.13' and the 'Architecture' is 'x86\_64'. In the 'Choose a layer' section, the 'Custom layers' option is selected. A dropdown menu shows 'python\_modules' and the 'Version' dropdown is set to '4'. The 'Add' button is highlighted in orange.

Abbildung 31: AWS - Lambda Create Layer

Wie im Skript schon gesehen, gibt es einen Discord Webhook, also eine Möglichkeit um eine Notification zu bekommen. Dies wird im nächsten Kapitel erstellt.

Und siehe da: Durch einen einfachen Test mit dem folgenden Testevent eines bestehenden Logs funktioniert das Skript.

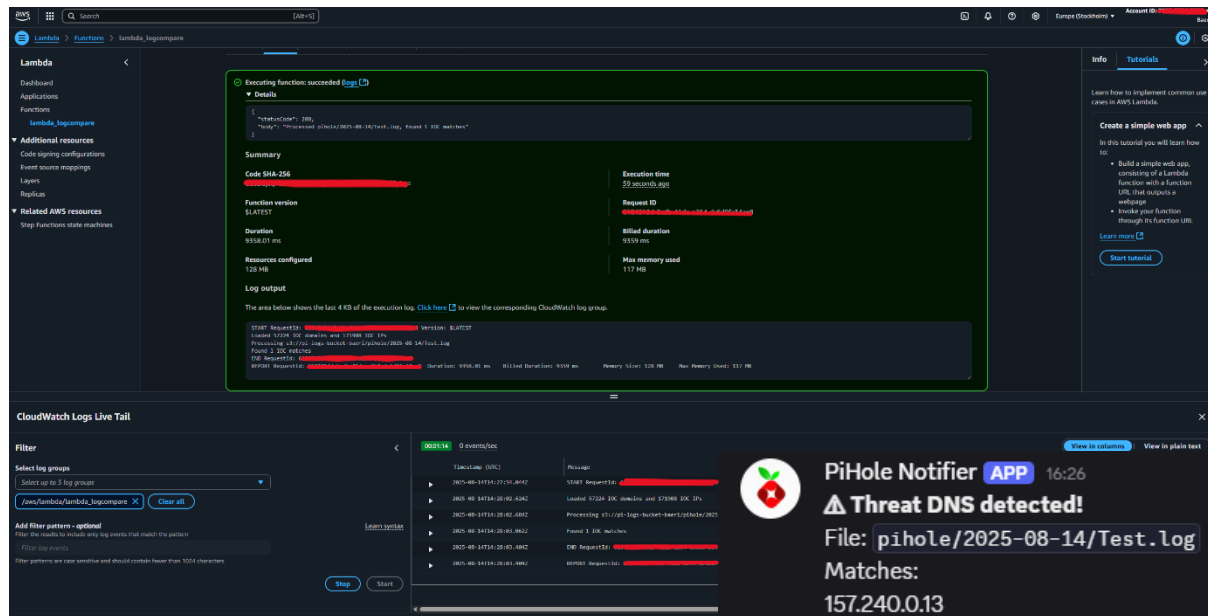


Abbildung 33: AWS - Lambda Success

## 5.3 Discord Webhook

Es wurde ausserdem ein Discord Webhook verwendet, um bei einem Vorfall (=IoC) stets benachrichtigt zu werden.

Die Webhook kann ganz einfach im eigenen Discord Server erstellt werden unter:  
Server Settings → Apps → Integrations → Webhooks

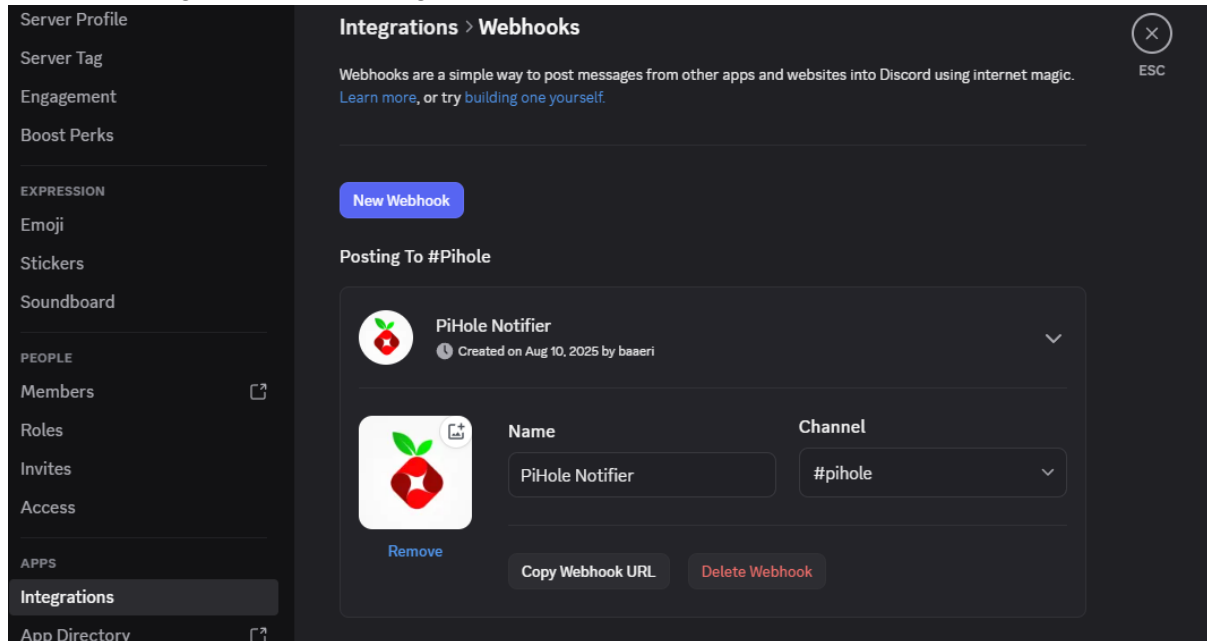


Abbildung 34: Discord - Webhook

## 5.4 N8n

N8n ist ein quellenoffenes Workflow Automatisierungstool mit GUI-Editor. Durch API-first-Architektur und einem modularen Aufbau eignet es sich perfekt für einen Self-hosted RSS-Feed.

Das Tool wurde über Docker mit der Docker-Dokumentation erstellt.

Auf n8n kann anschliessend ein Workflow erstellt werden.

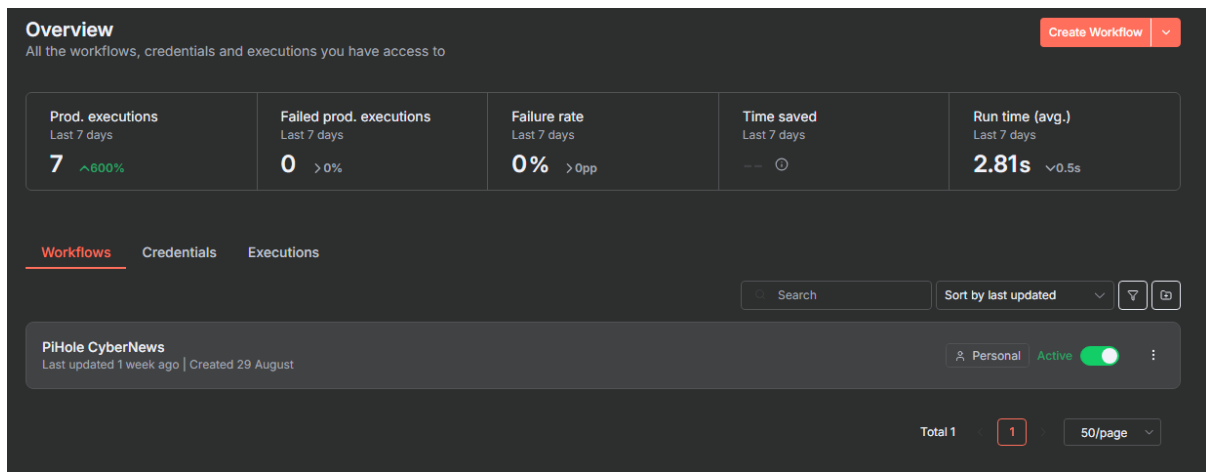


Abbildung 35: N8n - Startseite

Darauf wurde ein Workflow erstellt, der jeden Tag um 8:00 Uhr vier RSS-Feeds abfragt und alle in eine Liste zusammenführt. Anschliessend wird die Liste auf den aktuellen Tag gefiltert und per Discord Webhook formatiert geschickt.

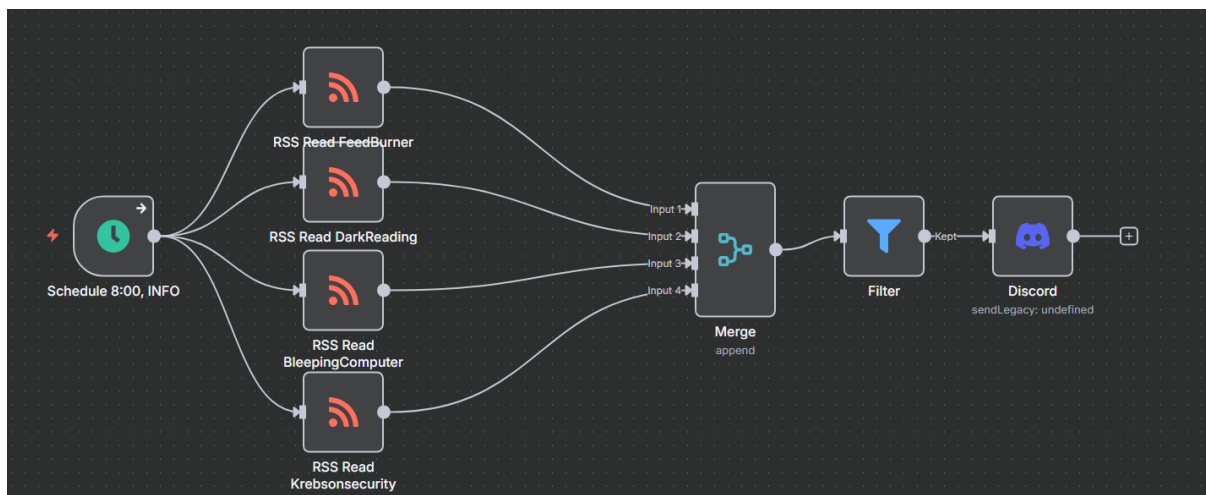




Abbildung 36: N8n - Workflow

Die RSS-Feeds sind im Anhang definiert.





Security Info **APP** 08:00



**Saturday, 06. September 2025**

## **GOP Cries Censorship Over Spam Filters That Work**

The chairman of the Federal Trade Commission (FTC) last week sent a letter to Google's CEO demanding to know why Gmail was blocking messages from Republican senders while allegedly failing to block similar missives supporting Democrats. The letter followed media reports accusing Gmail of disproportionately flagging messages from the GOP fundraising platform WinRed and sending them to the spam folder. But according to experts who track daily spam volumes worldwide, WinRed's messages are getting blocked more because its methods of blasting email are increasingly way more spammy than that of ActBlue, the fundraising platform for Democrats.

<https://krebsonsecurity.com/2025/09/gop-cries-censorship-over-spam-filters-that-work/>

27

## 6. Evaluation und Validation

Die Plattform konnte erfolgreich umgesetzt und in Betrieb genommen werden. Die wichtigsten Ziele – zentrale Verwaltung, Automatisierung und Energieeffizienz – wurden erreicht:

- Zentrale Verwaltung: Alle Dienste sind über Dashy und NGINX Proxy Manager erreichbar, wodurch ein klar strukturiertes und konsistentes System entsteht.
- DNS-Sicherheit: Durch die Kombination von Pi-hole und Unbound wird Werbung blockiert und die Abhängigkeit von externen DNS-Anbietern reduziert.
- Monitoring und Benachrichtigung: Uptime Kuma erkennt zuverlässig Ausfälle, die in Echtzeit per Discord Webhook gemeldet werden.
- Automatisierung: Watchtower aktualisiert Container selbstständig, während n8n und AWS Lambda wiederkehrende Prozesse ohne manuelles Eingreifen abwickeln.
- Cloud-Anbindung: Backups und Logdaten werden automatisiert in AWS S3 übertragen, wodurch auch im Fehlerfall ein Wiederherstellungsszenario gewährleistet ist.
- Skalierbarkeit: Dank Docker lassen sich weitere Services problemlos ergänzen.

Während der Implementierung traten jedoch auch Einschränkungen auf:

- Problem mit .local: Der eingesetzte Swisscom-Router verursachte Konflikte mit der .local-Namensauflösung. Daher wurde die Domainendung auf .lan umgestellt, um eine stabile Funktion des Proxy Managers sicherzustellen.
- Leistungslimitierungen: Die begrenzten Hardware-Ressourcen des Raspberry Pi schränken die Langzeitstabilität unter höherer Last ein.

Insgesamt bestätigt die Evaluation, dass die entwickelte Plattform sowohl im Homelab als auch in kleinen Unternehmensumgebungen einsetzbar ist und die identifizierten Probleme – fehlende Übersicht, manuelle Wartung und Sicherheitslücken – wirksam adressiert.

## 7. Ausblick

Die Plattform hat gezeigt, dass sich mit geringem Hardwareeinsatz ein stabiles, automatisiertes und zentrales System aufbauen lässt. Dennoch bestehen verschiedene Ansatzpunkte für eine Weiterentwicklung:

- Mehr Leistung: Einsatz stärkerer Hardware oder Virtualisierung für höhere Lasten.
- Mehr Sicherheit: Ergänzung um IDS/IPS-Lösungen wie Suricata.
- Cloud-Optionen: Anbindung alternativer Provider oder Self-Hosted-Lösungen wie MinIO.
- Mehr Komfort: Automatisierte Installationsskripte für einfachere Einrichtung.

Die bisherige Lösung bildet damit eine solide Basis, die sowohl im Homelab als auch in kleinen Unternehmensumgebungen einen stabilen Betrieb ermöglicht. Mit gezielten Weiterentwicklungen kann die Plattform in Richtung einer noch professionelleren und breiter einsetzbaren Infrastruktur wachsen.



## 8. Anhänge

### 8.1 Links

Raspberry Pi v1.8.5 Imager	<a href="https://www.raspberrypi.com/software/">https://www.raspberrypi.com/software/</a>
Raspberry Pi Lite OS 64-Bit	<a href="https://www.raspberrypi.com/software/operating-systems/">https://www.raspberrypi.com/software/operating-systems/</a>
Docker	<a href="https://docs.docker.com/engine/install/">https://docs.docker.com/engine/install/</a>
PiHole	<a href="https://github.com/pi-hole/pi-hole">https://github.com/pi-hole/pi-hole</a>
Unbound	<a href="https://hub.docker.com/r/mvance/unbound">https://hub.docker.com/r/mvance/unbound</a>
Uptime Kuma	<a href="https://github.com/louislam/uptime-kuma/tree/master?tab=readme-ov-file">https://github.com/louislam/uptime-kuma/tree/master?tab=readme-ov-file</a>
Dashy	<a href="https://github.com/Lissy93/dashy/blob/master/docker-compose.yml">https://github.com/Lissy93/dashy/blob/master/docker-compose.yml</a>
Watchtower	<a href="https://containrrr.dev/watchtower/">https://containrrr.dev/watchtower/</a>
AWS CLI	<a href="https://github.com/aws/aws-cli">https://github.com/aws/aws-cli</a>
N8n	<a href="https://hub.docker.com/r/n8nio/n8n">https://hub.docker.com/r/n8nio/n8n</a>

Tabelle 2: Links

#### Verwendete IoC-Listen

URLhaus	<a href="https://urlhaus.abuse.ch/downloads/hostfile/">https://urlhaus.abuse.ch/downloads/hostfile/</a>
Threatfox	<a href="https://threatfox.abuse.ch/downloads/hostfile/">https://threatfox.abuse.ch/downloads/hostfile/</a>
Feodotracker	<a href="https://feodotracker.abuse.ch/downloads/ipblocklist.txt">https://feodotracker.abuse.ch/downloads/ipblocklist.txt</a>
Ipsum (Level 4)	<a href="https://raw.githubusercontent.com/stamparm/ipsum/master/levels/4.txt">https://raw.githubusercontent.com/stamparm/ipsum/master/levels/4.txt</a>
Ipsum (Level 5)	<a href="https://raw.githubusercontent.com/stamparm/ipsum/master/levels/5.txt">https://raw.githubusercontent.com/stamparm/ipsum/master/levels/5.txt</a>
Ipsum (Level 6)	<a href="https://raw.githubusercontent.com/stamparm/ipsum/master/levels/6.txt">https://raw.githubusercontent.com/stamparm/ipsum/master/levels/6.txt</a>
Ipsum (Level 7)	<a href="https://raw.githubusercontent.com/stamparm/ipsum/master/levels/7.txt">https://raw.githubusercontent.com/stamparm/ipsum/master/levels/7.txt</a>
Ipsum (Level 8)	<a href="https://raw.githubusercontent.com/stamparm/ipsum/master/levels/8.txt">https://raw.githubusercontent.com/stamparm/ipsum/master/levels/8.txt</a>

Tabelle 3: Verwendete IoC-Listen

#### RSS-Feeds

FeedBurner	<a href="https://feeds.feedburner.com/TheHackersNews">https://feeds.feedburner.com/TheHackersNews</a>
DarkReading	<a href="https://www.darkreading.com/rss.xml">https://www.darkreading.com/rss.xml</a>
BleepingComputer	<a href="https://www.bleepingcomputer.com/feed/">https://www.bleepingcomputer.com/feed/</a>
Krebsonsecurity	<a href="https://krebsonsecurity.com/feed/">https://krebsonsecurity.com/feed/</a>

Tabelle 4: Verwendete RSS-Feeds

## 8.2 Skripte

Das Pythonskript, welches für die AWS Lambda Funktion verwendet wurde:

```
import boto3
import os
import re
import requests
from urllib.parse import urlparse
import tldextract

def domain_in_ioc(domain):
    ext = tldextract.extract(domain.lower())
    if not ext.domain or not ext.suffix:
        return False

    # Registrierbare Domain
    base_domain = f"{ext.domain}.{ext.suffix}"

    # Direkt-Check: Domain in IOC-Liste?
    if base_domain in ioc_domains:
        return True

    # Falls Subdomain vorhanden → auch die ganze Domain prüfen
    if ext.subdomain:
        full_domain = f"{ext.subdomain}.{base_domain}"
        if full_domain in ioc_domains:
            return True
    return False

s3 = boto3.client('s3')

DISCORD_WEBHOOK_URL = "https://discord.com/api/webhooks/xxxxxxxxxxxxxxxxxxxxx"
IOC_FEEDS = [
    # Domains (127.0.0.1 <DOMAIN>)
    "https://urlhaus.abuse.ch/downloads/hostfile/", # Payload delivery & C2 Botnets
    "https://threatfox.abuse.ch/downloads/hostfile/", # Payload delivery & C2 Botnets

    # IPs
    "https://feodotracker.abuse.ch/downloads/ipblocklist.txt", # C2 Botnets & IoC
    "https://raw.githubusercontent.com/stamparm/ipsu/master/levels/4.txt", # Level 4
    "https://raw.githubusercontent.com/stamparm/ipsu/master/levels/5.txt", # Level 5
    "https://raw.githubusercontent.com/stamparm/ipsu/master/levels/6.txt", # Level 6
    "https://raw.githubusercontent.com/stamparm/ipsu/master/levels/7.txt", # Level 7
    "https://raw.githubusercontent.com/stamparm/ipsu/master/levels/8.txt", # Level 8
]

ioc_domains = set()
ioc_ips = set()
ext = tldextract.TLDExtract(cache_dir="/tmp/tld_cache") # tldextract cachefolder

# IPv4 Regex: keine CIDR, nur einzelne IPs
ipv4_pattern = re.compile(r"^(?:\d{1,3}\.){3}\d{1,3}$")

def load_ioc_feeds():
    ioc_domains.clear()
    ioc_ips.clear()

    for feed in IOC_FEEDS:
        try:
            resp = requests.get(feed, timeout=10)
            resp.raise_for_status()
            lines = resp.text.splitlines()

            for line in lines:
```

```

        line = line.strip()
        if not line or line.startswith("#") or line.startswith(";"):
            continue

        # Domains: urlhaus.abuse.ch + threatfox.abuse.ch
        if "urlhaus.abuse.ch" in feed or "threatfox.abuse.ch" in feed:
            # Hostfile: "127.0.0.1<TAB>domain.xyz"
            if line.startswith("127.0.0.1"):
                parts = line.split()
                if len(parts) >= 2:
                    ioc_domains.add(parts[1].lower())

        # IPs: feodotracker.abuse.ch + Github/Ipsum
        elif "feodotracker.abuse.ch" in feed or "raw.githubusercontent.com" in
feed:
            ip_part = line.split()[0] # Ignore number of blacklists

            if ipv4_pattern.match(ip_part):
                if all(0 <= int(octet) <= 255 for octet in ip_part.split(".")):
                    ioc_ips.add(ip_part)

    except Exception as e:
        print(f"Error loading IOC feed {feed}: {e}")

    print(f"Loaded {len(ioc_domains)} IOC domains and {len(ioc_ips)} IOC IPs")

# Log-Parser, IPs & Domains
def extract_domains_from_log(log_text):
    domain_pattern = re.compile(r"(?:forwarded|reply|query\[A\]) ([a-zA-Z0-9.-]+\.[a-zA-Z]{2,})")
    return set(match.group(1).lower() for match in domain_pattern.finditer(log_text))

# Subdomains (only TLDs)
def domain_in_ioc(domain):
    ext = tldextract.extract(domain.lower())
    if not ext.domain or not ext.suffix:
        return False

    base_domain = f"{ext.domain}.{ext.suffix}"

    if base_domain in ioc_domains:
        return True

    # if subdomain: check that aswell
    if ext.subdomain:
        full_domain = f"{ext.subdomain}.{base_domain}"
        if full_domain in ioc_domains:
            return True
    return False

def extract_ips_from_log(log_text):
    ip_pattern = re.compile(r"(?:cached[-\w]*|reply) .* is (\d{1,3}(\.?\d{1,3}){3})")
    return set(match.group(1) for match in ip_pattern.finditer(log_text))

# Discord alerting
def send_discord_alert(matches, s3_key):
    content = f"*⚠ Threat DNS detected!**\nFile: `{s3_key}`\nMatches:\n" +
    "\n".join(matches)
    try:
        response = requests.post(DISCORD_WEBHOOK_URL, json={"content": content}, timeout=5)
        if response.status_code != 204:
            print(f"Discord alert failed with status {response.status_code}")
    except Exception as e:
        print(f"Error sending Discord alert: {e}")

# Getting bucket & file from s3, sending logs from lambda, actual matching mechanism

```

```
def lambda_handler(event, context):
    load_ioc_feeds()

    record = event['Records'][0]
    bucket = record['s3']['bucket']['name']
    key = record['s3']['object']['key']
    print(f"Processing s3://{bucket}/{key}")

    tmp_file = f"/tmp/{os.path.basename(key)}"
    s3.download_file(bucket, key, tmp_file)

    with open(tmp_file, "r", encoding="utf-8", errors="ignore") as f:
        log_text = f.read()

    domains_in_log = extract_domains_from_log(log_text)
    ips_in_log = extract_ips_from_log(log_text)

    matches = sorted(
        [d for d in domains_in_log if domain_in_ioc(d)] +
        [ip for ip in ips_in_log if ip in ioc_ips]
    )

    if matches:
        print(f"Found {len(matches)} IOC matches")
        send_discord_alert(matches, key)
    else:
        print("No IOC matches found")

    return {
        'statusCode': 200,
        'body': f'Processed {key}, found {len(matches)} IOC matches'
    }
```

## 8.3 Abbildungsverzeichnis

Abbildung 1: Konzeptplan .....	3
Abbildung 2: Installation - Raspberry (OS Customisation).....	6
Abbildung 3: Installation - Raspberry (Konfiguration).....	6
Abbildung 4: Raspberry - statische IP setzen .....	7
Abbildung 5: Docker - Version.....	7
Abbildung 6: Pi-Hole - Container start .....	8
Abbildung 7: Pi-Hole - Blockliste .....	8
Abbildung 8: Unbound - Installation.....	9
Abbildung 9: Uptime Kuma - Admin Account .....	10
Abbildung 10: Uptime Kuma - Startseite.....	10
Abbildung 11: Dashy - Startseite .....	11
Abbildung 12: Dashy - Startseite (Modifiziert) .....	11
Abbildung 13: Nginx Proxy Manager - Anmelden .....	12
Abbildung 14: Pi-Hole - local DNS records.....	12
Abbildung 15: Nginx Proxy Manager - Proxy Einträge.....	13
Abbildung 16: Nginx Proxy Manager - Einzelner Proxy Eintrag .....	13
Abbildung 17: AWS - IAM Dashboard.....	15
Abbildung 18: AWS - IAM Create User .....	15
Abbildung 19: AWS - All Services:.....	15
Abbildung 20: AWS - pi-log-uploader IAM Optionen .....	15
Abbildung 21: AWS - pi-log-uploader Permissions .....	16
Abbildung 22: AWS - S3 Bucket Konfigurationen .....	17
Abbildung 23: AWS - S3 Bucket Lifecycle Management.....	18
Abbildung 24: AWS - S3 Bucket Lifecycle Rule .....	19
Abbildung 25: AWS - S3 Bucket Exception Policy .....	20
Abbildung 26: AWS - Lambda Create function .....	20
Abbildung 27: AWS - Lambda IAM Policy.....	21
Abbildung 28: AWS - IAM Policy Lambda-pihole-analyzer.....	22
Abbildung 29: AWS - Lambda leeres Template.....	22
Abbildung 30: AWS - Lambda Trigger konfiguration .....	22
Abbildung 31: AWS - Lambda Create Layer .....	23
Abbildung 32: AWS - Lambda Layers.....	23
Abbildung 33: AWS - Lambda Success .....	24
Abbildung 34: Discord - Webhook .....	25
Abbildung 35: N8n - Startseite .....	26
Abbildung 36: N8n - Workflow.....	26
Abbildung 37: N8n - Discord Webhookparameter .....	27
Abbildung 38: N8n - Filterparameter .....	27
Abbildung 39: N8n - Discordnachricht .....	27

## 8.4 Tabellenverzeichnis

Tabelle 1: Auflösung Dienste .....	13
Tabelle 2: Links .....	30
Tabelle 3: Verwendete IoC-Listen.....	30
Tabelle 4: Verwendete RSS-Feeds.....	30