

[DIY-Projekt] Traefik2 (Google-OAuth/TLS) feat. Nextcloud und Guacamole (und mehr)

(oder: von hinten durch die Brust ins Auge - eine Odyssee)

1. Vorwort

Mein Netzwerk ist überschaubar aber wächst immer weiter. 2 PCs, ein Notebook, Drucker, Scanner und das Smartphone bildeten den Start. Das alte Notebook wurde durch ein Tablet ersetzt und ein ‚Mini-PC‘ kam hinzu. Letzterer ist ein µATX System mit Celeron J1900 und 8 GB RAM. Mit SSD das perfekte System für einen kleinen, sparsamen ‚Server‘.

Eine Möglichkeit zur Vernetzung und Steuerung war damit ein naheliegender Gedanke. Dafür bieten sich unterschiedlichste Ansätze an. WakeOnLAN und Remote Desktop waren die ersten Ziele, eine selbstgehostete Cloud (Nextcloud) als Wunsch im Hinterkopf.

Eigentlich brauchte ich nur ein neues Projekt gegen die Langeweile, ich hatte nicht im Ansatz vermutet, was für eine Odyssee ich starte. Für einen Neueinsteiger war das eine recht steile Lernkurve. Lange Weile hatte ich zumindest keine.

UPDATE: Nach längerer Suche und einigen Experimenten ein kurzes Fazit und eine Neuorientierung. Dabei interessierten mich u.a. Alternativen für Yunohost, da gibt es einige, Dockstarter und Cloudron finden hier öfter Erwähnung, neben einigen proprietären und kostenpflichtigen Lösungen.

Wirklichen Fortschritt bei meiner Suche nach der ‚idealen‘ Lösung brachte ein Tipp von @LieberNetterFloh. Die eierlegende Wollmilchsau als Server, allerdings anspruchsvoll bei der Erstkonfiguration, doch auch keine Raketenwissenschaft. Mit einem kleinen bisschen Erfahrung im Umgang mit der Linux-CLI ist das aber zu bewältigen.

Das Endergebnis kann sich dann sehen lassen, dazu mehr unter 3.2.

2. Vorbereitungen

Das System sollte modular werden und einfach zu warten, upzudaten und zu erweitern sein.

Die Basis ist unverändert ein Windows 10 mit einem Linuxmint in der VM. Wahlweise kann auch Ubuntu oder ein anderes Linux der Wahl benutzt werden. Zum Anfang werden hauptsächlich Docker und Docker-Compose benötigt. Dazu im entsprechenden Anhang dann mehr.

Allen beteiligten Geräten und Systemen habe ich im Router eine feste IP innerhalb des LANs zugewiesen.

3. Die Qual der Wahl (der Mittel)

3.1. Die Qual

Beim Testen der Optionen für WakeOnLAN und Remote Desktop wurde schnell klar, daß zum einen der Einsatz mehrerer Tools die Verlässlichkeit nicht gerade erhöht und zum anderen die

proprietären Remote Tools (Teamspeak, Anydesk und co.) zwar komfortabel sein können, allerdings nur, wenn sie denn auch laufen wie gewünscht. Gerade bei letzterem haperte es in meinem Test aber deutlich. Zudem ist der Einsatz solcher Lösungen in den meisten Fällen Overkill.

Schnell kristallisierte sich heraus, daß eine browserbasierte Lösung das Ziel sein wird. Diese hat den Vorteil, daß sie auf mobilen Geräten genauso funktioniert wie auch auf meinen Desktops (betriebssystemunabhängig), ohne zusätzliche Apps nutzen zu müssen.

3.2. Die Wahl Teil 1: Lokales Netz

Update: Hier gab es einige Änderungen bei der Lösung des Hostings via Traefik2.

Die Ziele waren WakeOnLAN, SleepOnLAN, Remote Desktop, SSH und die Nextcloud. Zur Steuerung des Ganzen ein lokaler (Apache) HTTP Server mit mehrfacher Redundanz (man weiß ja nie) und eine HTML-Lösung zum Absetzen der Steuerungsbefehle.

Vorweg; wer es einfach will, ist, was Nextcloud anbelangt, bei Yunohost gut aufgehoben. Die geführte Installation ist einfach und benötigt nur funktionierende (Sub-)Domains, ich nutze DynDNS via DuckDNS. Leider funktioniert hier die Guacamole Integration nicht.

Alternativ kann hier ein externes Docker-Stack genutzt werden, welches mit wenig Aufwand eingerichtet ist.

Keine der anderen Lösungen, die ich getestet habe, kam an die intuitive Einbindung der Domains von Yunohost nahe. Auch die Einbindung von Fail2Ban und Let's Encrypt Zertifikaten ist übersichtlich gelöst. Einzig die eingeschränkte Auswahl an ‚Applications‘ ist hier ein kleines Manko, nicht alle werden aktiv gewartet.

Die Anhänge für Yunohost und Guacamole sind jetzt in den **Anhang Part X (Archiv)** gewandert.

Lokales Hosting für Nextcloud, Guacamole und co.

Dank, wie schon erwähnt, @LieberNetterFloh und seinem Link zu einem Docker Server Tutorial auf smarthomebeginners.com, ist mein Konglomerat einem einzelnen Linux-Gast gewichen.

<https://www.smarthomebeginner.com/traefik-2-docker-tutorial>

Dank Traefik2 als Reverse-Proxy ist das System via Google OAuth2 abgesichert, bringt strikte Policies bezüglich Rate-Limit und Header-Authentication mit und autogeneriert Let's Encrypt Zertifikate für alle internen Sub-Domains. Nextcloud bildet die Einzige Ausnahme und braucht manuelle Zuwendung, ist einfach zu lösen, mehr im Anhang.

Keine Bange, sieht komplizierter aus als es ist. Im Anhang Anmerkungen zu den kritischen Stellen und potentiellen Fallstricken.

Vorbereitungen:

- Linux System der Wahl, ich nutze Mint, habe aber auch auf Ubuntu keine Probleme gehabt.
- Docker Installation
- Docker-Compose Installation
- DynDNS Domain via DuckDNS oder CloudFlare
- Google-Account für OAuth-Einbindung in Traefik

Traefik2-Tutorial:

Ich empfehle hier in Ruhe zu arbeiten, kleine Fehler haben große Wirkung.

Die Docker-Compose YAML ist sehr sensitiv, was Formatierung angeht. Die hierarchischen Einrückungen sind immer 2 Spalten tief. Nicht umsonst empfiehlt der Autor die Codebeispiele mit allen Leerzeichen zu übernehmen. Meine Skriptbeispiele als Referenz nehmen.

Entscheidend ist die Ersteinrichtung von Traefik2 mit Let's Encrypt und anschließend OAuth. Wer hier gründlich arbeitet, hat im Anschluß die perfekte Basis zur Einbindung weiterer Container. Letzteres ist recht unkompliziert, wenn das Compose-Skript richtig eingerichtet ist.

Ich hatte beim Folgen der Anleitung nur kleinere Fragen, die sich schnell recherchieren liessen. Dazu im Anhang einige Tipps.

Wichtiger Grundpfeiler ist die .ENV Datei für die verschiedenen Umgebungsvariablen des Stacks. Dazu kommen noch die TOMLs für die Traefik2-Middlewares. Alles keine Hexerei, anschauliche Beispiele zu den Dateien im Anhang.

Im Anhang gehe ich den Prozess zum Großteil durch, für mehr Infos einfach im Thread fragen.

Weiter geht's im **Anhang Part 3**.

WakeOnLAN:

Ein smartes PHP Skript war der perfekte Anfang. Dieses läßt sich einfach (in meinem Fall per iFrame) in eine HTML einbinden und funktioniert wie beworben.

Wake-on-lan.php: <https://github.com/AndiSHFR/wake-on-lan.php>

TIPP: Einen Eintrag bei laufendem Server über die Website erstellen und über das Tools-Dropdown-Menü mittels 'Save Configuration' speichern. Die gespeicherte JSON Datei sichern. Diese läßt sich komfortabel über den Texteditor der Wahl erweitern und nach Belieben sortieren. Danach wieder in den Ordner kopieren (oder via SFTP schicken) und über die Website mittels 'Load Configuration' laden.

SleepOnLAN:

Die Lösung war hier ein Mikroserver (Windows/Linux), der unauffällig im Hintergrund läuft und an einem definierten Port auf Steuerbefehle wartet

(an `http://<IP_or_Hostname>:<Port>/your_command`).

Das Github-Projekt kann über eine JSON konfiguriert werden.

Sleep-On-LAN (SOL): <https://github.com/SR-G/sleep-on-lan>

Unter Windows kann die EXE per Aufgabenplanung unabhängig von der Useranmeldung gestartet werden. Das erlaubt die Steuerung ohne eingeloggt zu sein (der Haken hier: funktioniert nur für Befehle ohne Interaktion durch den User).

Es können auch mehrere Instanzen an unterschiedlichen Ports ‚lauschen‘.

Siehe **Anhang Part 1** mit weiterführenden Tipps.

SSH Zugang:

OpenSSH ist glücklicherweise auch unter Windows 10/11 mittlerweile als zusätzliches Feature aktivierbar. Einfach mittels Powershell aktivieren und den OpenSSH Dienst starten und auf automatisch setzen.

Unter Linux einfach das passende OpenSSH-Paket installieren.

Unter Android habe ich Termux installiert und mittels `pkg upgrade && pkg install openssh` das Paket geladen. Alternativ läßt sich auch 'apt' benutzen. Wichtig: nicht die Store App installieren, da gibt es gerade Probleme daher den GitHub-Link nutzen (ist auch auf der Store Page): <https://github.com/termux/termux-app#installation>.

Siehe **Anhang Part 5** mit weiterführenden Tipps.

Apache Server:

Linux: einfach das Apache2 Paket installieren und eine Konfigurationsdatei anlegen und zuweisen.

Android: meine Wahl fiel auf die kostenpflichtige KSWEB App. Benötigt kein Root und tut was sie soll. Der Preis relativiert sich durch die Nutzung auf 2 Geräten.

Windows: Empfohlen wird gern Xampp, das wollte bei mir aber einfach nicht richtig funktionieren und warf bei eingebetteten PHP-Skripten Fehler aus. Also wurde es UniServerZ (http://uniformserver.com/ZeroXI_documentation/). Funktioniert sofort und arbeitet wie beworben.

Siehe **Anhang Part 6** mit weiterführenden Tipps.

Steuerung:

Übernimmt eine statische HTML-Seite, welche über das WOL-PHP-Skript die Rechner weckt und dann mit den SOL-Instanzen auf den Rechnern kommuniziert, um Befehle zu übermitteln. Eine Landing-Page bindet per iFrames das PHP-Skript und mehrere Unterseiten ein, letztere steuern die Virtuellen Gäste und die verschiedenen Hosts über entsprechende Sammelskripte mit Sprungzielen für die unterschiedlichen Funktionen.

Die VMware-Gäste lassen sich komfortabel mittels vmrun.exe steuern. So lassen sie sich bei Bedarf starten, stoppen und neustarten, ebenso wie Snapshots erstellen, löschen und zurücksetzen.

Die Antworten werden vom Skript als HTML per SFTP an die gerade laufenden Apache-Instanzen geschickt.

Somit lassen sich die meisten Aktionen und Checks per Fernbedienung auslösen und überwachen.

3.3. Die Wahl Teil 1: Fazit

Der Celeron-Server läuft und ist die meiste Zeit im Idle. Apache-Server und SOL arbeiten zuverlässig im Hintergrund.

Das Ubuntu System hat das Upgrade auf 22.04 LTS gemeistert, die Docker-Installation ist pflegeleicht und läßt sich per ‚pull‘ und ‚up -d‘ mittels Docker-Compose sehr einfach updaten.

Update (27.05.2022): Das Ubuntu ist einem Mint Xfce gewichen. Letzteres ist nur ein Viertel so groß und macht seinen Job perfekt. Mehr siehe Anhang Part 2.

Auch der Yunohost ist über die Weboberfläche einfach aktuell zu halten.

Backups der VMs (als Snapshots) werden, wie auch die Acronis Images des Windows Hosts, regelmäßig an verschiedenen Stellen gespeichert.

Der Langzeittest ist am Laufen.

3.4. Die Wahl Teil 2: Der Zugriff von außen

Generell ist das natürlich ein heikles Thema mit einigen Fallstricken.

Erste Empfehlung ist meist der Zugriff über VPN. Ist allerdings auch kein Allheilmittel wenn die dahinterliegende Sicherheitsstruktur löchrig ist.

Alle Geräte im LAN, auch die VM Gäste, haben ihre eigene, feste IP im LAN.

Das erlaubt nur die nötigen Ports dediziert für die Yunohost-VM freizugeben, ohne den Windowshost dafür öffnen zu müssen. Ein Port ermöglicht bei Bedarf Zugriff auf die Guacamole-VM (ist sonst heruntergefahren).

Nextcloud-Sicherheits-Check: <https://scan.nextcloud.com>

Braucht es nur noch eine eigene Domain. Am Besten natürlich ohne Kosten, wenn möglich. Stichwort ist hier Dynamic DNS.

DynDNS:

Meine Suche förderte mehrere Kandidaten zu Tage.

Oftmalige Empfehlung ist DuckDNS. Hier war jedoch schon die Dokumentation entweder 404 oder hoffnungslos veraltet. Nicht so mein Ding.

<https://www.duckdns.org>

Selfhost.de sah vielversprechend aus. Als Pro auf jeden die Möglichkeit zig Subdomains anzulegen. Kontrapunkte sind jedoch der fehlende Direktkontakt zur API für die Aktualisierung der IP und die Inkonsistenz bei der Namensauflösung. Nicht zu gebrauchen trotz gutem Erstem Eindruck.

<https://selfhost.de/cgi-bin/selfhost?p=cms&article=index&CGISESSID=9befd2744a98a93033f352f4d6ac8813>

No-IP macht seine Sache zwar gut, aber nur eine Domain aktualisierbar und ebenfalls nur ein proprietäres Tool für die Aktualisierung haben für einen guten 2. Platz gereicht.

ACHTUNG!: No-IP deaktiviert schnell Domains, wenn man sich nicht regelmäßig einloggt.

<https://www.noip.com>

Sieger wurde DDNSS.de mit einigem Abstand.

<https://ddnss.de/index.php>

Der freie Account erlaubt 5 Domains mit 60 Aktualisierungen pro Tag. Die Aktualisierungen lassen sich via PHP-Seite (mit Schlüssel) einzeln oder in einem Rutsch durchführungen. Die Weiterleitung reagiert prompt und das auch zuverlässig bisher.

Fernwartung (HTTPS zu HTTP Tunnel ins lokale Netz):

In diesem Zusammenhang wird Ngrok bevorzugt ins Rennen geworfen. Macht zugegebenermaßen was es soll, jedoch fand ich keine Möglichkeit eine passive Verbindung aufzubauen, die auf Anfrage den Tunnel aufbaut.

<https://ngrok.com>

Die Suche nach einer Alternative führte zu LocalToNet. Der Client baut mittels Token eine passive Verbindung auf und der Tunnel kann über das Account-Dashboard gestartet werden. Perfekt für meine Zwecke.

<https://localtonet.com>

Der Client ist ein spartanisches Kommandozeilentool (einfach starten), verlangt beim ersten Start das Token aus dem Account-Dashboard und checkt dann die Verbindung. Danach wartet es unauffällig im Hintergrund.

Als Startrampe nutze ich wiederholt die Aufgabenplanung.

Hinweis: bei Veränderung der WAN IP verliert es die Verbindung. Wie erwähnt kümmert sich bei mir das Skript für die DynDNS Aktualisierung um den Neustart des Tasks.

3.5. Die Wahl Teil 2: Fazit

Der Check ergab ein A+ Rating für die Nextcloud, ein guter Anfang.

Ein Skript checkt im Hintergrund die WAN IP und aktualisiert bei Veränderungen durch den täglichen Provider-Reset (und/oder Router-Neustarts) den DDNSS Account. Gleichzeitig startet es den LocalToNet Tunnel neu, sonst wird der unansprechbar.

Letzterer erlaubt den Fernzugriff auf den lokalen Apache Server. So lassen sich Rechner und VMs steuern.

Brauche ich direkten Zugriff auf den Desktop eines der Rechner, wird erst der Tunnel gestartet, dann der Guacamole-Server und ich habe auch am Tablet und Smartphone über den Browser eine komfortable Steuerung. Nach erledigter Arbeit wird Guacamole herunterfahren und der Tunnel beendet.

4. Die Zukunft...

Der Langzeittest läuft, sieht gut aus bisher und ist wartungsarm. Direkter Zugriff auf den Windowshosts und die virtuellen Gäste ist selten nötig, der Großteil läuft über Weboberflächen und SSH.

Nächster Schritt wird die Steuerung von Steckdosen und Licht. Da habe ich in einigen Threads hier in letzter Zeit einige Anregungen aufgeschnappt und als Lesezeichen gesichert.

Da geht noch was.

Update (28.05.2022) Teil 1:

Das Ubuntu ist einem Mint Xfce (1/4 der Größe des Ubuntu) gewichen. Während dieses Prozesses wurden die Notizen aktualisiert und Anhang Part 2 erstellt mit dem Guacamole Installationsprozeß.

Anhang Part 1 wurde um den Installationsprozeß für SOL unter Linux erweitert.

Update (28.05.2022) Teil 2:

Anhang Part 3 für Yunohost und nextcloud angefügt.

Update (28.05.2022) Teil 3:

Anhang Part 4 für HTTPS zu HTTP Tunnel angefügt.

Update (29.05.2022) Teil 1:

Anhang Part 5 für OpenSSH unter Windows 10 erstellt.

Update (29.05.2022) Teil 2:

Anhang Part 6 für Apache unter Linux erstellt.

Update (29.05.2022) Teil 3:

Webseitenbeispiel zur SOL-Steuerung eingefügt.

Update (29.05.2022) Teil 4:

Bilder eingefügt, PDF aktualisiert

Update (30.05.2022) Teil 1:

Index für einfachere Navigation erstellt.

Update (30.05.2022) Teil 2:

Index für einfachere Navigation erstellt.

Update (05.12.2022):

Aktualisierung für Traefik v2 in aktueller Version.

ANHÄNGE

Anmerkungen und Codebeispiele als Repository für mich und alle Interessierten.

Anhang Part 1

SleepOnLAN:

Windows: Die Githubseite schlägt einen 3rd Party Dienstinstallator vor. Das läßt sich natürlich auch über die Dienstverwaltung realisieren.

Meine Wahl fiel auf die Aufgabenplanung. Diese erlaubt den Start unabhängig von der Useranmeldung. So kann der Rechner auch gesteuert werden, wenn er im LogIn-Screen wartet. Darüber steuere ich die Grundfunktionalität, sprich Start/Stop/Reboot. Es gibt eine Limitierung, da keine Userinteraktion mit den gestarteten Prozessen möglich ist.

Wie schon erwähnt können multiple Instanzen an benachbarten Ports 'lauschen'. Z.B. eine unabhängige (Port 8009) und eine, die bei Useranmeldung startet (an Port 8010).

Linux: Hier ist die angebotene Anleitung nützlich. Habe zum Test das Ganze auf dem Ubuntu-System installiert. Wie auch unter bei der Installation als Dienst unter Windows, funktioniert das nur bei angemeldetem User.

Benötigt werden die Linux x86_64 Binary mit JSON aus dem Release-Pack. Ich habe unter /home/mae1cum77 den Ordner /applications/sleep-on-lan angelegt mit Binary und JSON.

Die Service-Config mit:

```
[CODE=Bash]
Sudo nano /etc/systemd/system/sleep-on-lan.service
[/CODE]
```

anlegen und folgenden Inhalt einfügen (den Usernamen in den Pfaden anpassen):

```
[CODE=Bash]
[Unit]
Description=Sleep-On-LAN daemon

[Service]
User=root
WorkingDirectory=/home/mae1cum77/applications/sleep-on-lan
ExecStart=/home/mae1cum77/applications/sleep-on-lan/sol
Restart=always

[Install]
WantedBy=multi-user.target
[/CODE]
```

Mit Strg+O, gefolgt von ENTER, speichern und mit Strg+X schließen.

Zuletzt noch den Service etablieren und starten mit:

```
[CODE=Bash]
systemctl daemon-reload
systemctl enable sleep-on-lan.service
systemctl start sleep-on-lan.service
[/CODE]
```

Konfiguration: Die Steuerung erfolgt über die JSON Datei. Ein Beispiel für Windows:

```
[CODE=json]
{
  "Listeners" : ["UDP:9", "HTTP:8009" ],
  "Auth" : {
    "Login" : "account_name_here",
    "Password" : "passphrase_here"
  },
  "ExitIfAnyPortIsAlreadyUsed" : false,
  "AvoidDualUDPSending" : {
    "Active": true,
    "Delay": "100ms"
  },
  "LogLevel" : "INFO",
}
```

```

"BroadcastIP" : "255.255.255.255",
"Commands" : [
{
  "Operation" : "halt",
  "Command" : "C:\\Windows\\System32\\Shutdown.exe -s -t 0"
},
{
  "Operation" : "restart",
  "Command" : "C:\\Windows\\System32\\Shutdown.exe -r -t 0"
},
{
  "Operation" : "rest",
  "Command" : "C:\\Windows\\System32\\Shutdown.exe -l -t 0"
}
]
}
[/CODE]

```

Login und Password sind hier nach belieben zu konfigurieren.

Steuerbefehle senden: erfolgt über die Browseradresszeile direkt oder als eingebettete Links einer HTML Seite. Die Befehlsnamen werden unter "Operation" definiert.

```

[CODE]
http://<IP_or_Hostname>:<Port>/your_command
::Beispiel (in der JSON referenziert):
http://192.168.1.5:8009/restart
[/CODE]

```

Steuerskript:

Ein CMD-Sammelskript zum Steuern der VMs und einiger Hostfunktionen.

```

[CODE=bash]
@echo off
:=====
=====
::=====
=====
setlocal EnableExtensions
setlocal EnableDelayedExpansion
pushd "%~dp0"
cd "%~dp0"
call %~1
goto :EXIT

```

```

:=====
=====
::=====
=====
:vm2start
"c:\Program Files (x86)\VMware\VMware Workstation\vmrun" -T ws start
"d:\VIRTUALSPACE\Yunohost_Proper\Yunohost_Proper.vmx"
goto:eof
:=====
=====
::=====
=====
:EXIT
endlocal
exit
[/CODE]

```

In der sol.json:

```

[CODE=json]
"Commands" : [
  {
    "Operation" : "vm2start",
    "Command" : "C:\\SOL2\\command.cmd :vm2start"
  }
]
[/CODE]

```

Website zur Steuerung:

Hier habe ich schmerzlos und schmutzig eine HTML als Landing Page erstellt. Diese bindet über iFrame eine Unterseite mit eingebettetem WOL-PHP für die Grundsteuerung der Rechner ein.

Eine oben positionierte Tabelle dient zur Navigation.

Eine index.html im Hauptordner und ein Unterordner jeweils für die Unterseiten. Ein weiterer Unterordner ist jeweils für Fav-Icon und die Platzhalterseite für die Antwort der SOL Binary.

Beim Basteln der Seite zeigte sich, daß inline CSS am robustesten ist.

Die Seite ist ‚screen aware‘ und somit auch auf dem Smartphone-Bildschirm perfekt benutzbar. Nur die PHP muß man am Smartphone seitlich wischen für alle Optionen. Ein verschmerzbares Opfer.

```

[CODE=HTML]
<!DOCTYPE html>
<html>
<head>
<title>CBS INDEX</title>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1">
<meta name="Author" content="mae1cum77">
<meta name="Description" content="Managing Powerstates of LAN PCs">

```

```
<link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
<link href="https://fonts.googleapis.com/css2?family=Open+Sans&display=swap"
rel="stylesheet">
<link rel="icon" href="wolremote/wol/star77.ico">
<style>
html,
body{
    min-height: 100%;
    height: 100%;
    padding: 0;
    background:#161616;
    font-family: 'Open Sans', sans-serif;
}
iframe {
    min-height: 100%;
    height: 100%;
    padding: 0;
}
h1 {
    color: #CBCBCB;
    padding: 1%;
    font-size: 50%;
}
p {
    color: #CBCBCB;
    padding: 1%;
    font-size: 100%;
}
table {
    color: #CBCBCB;
    padding: 1%;
    font-size: 100%;
    background:#252525;
}
a:visited {
    color: #CBCBCB;
}
a {
    color: #CBCBCB;
}
@media screen and (min-width:481px){
    html,
    body{
        min-height: 100%;
        padding: 0;
        background:#161616;
        font-family: 'Open Sans', sans-serif;
    }
    iframe {
        min-height: 100%;
        height: 100%;
        padding: 0;
    }
}
```

```

h1 {
    color: #CBCBCB;
    padding: 1%;
    font-size: 50%;
}
p {
    color: #CBCBCB;
    padding: 1%;
}
table {
    color: #CBCBCB;
    padding: 1%;
    font-size: 100%;
    background:#252525;
}
a:visited {
    color: #CBCBCB;
}
a {
    color: #CBCBCB;
}
}
@media screen and (min-width:769px){
    html,
    body{
        min-height: 100%;
        height: 100%;
        padding: 0;
        background:#161616;
        font-family: 'Open Sans', sans-serif;
    }
    iframe {
        min-height: 100%;
        height: 100%;
        padding: 0;
    }
    h1 {
        color: #CBCBCB;
        padding: 1%;
        font-size: 50%;
    }
    p {
        color: #CBCBCB;
        padding: 1%;
    }
    table {
        color: #CBCBCB;
        padding: 1%;
        font-size: 100%;
        background:#252525;
    }
    a:visited {
        color: #CBCBCB;
    }

```

```

    }
    a {
        color: #CBCBCB;
    }
}
</style>
</head>
<body>
    <table border=1 width=100%>
        <tr>
            <td width=25% height=40px><center><a href="wolremote/wol.sol.html"
target="IScreen">REMOTE</a></center></td>
            <td width=25% height=40px><center><a href="vguests/vguests.html"
target="IScreen">VGUESTS</a></center></td>
            <td width=25% height=40px><center><a href="machines/machines.html"
target="IScreen">MACHINE</a></center></td>
            <td width=25% height=40px><center><a href="check/index.html"
target="IScreen">WEBLOG</a></center></td>
        </tr>
    </table>
    <iframe id="indexframe" name="IScreen" src="wolremote/wol.sol.html" min-
width="100%" width="100%" frameborder="0"></iframe>
</body>
</html>
[/CODE]

```

Die eigentliche Steuerseite, die dort eingebettet ist, hat im Body:

```

[CODE=HTML]
<body>
    <h1><center>LAN Remote Console</center></h1>
    <iframe name="WOLPHPScript" src="wol/wol.php" width="100%" height="420px"
frameborder="0"></iframe>
    <table border=1 width=100%>
        <tr>
            <td width=25% height=50px><center>PC 01</center></td>
            <td width=25% height=50px><center><a href="http://192.168.1.2:8009/halt"
target="WOLScreen">halt</a></center></td>
            <td width=25% height=50px><center><a href="http://192.168.1.2:8009/restart"
target="WOLScreen">restart</a></center></td>
            <td width=25% height=50px><center><a href="http://192.168.1.2:8009/rest"
target="WOLScreen">logout</a></center></td>
        </tr>
        <tr>
            <td width=25% height=50px><center>PC 02</center></td>
            <td width=25% height=50px><center><a href="http://192.168.1.4:8009/halt"
target="WOLScreen">halt</a></center></td>
            <td width=25% height=50px><center><a href="http://192.168.1.4:8009/restart"
target="WOLScreen">restart</a></center></td>
            <td width=25% height=50px><center><a href="http://192.168.1.4:8009/rest"
target="WOLScreen">logout</a></center></td>
        </tr>
    </table>

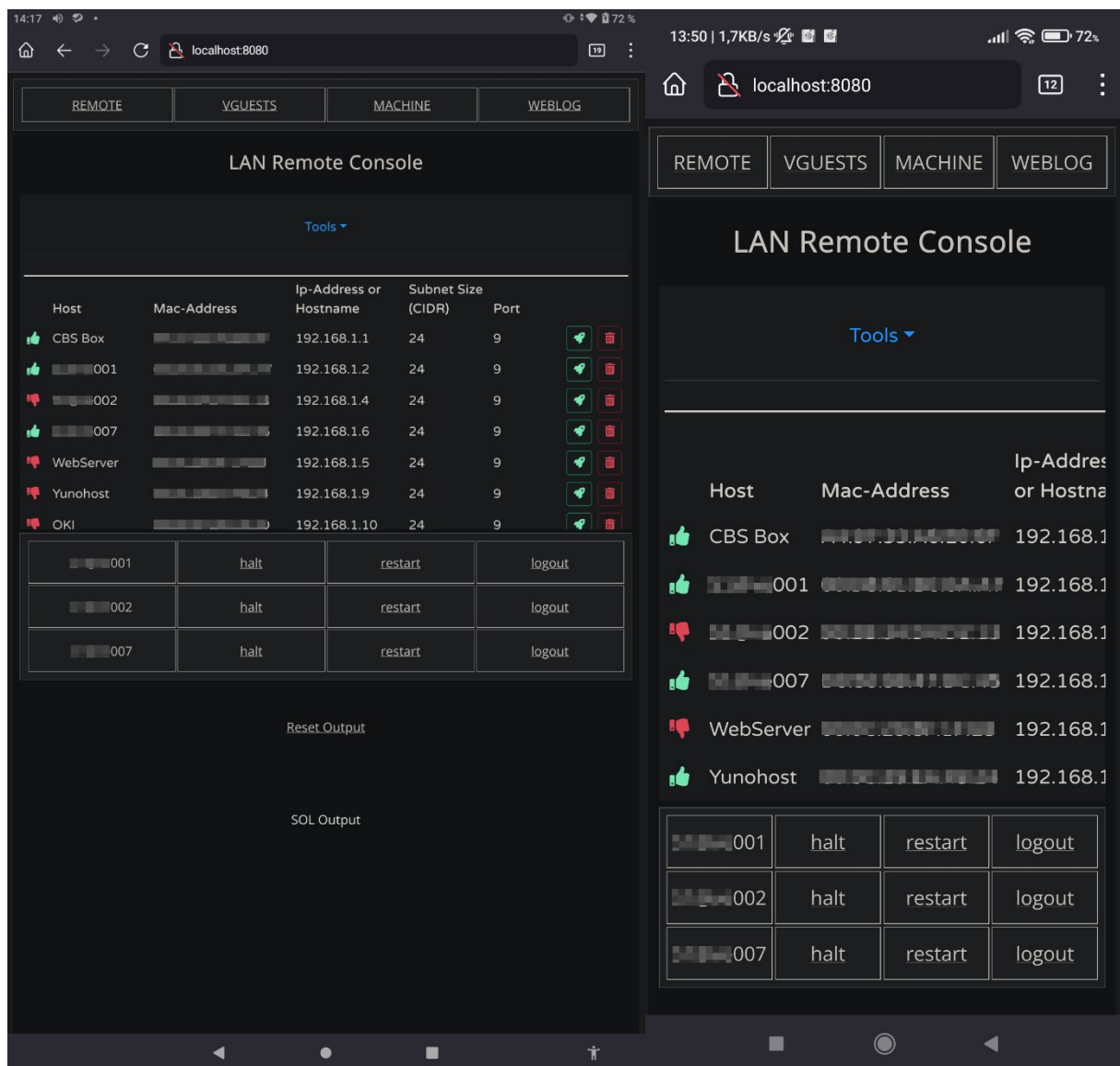
```

```

        <td width=25% height=50px><center>PC 03</center></td>
        <td width=25% height=50px><center><a href="http://192.168.1.6:8009/halt"
target="WOLScreen">halt</a></center></td>
        <td width=25% height=50px><center><a href="http://192.168.1.6:8009/restart"
target="WOLScreen">restart</a></center></td>
        <td width=25% height=50px><center><a href="http://192.168.1.6:8009/rest"
target="WOLScreen">logout</a></center></td>
    </tr>
</table>
<p><center><a href="wol/iframe.html" target="WOLScreen">Reset
Output</center></p>
    <iframe id="wolframe" name="WOLScreen" src="wol/iframe.html" width="100%"
height="400px" frameborder="0"></iframe>
</body> [/CODE]

```

Ergebnis auf Smartphone und Tablet:



Auf die Art kann die SOL Binary gesteuert werden, auch wenn ein Passwort verwendet wird. In dem Fall muß bei jeder Browser-Session Username und Passwort eingegeben werden. Ich automatisiere und synchronisiere das mit KeepPass über alle Systeme. Dank statischer IPs im LAN kein Problem.

Damit läßt sich einiges realisieren. Sowohl SOL als auch das Skript haben sich als äußerst robust erwiesen.

Das System läßt sich zudem einfach verändern und erweitern. Per SSH den Task beenden, die veränderten Dateien in die Freigabe kopieren und den Task wieder starten. Mission accomplished.

Anhang Part 2

Mint Installation und Vorbereitung:

Basis ist eine VMware VM mit 20 GB Disk (dynamisch), 4 GB RAM und 2 CPU-Cores. Das ist ausreichend performant, das Guacamole-System ist recht genügsam. Hat zudem den Vorteil der einfachen Snapshot-Erstellung. Es ist einfacher, bei einem Fehler, den letzten Snapshot wiederherzustellen, als ein verpfushtes System zu kurieren.

Livesystem gebootet und den Installer-Link auf dem Desktop gestartet. Die Installation ist selbsterklärend und recht schnell erledigt. Ein Neustart später und ein Login taucht auch schon der Xfce-Desktop auf. Schlicht und für unsere Zwecke sehr komfortabel.

Für noch mehr Komfort empfiehlt sich die Installation der VMware Tools. Das erklärt folgender Link recht gut: <https://kb.vmware.com/s/article/1022525>. Mint erkennt das gemountete ISO mit der TAR.GZ Datei automatisch und öffnet einen Ordner wie Windows, der Rest geht aus dem Link hervor.

Danach noch mit:

```
[CODE=Bash]
sudo apt update && sudo apt upgrade -y
sudo apt install open-vm-tools open-vm-tools-desktop
[/CODE]
```

die Open-VM-Tools installieren – nun können Dateien per Drag ´n Drop getauscht und Befehle in der Mint-CLI eingefügt werden. Das spart Eingabefehler und Nerven beim Transfer der nötigen Dateien.

Docker Installation:

APT-Package-Index updaten und Stand der Packages zur Nutzung eines Repositories über HTTPS checken mit:

```
[CODE=Bash]
sudo apt-get update
sudo apt-get install \
    ca-certificates \
    curl \
    gnupg \
    lsb-release
[/CODE]
```

Den offiziellen Docker-GPG-Schlüssel hinzufügen:

```
[CODE=Bash]
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o
/usr/share/keyrings/docker-archive-keyring.gpg
[/CODE]
```

Stabiles Repository aufsetzen:

```
[CODE=Bash]
echo \
    "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/docker-
archive-keyring.gpg] https://download.docker.com/linux/ubuntu \
    $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
[/CODE]
```

APT-Package-Index updaten und aktuelle Dockerversion installieren:

```
[CODE=Bash]
sudo apt update
sudo apt install docker*
[/CODE]
```

Docker-Compose Installation:

Dazu folgende Befehle abarbeiten:

```
[CODE=Bash]
sudo curl -L "https://github.com/docker/compose/releases/download/1.27.4/docker-
compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
sudo chmod +x /usr/local/bin/docker-compose
docker-compose --version
sudo apt install net-tools
[/CODE]
```

Anhang Part 3

Trafik2-Docker Installation:

Wenn CloudFlare statt DuckDNS verwendet werden soll, im Tutorial auf jeden die Punkte 3. Und 4. checken. Für DuckDNS kann mein Beispiel als Referenz genutzt werden, das hat gut funktioniert.

Unter *Setting Up Environmental Variables for Docker and Docker Compose* die .ENV Datei erstellen, das erleichtert alle späteren Schritte.

Diese dient als Basis für spätere Erweiterungen und für einige der CLI-Kommandos.

Interessant wird es nun unter *Prep Work for Traefik 2*.

Für DuckDNS sollten die Einträge in der .ENV wie folgt aussehen:

```
[CODE=PowerShell]
DOMAINNAME=subdomain.duckdns.org
DUCKDNS_EMAIL=mymail@gmail.com
DUCKDNS_TOKEN=TOKEN-GOES-HERE
[/CODE]
```

Hier die eigenen Daten setzen.

Sind die Pfadvariablen in der .ENV gesetzt, können die CLI-Kommandos meist ohne Anpassung genutzt werden, ansonsten ist da ein Hinweis.

Nach der Ersteinrichtung von Traefik2 sieht meine YAML so aus:

```
[CODE=PowerShell]
version: "3.7"

##### NETWORKS
networks:
  t2_proxy:
    external:
      name: t2_proxy
  default:
    driver: bridge

##### SERVICES
services:
# All services / apps go below this line

# Traefik 2 - Reverse Proxy
traefik:
  container_name: traefik
  image: traefik:2.2.1 # the chevrotin tag refers to v2.2.x but introduced a breaking change
in 2.2.2
  restart: unless-stopped
  command: # CLI arguments
    - --global.checkNewVersion=true
    - --global.sendAnonymousUsage=true
    - --entryPoints.http.address=:80
    - --entryPoints.https.address=:443
    # Allow these IPs to set the X-Forwarded-* headers - Cloudflare IPs:
    https://www.cloudflare.com/ips/
    - --
  entrypoints.https.forwardedHeaders.trustedIPs=173.245.48.0/20,103.21.244.0/22,103.22.20
0.0/22,103.31.4.0/22,141.101.64.0/18,108.162.192.0/18,190.93.240.0/20,188.114.96.0/20,1
```

97.234.240.0/22,198.41.128.0/17,162.158.0.0/15,104.16.0.0/12,172.64.0.0/13,131.0.72.0/22

```
- --entryPoints.traefik.address=:8080
- --api=true
# - --api.insecure=true
# - --serversTransport.insecureSkipVerify=true
- --log=true
- --log.level=DEBUG # (Default: error) DEBUG, INFO, WARN, ERROR, FATAL, PANIC
- --accessLog=true
- --accessLog.filePath=/traefik.log
- --accessLog.bufferingSize=100 # Configuring a buffer of 100 lines
- --accessLog.filters.statusCodes=400-499
- --providers.docker=true
- --providers.docker.endpoint=unix:///var/run/docker.sock
- --providers.docker.defaultRule=Host(`{{ index .Labels "com.docker.compose.service"
}}.$DOMAINNAME`)
- --providers.docker.exposedByDefault=false
- --providers.docker.network=t2_proxy
- --providers.docker.swarmMode=false
- --providers.file.directory=/rules # Load dynamic configuration from one or more .toml
or .yaml files in a directory.
# - --providers.file.filename=/path/to/file # Load dynamic configuration from a file.
- --providers.file.watch=true # Only works on top level files in the rules folder
# - --certificatesResolvers.duckdns.acme.caServer=https://acme-staging-
v02.api.letsencrypt.org/directory # LetsEncrypt Staging Server - uncomment when testing
- --certificatesResolvers.duckdns.acme.email=$DUCKDNS_EMAIL
- --certificatesResolvers.duckdns.acme.storage=/acme.json
- --certificatesResolvers.duckdns.acme.dnsChallenge.provider=duckdns
- --certificatesResolvers.duckdns.acme.dnsChallenge.resolvers=1.1.1.1:53,1.0.0.1:53
# networks:
#   t2_proxy:
#     ipv4_address: 192.168.90.254 # You can specify a static IP
networks:
- t2_proxy
security_opt:
- no-new-privileges:true
ports:
- target: 80
  published: 80
  protocol: tcp
  mode: host
- target: 443
  published: 443
  protocol: tcp
  mode: host
- target: 8080
  published: 8080
  protocol: tcp
  mode: host
volumes:
- $DOCKERDIR/traefik2/rules:/rules
- /var/run/docker.sock:/var/run/docker.sock:ro
- $DOCKERDIR/traefik2/acme/acme.json:/acme.json
```

```

- $DOCKERDIR/traefik2/traefik.log:/traefik.log
- $DOCKERDIR/shared:/shared
environment:
- DUCKDNS_TOKEN=$DUCKDNS_TOKEN
labels:
- "traefik.enable=true"
# HTTP-to-HTTPS Redirect
- "traefik.http.routers.http-catchall.entrypoints=http"
- "traefik.http.routers.http-catchall.rule=HostRegexp(`{host:.+}`)"
- "traefik.http.routers.http-catchall.middlewares=redirect-to-https"
- "traefik.http.middlewares.redirect-to-https.redirectscheme.scheme=https"
# HTTP Routers
- "traefik.http.routers.traefik-rtr.entrypoints=https"
- "traefik.http.routers.traefik-rtr.rule=Host(`traefik.$DOMAINNAME`)"
- "traefik.http.routers.traefik-rtr.tls=true"
# - "traefik.http.routers.traefik-rtr.tls.certresolver=duckdns" # Comment out this line
after first run of traefik to force the use of wildcard certs
- "traefik.http.routers.traefik-rtr.tls.domains[0].main=$DOMAINNAME"
- "traefik.http.routers.traefik-rtr.tls.domains[0].sans=*. $DOMAINNAME"
# - "traefik.http.routers.traefik-rtr.tls.domains[1].main=$SECONDDOMAINNAME" # Pulls
main cert for second domain
# - "traefik.http.routers.traefik-rtr.tls.domains[1].sans=*. $SECONDDOMAINNAME" #
Pulls wildcard cert for second domain
## Services - API
- "traefik.http.routers.traefik-rtr.service=api@internal"
## Middlewares
- "traefik.http.routers.traefik-rtr.middlewares=traefik-headers,middlewares-rate-
limit@file,middlewares-basic-auth@file"
- "traefik.http.middlewares.traefik-headers.headers.accesscontrolallowmethods=GET,
OPTIONS, PUT"
- "traefik.http.middlewares.traefik-
headers.headers.accesscontrolalloworiginlist=https://$DOMAINNAME"
- "traefik.http.middlewares.traefik-headers.headers.accesscontrolmaxage=100"
- "traefik.http.middlewares.traefik-headers.headers.addvaryheader=true"
- "traefik.http.middlewares.traefik-
headers.headers.allowedhosts=traefik.$DOMAINNAME"
- "traefik.http.middlewares.traefik-headers.headers.hostsproxyheaders=X-Forwarded-
Host"
- "traefik.http.middlewares.traefik-headers.headers.sslredirect=true"
- "traefik.http.middlewares.traefik-headers.headers.sslhost=traefik.$DOMAINNAME"
- "traefik.http.middlewares.traefik-headers.headers.sslforcehost=true"
- "traefik.http.middlewares.traefik-headers.headers.sslproxyheaders.X-Forwarded-
Proto=https"
- "traefik.http.middlewares.traefik-headers.headers.stsseconds=63072000"
- "traefik.http.middlewares.traefik-headers.headers.stsincludesubdomains=true"
- "traefik.http.middlewares.traefik-headers.headers.stspreload=true"
- "traefik.http.middlewares.traefik-headers.headers.forcestsheader=true"
- "traefik.http.middlewares.traefik-headers.headers.framedeny=true"
# - "traefik.http.middlewares.traefik-
headers.headers.customframeoptionsvalue=SAMEORIGIN" # This option overrides
FrameDeny
- "traefik.http.middlewares.traefik-headers.headers.contenttypenosniff=true"
- "traefik.http.middlewares.traefik-headers.headers.browserxssfilter=true"

```

```
# - "traefik.http.middlewares.traefik-headers.headers.contentsecuritypolicy=frame-
ancestors 'none'; object-src 'none'; base-uri 'none';"
- "traefik.http.middlewares.traefik-headers.headers.referrerpolicy=same-origin"
- "traefik.http.middlewares.traefik-headers.headers.featurepolicy=camera 'none';
geolocation 'none'; microphone 'none'; payment 'none'; usb 'none'; vr 'none';"
- "traefik.http.middlewares.traefik-headers.headers.customresponseheaders.X-Robots-
Tag=none,noarchive,nosnippet,notranslate,noimageindex,"
[/CODE]
```

Die middlewares.toml

```
[CODE=PowerShell]
[http.middlewares]
  [http.middlewares.middlewares-basic-auth]
    [http.middlewares.middlewares-basic-auth.basicAuth]
#     users = [
#       "admin:$apr1$9jd4DjW.$szNFe1uJy.27HqR1J3CZe0",
#     ]
    realm = "Traefik2 Basic Auth"
    usersFile = "/shared/.htpasswd" #be sure to mount the volume through docker-
compose.yml
```

```
[http.middlewares.middlewares-rate-limit]
  [http.middlewares.middlewares-rate-limit.rateLimit]
    average = 100
    burst = 50
```

```
# Available Header Options:
#####https://github.com/unrolled/secure#available-options
#####https://docs.traefik.io/middlewares/headers/
# A great resource for these headers is your preferred browser's docs. Firefox:
https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers
# https://developers.google.com/search/reference/robots_meta_tag
#
https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/Clickjacking_Defense_Cheat_Sheet.md
# CSP for VNC: https://github.com/cockpit-project/cockpit/pull/5932
# Check headers here, don't include OAuth when checking headers, otherwise you are
checking google's headers: https://securityheaders.com
# or check them here: https://observatory.mozilla.org/
```

CAUTION: Any headers defined in docker-compose (yml) will OVERWRITE ALL of the headers defined below.

```
[http.middlewares.middlewares-secure-headers]
  [http.middlewares.middlewares-secure-headers.headers]
    accessControlAllowMethods= ["GET", "OPTIONS", "PUT"]
    accessControlMaxAge = 100
    hostsProxyHeaders = ["X-Forwarded-Host"]
    # sslRedirect = true #replaced with middlewares-https-redirectscheme for v2.5.x
    stsSeconds = 63072000
    stsIncludeSubdomains = true
    stsPreload = true
    forceSTSHeader = true
```

```

#   frameDeny = true #overwritten by customFrameOptionsValue
    customFrameOptionsValue = "allow-from https:nextcloud.s1ave77.duckdns.org" #CSP
takes care of this but may be needed for organizr.
    contentTypeNosniff = true
    browserXssFilter = true
#   sslForceHost = true # add sslHost and all of the
#   sslHost = "example.com"
    referrerPolicy = "same-origin"
#   Setting contentSecurityPolicy is more secure but it can break things. Proper auth will
reduce the risk.
#   the below line also breaks some apps due to 'none' - sonarr, radarr, etc.
#   contentSecurityPolicy = "frame-ancestors '*.example.com:>';object-src 'none';script-src
'none';"
    # Line below, featurePolicy, was deprecated in v2.5.x in favor permissionPolicy
    # featurePolicy = "camera 'none'; geolocation 'none'; microphone 'none'; payment
'none'; usb 'none'; vr 'none';"
    permissionsPolicy = "camera=(), microphone=(), geolocation=(), payment=(), usb=(),
vr=()"
    [http.middlewares.middlewares-secure-headers.headers.customResponseHeaders]
    X-Robots-Tag = "none,noarchive,nosnippet,notranslate,noimageindex,"
    server = ""

[http.middlewares.middlewares-oauth]
[http.middlewares.middlewares-oauth.forwardAuth]
    address = "http://oauth:4181" # Make sure you have the OAuth service in docker-
compose.yml
    trustForwardHeader = true
    authResponseHeaders = ["X-Forwarded-User"]

[http.middlewares.middlewares-authelia]
[http.middlewares.middlewares-authelia.forwardAuth]
    address = "http://authelia:9091/api/verify?rd=https://authelia.example.com"
    trustForwardHeader = true
    authResponseHeaders = ["Remote-User", "Remote-Groups"]

### Let's give them a new name so it won't conflict with others
[http.middlewares.nextcloud-middlewares-secure-headers]
### Change the name here as well
[http.middlewares.nextcloud-middlewares-secure-headers.headers]
### Comment out this line as Nextcloud uses quite a few methods for different apps
#   accessControlAllowMethods= ["GET", "OPTIONS", "PUT"]
    accessControlMaxAge = 100
    hostsProxyHeaders = ["X-Forwarded-Host"]
    sslRedirect = true #replaced with middlewares-https-redirectscheme for v2.5.x
    stsSeconds = 63072000
    stsIncludeSubdomains = true
    stsPreload = true
    forceSTSHheader = true
#   frameDeny = true #overwritten by customFrameOptionsValue
### We will modify this value for Nextcloud to remove the X-Frame-Options error:
    customFrameOptionsValue = "SAMEORIGIN" #CSP takes care of this but may be needed
for organizr.
    contentTypeNosniff = true

```

```

    browserXssFilter = true
#    sslForceHost = true # add sslHost to all of the services
#    sslHost = "example.com"
    referrerPolicy = "same-origin"
#    Setting contentSecurityPolicy is more secure but it can break things. Proper auth will
reduce the risk.
#    the below line also breaks some apps due to 'none' - sonarr, radarr, etc.
#    contentSecurityPolicy = "frame-ancestors '*.example.com:*';object-src 'none';script-src
'none';"
    # Line below, featurePolicy, was deprecated in v2.5.x in favor permissionPolicy
    # featurePolicy = "camera 'none'; geolocation 'none'; microphone 'none'; payment
'none'; usb 'none'; vr 'none';"
    permissionsPolicy = "camera=(), microphone=(), geolocation=(), payment=(), usb=(),
vr=()"
    ### Change the middleware name here as well
    [http.middlewares.nextcloud-middlewares-secure-
headers.headers.customResponseHeaders]
    ### We just need to set this to none
    X-Robots-Tag = "none"
    server = ""

### This section redirects requests for Nextcloud calendar and contacts service discovery
### source:
https://docs.nextcloud.com/server/21/admin\_manual/issues/general\_troubleshooting.html#s
ervice-discovery
    [http.middlewares.nextcloud-redirect]
    [http.middlewares.nextcloud-redirect.redirectRegex]
    permanent = true
    regex = "https://(.*)/.well-known/(card|cal)dav"
    replacement = "https://${1}/remote.php/dav/" [/CODE]

```

Die middlewares-chains.toml

```

[CODE=PowerShell]
[http.middlewares]
    [http.middlewares.chain-no-auth]
    [http.middlewares.chain-no-auth.chain]
    middlewares = [ "middlewares-rate-limit", "middlewares-secure-headers"]

    [http.middlewares.chain-basic-auth]
    [http.middlewares.chain-basic-auth.chain]
    middlewares = [ "middlewares-rate-limit", "middlewares-secure-headers", "middlewares-
basic-auth"]

    [http.middlewares.chain-oauth]
    [http.middlewares.chain-oauth.chain]
    middlewares = [ "middlewares-rate-limit", "middlewares-secure-headers",
"middlewares-oauth"]

    [http.middlewares.chain-authelia]
    [http.middlewares.chain-authelia.chain]
    middlewares = [ "middlewares-rate-limit", "middlewares-secure-headers", "middlewares-
authelia"]

```



```
[http.middlewares.chain-nextcloud]
[http.middlewares.chain-nextcloud.chain]
  middlewares = [ "middlewares-rate-limit", "middlewares-https-redirectscheme",
"nextcloud-middlewares-secure-headers", "nextcloud-redirect" ] [/CODE]
```

Endergebnis sollte ein laufender Traefik2-Proxy sein.

WICHTIG!: An diesem Punkt sollte das Traefik2-Dashboard unter:

<https://traefik.subdomain.duckdns.org/dashboard/>

erreichbar sein. Das ist äußerst nützlich, da es übersichtlich zeigt, was funktioniert und was nicht. Hat mir geholfen, kleine Konfigurationsfehler zu finden. Syntax ist ‚König‘!

Wenn alles läuft geht es mit *Google OAuth 2.0* weiter.

Die .ENV Einträge:

```
[CODE=PowerShell]
GOOGLE_CLIENT_ID=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx.apps.g
oogleusercontent.com
GOOGLE_CLIENT_SECRET=CLIENT-SECRET-HERE
OAUTH_SECRET=SECRET-HERE
MY_EMAIL=mymail@gmail.com
URL_PATH=https://oauth.subdomain.duckdns.org/_oauth
[/CODE]
```

Die TOMLs bleiben und der YAML Part:

```
[CODE=PowerShell]
# Google OAuth - Single Sign On using OAuth 2.0
oauth:
  container_name: oauth
  image: thomseddon/traefik-forward-auth:latest
  restart: unless-stopped
  networks:
    - t2_proxy
  security_opt:
    - no-new-privileges:true
  environment:
    - CLIENT_ID=$GOOGLE_CLIENT_ID
    - CLIENT_SECRET=$GOOGLE_CLIENT_SECRET
    - SECRET=$OAUTH_SECRET
    - COOKIE_DOMAIN=$DOMAINNAME
    - INSECURE_COOKIE=false
    - AUTH_HOST=oauth.$DOMAINNAME
    - URL_PATH=/_oauth
    - WHITELIST=$MY_EMAIL
    - LOG_LEVEL=info
    - LOG_FORMAT=text
    - LIFETIME=2592000 # 30 days
```

labels:

- "traefik.enable=true"
- ## HTTP Routers
- "traefik.http.routers.oauth-rtr.entrypoints=https"
- "traefik.http.routers.oauth-rtr.rule=Host(`oauth.\$DOMAINNAME`)"
- "traefik.http.routers.oauth-rtr.tls=true"
- ## HTTP Services
- "traefik.http.routers.oauth-rtr.service=oauth-svc"
- "traefik.http.services.oauth-svc.loadbalancer.server.port=4181"
- ## Middlewares
- "traefik.http.routers.oauth-rtr.middlewares=chain-oauth@file" [/CODE]

Gesetzt den Fall, alles läuft, ist das Schwierigste geschafft

Die Nutzung der Traefik2-Middlewares ermöglicht nun die einfache Einbindung weiterer Container.

Erste Anlaufstelle ist hier immer das Traefik2-Dashboard, das zeigt den Erfolg.

Weitere Variablen können in der .ENV referenziert werden oder direkt in der YAML.

Portainer – Container-Management:

Ist eine komfortable Lösung um einzelne Container schnell (neu)starten oder beenden zu können.

Am Skript gibt es bei unserer Konstellation nichts anzupassen.

[CODE=PowerShell]

Portainer - WebUI for Containers

portainer:

container_name: portainer

image: portainer/portainer-ce:latest

restart: unless-stopped

command: -H unix:///var/run/docker.sock

networks:

- t2_proxy

security_opt:

- no-new-privileges:true

ports:

- "\$PORTAINER_PORT:9000"

volumes:

- /var/run/docker.sock:/var/run/docker.sock:ro

- \$DOCKERDIR/portainer/data:/data

environment:

- TZ=\$TZ

labels:

- "traefik.enable=true"

HTTP Routers

- "traefik.http.routers.portainer-rtr.entrypoints=https"

- "traefik.http.routers.portainer-rtr.rule=Host(`portainer.\$DOMAINNAME`)"

- "traefik.http.routers.portainer-rtr.tls=true"

Middlewares

```
# - "traefik.http.routers.portainer-rtr.middlewares=chain-no-auth@file" # No
Authentication
# - "traefik.http.routers.portainer-rtr.middlewares=chain-basic-auth@file" # Basic
Authentication
- "traefik.http.routers.portainer-rtr.middlewares=chain-oauth@file" # Google OAuth 2.0
## HTTP Services
- "traefik.http.routers.portainer-rtr.service=portainer-svc"
- "traefik.http.services.portainer-svc.loadbalancer.server.port=9000"
[/CODE]
```

Die UI findet sich dann unter: <https://portainer.subdomain.duckdns.org>

Einrichtung erfolgt beim ersten Start.

Organizr – Universelles Interface für alle Container:

Erlaubt den Zugriff auf alle installierten Apps über eine Oberfläche. Ausgenommen sind einige Apps, wie Nextcloud, die das Laden in iFrames aus Sicherheitsgründen unterbinden. Erübrigt aber das Eingeben der einzelnen Sub-Domains in die Browserleiste.

Auch hier nichts anzupassen:

```
[CODE=PowerShell]
# Organizr - Unified Frontend
organizr:
  container_name: organizr
  image: organizr/organizr:latest
  restart: unless-stopped
  networks:
    - t2_proxy
  security_opt:
    - no-new-privileges:true
# ports:
# - "$ORGANIZR_PORT:80"
volumes:
  - $DOCKERDIR/organizr:/config
environment:
  - PUID=$PUID
  - PGID=$PGID
  - TZ=$TZ
labels:
  - "traefik.enable=true"
## HTTP Routers
- "traefik.http.routers.organizr-rtr.entrypoints=https"
- "traefik.http.routers.organizr-
rtr.rule=Host(`$DOMAINNAME`, `www.$DOMAINNAME`)"
- "traefik.http.routers.organizr-rtr.tls=true"
## Middlewares
- "traefik.http.routers.organizr-rtr.middlewares=chain-oauth@file"
## HTTP Services
- "traefik.http.routers.organizr-rtr.service=organizr-svc"
- "traefik.http.services.organizr-svc.loadbalancer.server.port=80"
```

[/CODE]

ACHTUNG: Als Verwaltungs-Seite wird Organizr (wie auch Heimdall als Alternative) über die Hauptdomain angesprochen; also <https://subdomain.duckdns.org>.

Einrichtung erfolgt beim ersten Start.

MariaDB – MySQL-Datenbank:

Im Tutorial wird eine statische IP-Konfiguration verwendet, daher in meinem Code die dynamische Variante.

In der .ENV:

```
[CODE=PowerShell]
MYSQL_ROOT_PASSWORD="password"
[/CODE]
```

das Passwort anpassen.

Der YAML Part:

```
[CODE=PowerShell]
# MariaDB - MySQL Database
mariadb:
  container_name: mariadb
  image: linuxserver/mariadb:latest
  restart: always
  networks:
    - t2_proxy
  security_opt:
    - no-new-privileges:true
  ports:
    - "3306:3306"
  volumes:
    - $DOCKERDIR/mariadb/data:/config
    - /etc/timezone:/etc/timezone:ro
    - /etc/localtime:/etc/localtime:ro
  environment:
    - PUID=$PUID
    - PGID=$PGID
    - MYSQL_ROOT_PASSWORD=$MYSQL_ROOT_PASSWORD
[/CODE]
```

Adminer – MariaDB Management:

Ohne weitere Anpassungen übernehmbar:

```
[CODE=PowerShell]
# Adminer - Database management
```

```

adminer:
  image: adminer:standalone
  container_name: adminer
  restart: unless-stopped
  networks:
    - t2_proxy
  security_opt:
    - no-new-privileges:true
  volumes:
    - $DOCKERDIR/adminer:/etc/adminer
  depends_on:
    - mariadb
  links:
    - mariadb:db
  labels:
    - "traefik.enable=true"
    ## HTTP Routers
    - "traefik.http.routers.adminer-rtr.entrypoints=https"
    - "traefik.http.routers.adminer-rtr.rule=Host(`adminer.${DOMAINNAME}`)"
    - "traefik.http.routers.adminer-rtr.tls=true"
    ## Middlewares
    - "traefik.http.routers.adminer-rtr.middlewares=chain-oauth@file"
    ## HTTP Services
    - "traefik.http.routers.adminer-rtr.service=adminer-svc"
    - "traefik.http.services.adminer-svc.loadbalancer.server.port=8080"
[/CODE]

```

Apache Guacamole – Remote Desktop via HTML5:

An dieser Stelle vorgezogen um das Anlegen einer Datenbank in MariaDB zu demonstrieren. Ich nutze ebenfalls den CLI-Weg.

Dazu in den Ordner navigieren, die Init-Datei anlegen, in den entsprechenden Docker-Ordner kopieren und die SQL-Shell starten:

```

[CODE=PowerShell]
cd ${USERDIR}/docker
docker run --rm guacamole/guacamole /opt/guacamole/bin/initdb.sh --mysql >
guac_initdb.sql
sudo cp ${USERDIR}/docker/guac_initdb.sql ${USERDIR}/docker/mariadb/data
docker exec -ti mariadb /bin/bash
[/CODE]

```

In den Root-Account einloggen:

```

[CODE=PowerShell]
mysql -u root -p
[/CODE]

```

Datenbasis und User anlegen, Rechte zu weisen, neustarten und verlassen (username und password anpassen):

```
[CODE=PowerShell]
create database guacamole;
CREATE USER 'username' IDENTIFIED BY 'password';
GRANT ALL ON `guacamole%`.`.* TO 'username';
flush privileges;
quit
[/CODE]
```

MySQL-Bash nicht verlassen! Sondern mit:

```
[CODE=PowerShell]
cat /config/guac_initdb.sql | mysql -u username -p guacamole;
[/CODE]
```

Das vergebene Passwort eintragen und starten.

In den neuen User_account einloggen und verlassen zum Test der Einträge mit:

```
[CODE=PowerShell]
mysql -u username -p
use guacamole;
show tables;
quit
[/CODE]
```

Wie es aussehen sollte, sieht man in dem Link: <https://www.smarthomebeginner.com/install-guacamole-on-docker/>.

Die Daten lassen sich wieder in der .ENV speichern, ich habe sie direkt in der YAML genutzt, da Guacamole da heikel reagiert hat. Sicher ist sicher.

An dieser Stelle empfehle ich direkt die Integration der TOTP-Extensions, Guacamole ermöglicht weitreichenden Zugriff, das sollte gesichert sein.

3 Ordner im Dockerverzeichnis erstellen, dem Konfig-Ordner Rechte geben, guacamole.properties Datei erstellen mit:

```
[CODE=PowerShell]
mkdir ${USERDIR}/guacamole-docker/guachome
mkdir ${USERDIR}/docker/guachome/extensions
mkdir ${USERDIR}/docker/extensions
chmod -R 757 ${USERDIR}/docker/guachome
nano ${USERDIR}/docker/guachome/guacamole.properties
[/CODE]
```

Dateiinhalt:

```
[CODE=PowerShell]
# guacamole.properties
enable-environment-properties: true
http-auth-header: REMOTEUSER
[/CODE]
```

Speichern mit Strg+O und Enter, dann verlassen mit Strg+X.

Die benötigten Extensions als TAR.GZ gibt es hier: <https://guacamole.apache.org/releases/>

Diese gehen direkt als JARs nach: \${USERDIR}/docker/extensions.

ich habe die TAR.GZ Archive mit Total Commander unter Windows entpackt, gibt aber auch genug Anleitungen, wie es unter Linux funktioniert.

Die YAML enthält alle Einträge für die Integration.

[CODE=PowerShell]

Guacamole Daemon - Needed for Guacamole

guacd:

- image: guacamole/guacd
- container_name: guacd
- restart: unless-stopped
- security_opt:
 - no-new-privileges:true
- networks:
 - t2_proxy

Guacamole - Remote desktop, SSH, on Telnet on any HTML5 Browser

guacamole:

- image: guacamole/guacamole:latest
- container_name: guacamole
- restart: unless-stopped
- networks:
 - t2_proxy
- security_opt:
 - no-new-privileges:true
- depends_on:
 - mariadb
- environment:
 - GUACD_HOSTNAME: guacd
 - MYSQL_HOSTNAME: mariadb
 - MYSQL_PORT: 3306
 - MYSQL_DATABASE: guacamole
 - MYSQL_USER: username
 - MYSQL_PASSWORD: 'password'
 - GUACAMOLE_HOME: /guachome
- volumes:
 - ./guachome:/guachome
 - ./extensions:/guachome/extensions
- labels:
 - "traefik.enable=true"
 - ## HTTP Routers
 - "traefik.http.routers.guacamole-rtr.entrypoints=https"
 - "traefik.http.routers.guacamole-rtr.rule=Host(`guac.\$DOMAINNAME`)"
 - "traefik.http.routers.guacamole-rtr.tls=true"
 - ## Middlewares
 - "traefik.http.routers.guacamole-rtr.middlewares=chain-oauth@file,add-guacamole"
 - "traefik.http.middlewares.add-guacamole.addPrefix.prefix=/guacamole"

```
## HTTP Services
- "traefik.http.routers.guacamole-rtr.service=guacamole-svc"
- "traefik.http.services.guacamole-svc.loadbalancer.server.port=8080"
[/CODE]
```

Hier gilt es username und password anzupassen. Achtung mit den Anführungszeichen beim Passwort, die sind wichtig, wie ich lernen durfte!

Der Rest kann übernommen werden und passt.

UI ist über <https://guac.subdomain.duckdns.org> erreichbar. Login ist 2x *guacadmin*.

Ich empfehle die Ersteinrichtungstipps im weiter oben genannten Tutorial zu nutzen, das erspart Frust und sichert das System. Gerade den Standard-Admin gilt es zu löschen.

Nextcloud – Local Cloud Storage:

Hier bleibt das Tutorial vage aber ich konnte es mir erfolgreich zusammenreimen.

Update: Mit Hilfe von @LieberNetterFloh und einem Post im Nextcloud Forum hier eine neue Nextcloud Instanz. Für WebDav-Zugriff kann kein OAuth benutzt werden, aber die angepassten Middlewares übernehmen die Basissicherheit. Der integrierte Zugriffsschutz und zusätzlich TOTP machen einen guten Job. Zusätzlich läßt sich Fail2Ban konfigurieren, die Logs zu überwachen.

Ich habe für Nextcloud diesmal eine MariaDB Datenbank erstellt, wie im Beispiel von Guacamole gezeigt.

Keine Bange, mein Beispiel trägt dem Rechnung. Die Middlewares TOMLs bleiben unverändert.

```
[CODE=PowerShell]
# Nextcloud
nextcloud:
  container_name: nextcloud
  image: nextcloud:latest
  restart: unless-stopped
  networks:
    - t2_proxy
  security_opt:
    - no-new-privileges:true
#   ports:
#     - 8080:80
  volumes:
    - $DOCKERDIR/nextcloud/data:/var/www/html
  depends_on:
    - mariadb
  links:
    - mariadb:db
  environment:
    - PUID=$PUID
    - PGID=$PGID
    - TZ=$TZ
```


labels:

```
- "traefik.enable=true"
## HTTP Routers
- "traefik.http.routers.nextcloud.entrypoints=https"
- "traefik.http.routers.nextcloud.rule=HostHeader(`nextcloud.${DOMAINNAME}`)"
- "traefik.http.routers.nextcloud.tls=true"
## Middlewares
- "traefik.http.routers.nextcloud.middlewares=nextcloud-headers,nextcloud-redirectregex1,nextcloud-redirectregex2,chain-no-auth@file" # No Auth
- "traefik.http.middlewares.nextcloud-redirectregex1.redirectregex.permanent=true"
- "traefik.http.middlewares.nextcloud-redirectregex1.redirectregex.regex=https?:/([^\s]*)/.well-known/(card|cal)dav"
- "traefik.http.middlewares.nextcloud-redirectregex1.redirectregex.replacement=https://${1}/remote.php/dav/"
- "traefik.http.middlewares.nextcloud-redirectregex2.redirectregex.permanent=true"
- "traefik.http.middlewares.nextcloud-redirectregex2.redirectregex.regex=https?:/([^\s]*)/(.well-known[^\s]*)"
- "traefik.http.middlewares.nextcloud-redirectregex2.redirectregex.replacement=https://${1}/index.php${2}"
- "traefik.http.middlewares.nextcloud-headers.headers.customFrameOptionsValue=SAMEORIGIN"
- "traefik.http.middlewares.nextcloud-headers.headers.framedeny=true"
- "traefik.http.middlewares.nextcloud-headers.headers.sslredirect=true"
- "traefik.http.middlewares.nextcloud-headers.headers.STSIncludeSubdomains=true"
- "traefik.http.middlewares.nextcloud-headers.headers.STSPreload=true"
- "traefik.http.middlewares.nextcloud-headers.headers.STSSeconds=315360000"
- "traefik.http.middlewares.nextcloud-headers.headers.forceSTSHeader=true"
- "traefik.http.middlewares.nextcloud-headers.headers.sslProxyHeaders.X-Forwarded-Proto=https"
- "traefik.http.middlewares.nextcloud-headers.headers.customResponseHeaders.X-Robots-Tag=none"
- "traefik.http.middlewares.nextcloud-headers.headers.contentTypeNosniff=true"
- "traefik.http.middlewares.nextcloud-headers.headers.referrerPolicy=same-origin"
- "traefik.http.middlewares.nextcloud-headers.headers.browserXssFilter=true"
## HTTP Services
- "traefik.http.routers.nextcloud.service=nextcloud"
- "traefik.http.services.nextcloud.loadbalancer.server.port=80"
[/CODE]
```

Gibt auch hier wieder nichts anzupassen.

UI wieder unter <https://nextcloud.subdomain.duckdns.org> mit Konfiguration beim ersten Start. Hier MariaDB Username, Passwort, BD-Name und zuletzt **localhost** durch **db** ersetzen. Dann die Installation starten.

Ich empfehle über Apps/Security im Drop-Down-Menü oben rechts die TOTP-Extension zu installieren und in den Settings zu aktivieren.

Die neue Instanz erhält die SSL-Zertifikate ganz normal über Traefik, der Cert-Dumper ist damit überflüssig.

TIPP: Dateien können via SFTP in den Nextcloud-Datenordner kopiert werden. Um diese Dateien in den Index aufzunehmen muß Nextcloud den Ordner scannen mit:

```
[CODE=Bash]
sudo docker exec -ti --user www-data nextcloud /var/www/html/occ files:scan --all
sudo docker exec -ti --user www-data nextcloud /var/www/html/occ trashbin:cleanup --all-
users
[/CODE]
```

Der 2te Befehl bereinigt den Papierkorb.

Danach tauchen die Dateien in der Nextcloudoberfläche und via WebDAV auf.

ANMERKUNGEN:

1. Zur Beseitigung des imagemagick SVG-Support Fehlers:

```
[CODE=Bash]
sudo docker exec -t 859a39b4a8d4 /bin/bash -c "apt update && apt install -y libmagickcore-
6.q16-6-extra"
[/QUOTE]
```

2. Nextcloud Docker unterstützt von sich aus keine Cronjobs. Das Netz ist wie immer voller Tipps, die jedoch zu 99% in diesem Szenario nicht funktionieren.

Abhilfe schaffte ein Tipp, einen Eintrag direkt in der Crontab-Datei des Hostsystems anzulegen:

```
[CODE=Bash]
sudo nano /etc/crontab
[/CODE]
```

Mit (am Ende anfügen):

```
[CODE]
*/5 * * * * root docker exec -u www-data -t nextcloud php -f /var/www/html/cron.php
[/CODE]
```

DuckDNS-API IP Updater – Unabhängig von Tools oder Router-Einstellungen:

Praktisches Gimmick, welches alle 5 Minuten ein Status-Update bei der DuckDNS-API macht.

Subdomain (ohne duckdns.org) und sicherstellen, dass das Token in der .ENV angegeben ist.

```
[CODE=Bash]
# DuckDNS Host IP Update Container
duckdns:
  image: lscr.io/linuxserver/duckdns:latest
  container_name: duckdns
  environment:
    - PUID=$PUID
    - PGID=$PGID
    - TZ=$TZ
    - SUBDOMAINS=subdomain
    - TOKEN=$DUCKDNS_TOKEN
    - LOG_FILE=true #optional
```

```
volumes:
  - ./appdata/config:/config #optional
restart: unless-stopped
[/CODE]
```

Das Log findet sich dann unter `${USERDIR}/docker/appdata/config`.

Glances - Echtzeit-Monitoring

Informative Übersicht des Linux-Systems in Taskmanager-Optik.

Keine Anpassungen nötig:

```
[CODE=PowerShell]
## Glances - Container and System Monitor
glances:
  container_name: glances
  image: nicolargo/glances:latest
  pid: host
  restart: always
  networks:
    - t2_proxy
#   ports:
#     - "61208:61208"
  security_opt:
    - no-new-privileges:true
  volumes:
    - $USERDIR/docker/glances:/glances/conf # Use this if you want to add a glances.conf
file
  - /var/run/docker.sock:/var/run/docker.sock:ro
environment:
  - GLANCES_OPT=-w
labels:
  - "traefik.enable=true"
  ## HTTP Routers
  - "traefik.http.routers.glances-rtr.entrypoints=https"
  - "traefik.http.routers.glances-rtr.rule=Host(`glances.$DOMAINNAME`)"
  - "traefik.http.routers.glances-rtr.tls=true"
  ## Middlewares
  - "traefik.http.routers.glances-rtr.middlewares=chain-oauth@file"
  ## HTTP Services
  - "traefik.http.routers.glances-rtr.service=glances-svc"
  - "traefik.http.services.glances-svc.loadbalancer.server.port=61208"
[/CODE]
```

UI unter <https://glances.subdomain.duckdns.org>.

Dozzle – Echtzeit-Logviewer:

Eine weitere nützliche Erweiterung, die ebenfalls keine Anpassungen braucht:

[CODE=PowerShell]

Dozzle - Real-time Docker Log Viewer

dozzle:

image: amir20/dozzle:latest

container_name: dozzle

restart: unless-stopped

networks:

- t2_proxy

security_opt:

- no-new-privileges:true

ports:

- "\$DOZZLE_PORT:8080"

environment:

DOZZLE_LEVEL: info

DOZZLE_TAILSIZE: 300

DOZZLE_FILTER: "status=running"

DOZZLE_FILTER: "label=log_me" # limits logs displayed to containers with this label

DOCKER_HOST: tcp://socket-proxy:2375

volumes:

- /var/run/docker.sock:/var/run/docker.sock # Use Docker Socket Proxy instead for

improved security

labels:

- "traefik.enable=true"

HTTP Routers

- "traefik.http.routers.dozzle-rtr.entrypoints=https"

- "traefik.http.routers.dozzle-rtr.rule=Host(`dozzle.\$DOMAINNAME`)"

- "traefik.http.routers.dozzle-rtr.tls=true"

Middlewares

- "traefik.http.routers.dozzle-rtr.middlewares=chain-oauth@file"

HTTP Services

- "traefik.http.routers.dozzle-rtr.service=dozzle-svc"

- "traefik.http.services.dozzle-svc.loadbalancer.server.port=8080"

[/CODE]

UI unter <https://dozzle.subdomain.duckdns.org>

FileBrowser – Explorer

Keine Ahnung wieso, aber da war er halt.

[CODE=PowerShell]

File Browser - Explorer

filebrowser:

container_name: filebrowser

image: filebrowser/filebrowser:s6

restart: unless-stopped

networks:

- t2_proxy

security_opt:

```

- no-new-privileges:true
#ports:
# - "$FILEBROWSER_PORT:80"
volumes:
- $DOCKERDIR/appdata/filebrowser:/config
- /media:/data/media
- $USERDIR:/data/home
environment:
- PUID=$PUID
- PGID=$PGID
- TZ=$TZ
labels:
- "traefik.enable=true"
## HTTP Routers
- "traefik.http.routers.filebrowser-rtr.entrypoints=https"
- "traefik.http.routers.filebrowser-rtr.rule=Host(`fb.$DOMAINNAME`)"
- "traefik.http.routers.filebrowser-rtr.tls=true"
## Middlewares
- "traefik.http.routers.filebrowser-rtr.middlewares=chain-oauth@file"
## HTTP Services
- "traefik.http.routers.filebrowser-rtr.service=filebrowser-svc"
- "traefik.http.services.filebrowser-svc.loadbalancer.server.port=80"
[/CODE]

```

UI unter <https://fb.subdomain.duckdns.org>. Login ist 2X *admin*.

Fail2Ban – Access Logs Überwachung:

Sehr nützlich als zusätzliche Sicherheitsebene. Gerade wenn SSH über das Netz freigegeben werden soll ein Muß!

Die Konfiguration ist etwas knifflig, daher hänge ich die notwendigen Dateien als ZIP an den Post. Das wäre doch etwas lang.

Die Ordnerstruktur der ZIP spiegelt das fail2ban Verzeichnis nach der Installation. Überwacht werden das SSH Log (auth.log), das Traefik Log (traefik.log) und das Nextcloud Log (nextcloud.log). Letzteres da hier kein OAuth greift, sicher ist sicher.

Für SMTP wird ein Account benötigt, ich benutze Gmail mit OAuth2 und App-Passwort.

```

[CODE=Bash]
## Fail2ban - Network security against attacks
## Some fail2ban commands:
# iptables -L <jail> --line-numbers
# fail2ban-client set <jail> unbanip <ip to unban>
# fail2ban-regex '<log output>' 'regex'
# Other commands: https://www.fail2ban.org/wiki/index.php/Commands
## Check status of ssh jail:
# sudo fail2ban-client status sshd

fail2ban:
container_name: fail2ban
image: crazymax/fail2ban:latest

```

```

restart: always
network_mode: "host"
security_opt:
  - no-new-privileges:true
cap_add:
  - NET_ADMIN
  - NET_RAW
environment:
  - PUID=$PUID
  - PGID=$PGID
  - TZ=$TZ
  - F2B_LOG_LEVEL=INFO
#   - F2B_DB_PURGE_AGE=1d # Age at which bans should be purged from the database
  - F2B_IPTABLES_CHAIN=DOCKER-USER # Specifies the iptables chain to which the
Fail2Ban rules should be added
  - SSMTP_HOST=smtp.gmail.com
  - SSMTP_PORT=587
  - SSMTP_USER=myuser@gmail
  - SSMTP_PASSWORD=password # Create an "app password" if you use 2FA
  - SSMTP_TLS=YES
volumes:
#   - /var/log/docker:/var/log/docker:ro
  - /var/log/auth.log:/var/log/auth.log:ro
  - $USERDIR/docker/traefik2/traefik.log:/var/log/traefik.log:ro
  - $USERDIR/docker/nextcloud/data/data/nextcloud.log:/var/log/nextcloud.log:ro
  - $USERDIR/docker/fail2ban:/data
  - $USERDIR/docker/fail2ban/fail2ban.d:/etc/fail2ban/fail2ban.d
[/CODE]

```

Container installieren und danach über z.B. Portainer stoppen. Die Dateien aus der ZIP in den neuen fail2ban Ordner kopieren und den Container starten.

Das Log kann über Dozzle eingesehen werden.

Anhang Part 3.1 Weitere Apps

Wetty – SSH über das Web:

SSH-Zugang zum lokalen Netz ohne Konsole, direkt über den Browser.

Usernamen und IP anpassen.

```

[CODE=Bash]
## Wetty - SSH over the web
wetty:
  container_name: wetty
  image: svenihoney/wetty:latest
  restart: always
  networks:
    - t2_proxy

```

```
# ports:
#   - "3000:3000"
security_opt:
  - no-new-privileges:true
volumes:
  - /etc/timezone:/etc/timezone:ro
  - /etc/localtime:/etc/localtime:ro
environment:
  - VIRTUAL_HOST=wetty.$DOMAINNAME
  - VIRTUAL_PORT=3000
  - REMOTE_SSH_SERVER=192.168.1.5
  - REMOTE_SSH_PORT=22
  - REMOTE_SSH_USER=s1ave77
labels:
  - "traefik.enable=true"
  ## HTTP Routers
  - "traefik.http.routers.wetty-rtr.entrypoints=https"
  - "traefik.http.routers.wetty-rtr.rule=HostHeader(`wetty.$DOMAINNAME`)"
  - "traefik.http.routers.wetty-rtr.tls=true"
  ## Middlewares
  - "traefik.http.routers.wetty-rtr.middlewares=chain-oauth@file"
  ## HTTP Services
  - "traefik.http.routers.wetty-rtr.service=wetty-svc"
  - "traefik.http.services.wetty-svc.loadbalancer.server.port=3000" ## Wetty - SSH over
the web
[/CODE]
```

UI unter <https://wetty.subdomain.duckdns.org>

Bin – Pastebin

Wieder eine MariaDB Datenbank anlegen und Daten entsprechend anpassen.

[CODE=Bash]

Paste Bin

Change the theme:

sudo nano \$USERDIR/docker/bin/cfg/conf.php

change template = "bootstrap" to "bootstrap-dark-page"

Other options: <https://privatebin.info/screenshots.html>

bin:

container_name: bin

image: jgeusebroek/privatebin:latest

restart: always

networks:

- t2_proxy

- mariadb

ports:

- "80:80"

security_opt:

- no-new-privileges:true

```

depends_on:
  - mariadb
links:
  - mariadb:db
volumes:
  - $USERDIR/docker/bin/data:/privatebin/data
  - $USERDIR/docker/bin/cfg:/privatebin/cfg
environment:
  UID: $PUID
  GID: $PGID
  MYSQL_HOSTNAME: mariadb
  MYSQL_PORT: 3306
  MYSQL_DATABASE: bin_db
  MYSQL_USER: username
  MYSQL_PASSWORD: 'password'
labels:
  - "traefik.enable=true"
  ## HTTP Routers
  - "traefik.http.routers.bin-rtr.entrypoints=https"
  - "traefik.http.routers.bin-rtr.rule=HostHeader(`bin.$DOMAINNAME`)"
  - "traefik.http.routers.bin-rtr.tls=true"
  ## Middlewares
  - "traefik.http.routers.bin-rtr.middlewares=chain-oauth@file"
  ## HTTP Services
  - "traefik.http.routers.bin-rtr.service=bin-svc"
  - "traefik.http.services.bin-svc.loadbalancer.server.port=80"
[/CODE]

```

UI unter <https://bin.subdomain.duckdns.org>.

QDirStat – Verzeichnis-Statistiken:

Nichts anzupassen.

```

[CODE=Bash]
# qDirStat - Directory Statistics
qdirstat:
  image: jlesage/qdirstat:latest
  container_name: qdirstat
  restart: unless-stopped
  networks:
    - t2_proxy
  security_opt:
    - no-new-privileges:true
  volumes:
    - /volume1:/storage:ro
    - $DOCKERDIR/appdata/qdirstat/config:/config:rw
  environment:
    USER_ID: $PUID
    GROUP_ID: $PGID
    UMASK: 002
    TZ: $TZ

```



```

KEEP_APP_RUNNING: 1
CLEAN_TMP_DIR: 1
DISPLAY_WIDTH: 1600
DISPLAY_HEIGHT: 960
labels:
- "traefik.enable=true"
## HTTP Routers
- "traefik.http.routers.qdirstat-rtr.entrypoints=https"
- "traefik.http.routers.qdirstat-rtr.rule=HostHeader(`qdirh.$DOMAINNAME`)"
- "traefik.http.routers.qdirstat-rtr.tls=true"
## Middlewares
- "traefik.http.routers.qdirstat-rtr.middlewares=chain-oauth@file"
## HTTP Services
- "traefik.http.routers.qdirstat-rtr.service=qdirstat-svc"
- "traefik.http.services.qdirstat-svc.loadbalancer.server.port=5800"
[/CODE]

```

UI unter <https://qdirh.subdomain.duckdns.org>.

DupeGuru – Duplikatfinder:

Nichts anzupassen.

[CODE=Bash]

```
# DupeGuru - Duplicate File/Folder Remover
```

```
dupeguru:
```

```
image: jlesage/dupeguru:latest
```

```
container_name: dupeguru
```

```
restart: unless-stopped
```

```
networks:
```

```
- t2_proxy
```

```
security_opt:
```

```
- no-new-privileges:true
```

```
volumes:
```

```
- $USERDIR:/data/home:ro
```

```
- $DOCKERDIR/appdata/dupeguru/config:/config:rw
```

```
- /media/data:/data/data:rw
```

```
environment:
```

```
USER_ID: $PUID
```

```
GROUP_ID: $PGID
```

```
UMASK: 002
```

```
TZ: $TZ
```

```
KEEP_APP_RUNNING: 1
```

```
CLEAN_TMP_DIR: 1
```

```
DISPLAY_WIDTH: 1600
```

```
DISPLAY_HEIGHT: 960
```

```
labels:
```

```
- "traefik.enable=true"
```

```
## HTTP Routers
```

```
- "traefik.http.routers.dupeguru-rtr.entrypoints=https"
```

```
- "traefik.http.routers.dupeguru-rtr.rule=HostHeader(`dupe.$DOMAINNAME`)"
```

```
- "traefik.http.routers.dupeguru-rtr.tls=true"
```

```
## Middlewares
- "traefik.http.routers.dupeguru-rtr.middlewares=chain-oauth@file"
## HTTP Services
- "traefik.http.routers.dupeguru-rtr.service=dupeguru-svc"
- "traefik.http.services.dupeguru-svc.loadbalancer.server.port=5800"
[/CODE]
```

UI unter <https://dupe.subdomain.duckdns.org>.

Cloud Commander – Web Datei Manager:

Nichts anzupassen.

```
[CODE=Bash]
# Cloud Commander - web file manager
cloudcmd:
  image: coderaiser/cloudcmd
  container_name: cloudcmd
  restart: unless-stopped
  networks:
    - t2_proxy
  security_opt:
    - no-new-privileges:true
  volumes:
    - ./cloudcmd:/root
    - $USERDIR:/mnt/fs
  environment:
    PUID: $PUID
    PGID: $PGID
    TZ: $TZ
  labels:
    - "traefik.enable=true"
## HTTP Routers
- "traefik.http.routers.cloudcmd-rtr.entrypoints=https"
- "traefik.http.routers.cloudcmd-rtr.rule=HostHeader(`cloudcmd.$DOMAINNAME`)"
- "traefik.http.routers.cloudcmd-rtr.tls=true"
## Middlewares
- "traefik.http.routers.cloudcmd-rtr.middlewares=chain-oauth@file"
## HTTP Services
- "traefik.http.routers.cloudcmd-rtr.service=cloudcmd-svc"
- "traefik.http.services.cloudcmd-svc.loadbalancer.server.port=8000"
[/CODE]
```

UI unter <https://cloudcmd.subdomain.duckdns.org>.

Anhang Part 4

HTTPS zu HTTP tunneln:

Eine einfache Möglichkeit einen lokalen Webserver von außen erreichbar zu machen über einen sicheren Tunnel.

Eine gute Übersicht findet sich hier: <https://pythonrepo.com/repo/anderspitman-awesome-tunneling-python-networking-programming>.

Für jeden Geschmack etwas dabei, von frei über bezahlt zu selbstgehostet ist einiges möglich.

Wie erwähnt fiel meine Wahl auf localtonet.com, da der freie Account alles bietet, was ich benötige. Außerdem ist es das einzige Angebot, welches ich fand, daß es erlaubt eine passive Verbindung zu etablieren. Das umgeht die üblichen zeitlichen Limitierungen.

Die Verbindung von Client und Server erfolgt bei allen über Tokens, die im Dashboard zu finden sind. Bei LokalToNet lassen sich verschiedene Token für mehrere Tunnel generieren.

Windows: Beim ersten Start das Token eingeben und der Tunnel wird passiv aufgebaut. Die Informationen werden unter %localappdata%\localtonet gespeichert. Löschen des Ordners triggert eine neue Anmeldung.

Mit wartendem Client kann nun bei LocalToNet im Dashboard die Verbindung aktiviert werden. Ich nutze TCP mit entsprechendem Token, deutschem Server an Port 80. Tunnel erstellen und dann starten. Ergebnis wird angezeigt.

Nun kann über den gezeigten Hyperlink auf den lokalen Server zugegriffen werden.

Hinweis: bei Wechsel der IP (Provider- oder Router-Reset) muß der Tunnel neugestartet werden.

The screenshot shows the LocalToNet dashboard interface. On the left is a sidebar menu with options: Dashboard, My Tunnels (selected), HTTP, TCP-UDP, My Subscriptions, My Tokens, Browse Plans, and Order History. The main content area is titled 'TCP - UDP' and contains a form for creating a new tunnel. The form fields are: Protocol Type (TCP), AuthToken (Default Token), Server (DE-Germany(1)), Ip (127.0.0.1), and Port (80). A 'Create' button is at the bottom right of the form. Below the form is a 'Refresh' button. Underneath is a table listing the tunnels. The table has columns: AuthToken Name, Server Domain, Server Ip, Server Port, Is Reserved?, Client Ip, Client Port, Protocol Type, Status, and Action. There is one entry in the table with the following values: AuthToken Name: Default, Server Domain: de1.localtonet.com, Server Ip: 194.35.100.34, Server Port: 34, Is Reserved?: No, Client Ip: 127.0.0.1, Client Port: 80, Protocol Type: TCP, Status: Active, and Action: Stop, Delete. At the bottom of the table, it says 'Showing 1 to 1 of 1 entries'.

AuthToken Name	Server Domain	Server Ip	Server Port	Is Reserved?	Client Ip	Client Port	Protocol Type	Status	Action
Default	de1.localtonet.com	194.35.100.34	34	No	127.0.0.1	80	TCP	Active	Stop Delete

Szenario bei mir: solange der Server online ist, kann ich über den Tunnel auf meine Steuer-Webseite zugreifen und den Guacamole-Server starten für Remotezugriff auf meine Rechner. Alles am Smartphone oder Tablet über den Browser zu erledigen ohne zusätzliche Tools.

Anhang Part 5

OpenSSH unter Windows 10:

Läßt sich einfach und schnell über Powershell aktivieren und einrichten. Powershell als Administrator starten.

Aktivierung und Deaktivierung:

```
[CODE=PowerShell]
Add-WindowsCapability -Online -Name OpenSSH.Client*
Add-WindowsCapability -Online -Name OpenSSH.Server*
Remove-WindowsCapability -Online -Name OpenSSH.Server~~~~0.0.1.0
Remove-WindowsCapability -Online -Name OpenSSH.Client~~~~0.0.1.0
[/CODE]
```

Status checken:

```
[CODE=PowerShell]
Get-Service -Name *ssh*
[/CODE]
```

Automatischen Start konfigurieren und starten:

```
[CODE=PowerShell]
Start-Service ssh-agent
Start-Service sshd
Set-Service -Name agent-StartupType 'Automatic'
Set-Service -Name sshd -StartupType 'Automatic'
[/CODE]
```

Nur noch Port 22 in der Firewall öffnen:

```
[CODE=PowerShell]
New-NetFirewallRule -Name sshd -DisplayName 'OpenSSH Server (sshd)' -Enabled True -
Direction Inbound -Protocol TCP -Action Allow -LocalPort 22
[/CODE]
```

Die Verbindung wird hergestellt durch:

```
[CODE=PowerShell]
ssh user@<IP_or_Domain>
[/CODE]
```

Den Key mit 'yes' akzeptieren bei Erstverbindung und das Passwort eingeben.

Der Prompt sollte nun das entsprechende Login@hostname anzeigen.

Anhang Part 6

Apache Webserver unter Linux:

Alte Installationen beseitigen, installieren und Firewall konfigurieren:

```
[CODE=Bash]
sudo apt-get --purge remove apache2
sudo apt-get autoremove
sudo apt-get install apache2
sudo ufw allow 'Apache'
sudo systemctl status apache2
[/CODE]
```

Mit Strg+C abbrechen.

Ordner für die Webseite erstellen, Rechte vergeben und index.html erstellen mit:

Ordernamen nach belieben anpassen.

```
[CODE=Bash]
sudo mkdir /var/www/wolwake
sudo chown -R $USER:$USER /var/www/wolwake
sudo chmod -R 755 /var/www/wolwake
nano /var/www/wolwake/index.html
[/CODE]
```

Inhalt der index.html:

```
[CODE]
<html>
  <head>
    <title>Welcome to Your_domain!</title>
  </head>
  <body>
    <h1>Success! The your_domain virtual host is working!</h1>
  </body>
</html>
[/CODE]
```

Wieder mit Strg+O, gefolgt von ENTER, speichern und mit Strg+X schließen.

Konfig für die Domain erstellen:

Auch hier die Ordernamen jeweils wieder wie vorhin anpassen.

```
[CODE=Bash]
sudo nano /etc/apache2/sites-available/wolwake.conf
[/CODE]
```

Inhalt:

```
[CODE]
<VirtualHost *:80>
    ServerAdmin webmaster@localhost
    ServerName wolwake
    ServerAlias wolwake
    DocumentRoot /var/www/wolwake
    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
[/CODE]
```

Neue Konfig anmelden, alte Default abmelden und testen mit:

```
[CODE=Bash]
sudo a2ensite wolwake.conf
sudo a2dissite 000-default.conf
sudo apache2ctl configtest
[/CODE]
```

Ergebnis sollte etwa so aussehen:

```
[CODE]
Output
Syntax OK
[/CODE]
```

Zum Übernehmen der Änderungen Apache neustarten:

```
[CODE=Bash]
sudo systemctl restart apache2
[/CODE]
```

Geschafft. Der Server sollte jetzt unter localhost oder der LAN-IP erreichbar sein und die Testseite zeigen.

Anhang Part X (Archiv)

Remote Desktop

Die Suche nach einer lokal gehosteten Lösung stolpert man schnell über das Guacamole Projekt: <https://guacamole.apache.org/>.

Definitiv eine eierlegende Wollmilchsau. Blieb also die Entscheidung des Wie.

Direkte Installation auf einem Linux, Installation als Docker oder als Bestandteil von Yunohost.

Also eine (L)Ubuntu LTS 21.10 Installation in der VM gestartet und getestet...

...und damit beim ersten Problem. Die Anleitungen im Netz sind zahllos aber leider nicht immer stringent. Irgendwie fehlen da immer irgendwelche kleinen Dinge. Für die Ersteller mag dies logisch sein, für einen Quereinsteiger nicht immer komplett nachvollziehbar wo es klemmt. Auch die Tipps aus dem Netz halfen hier nicht weiter.

Nach einigen frustrierenden Tests legte ich das Ganze einen Tag beiseite. Ein neuer Tag, eine neue Suche und ein erster Erfolg.

GitHub: <https://github.com/boschkundendienst/guacamole-docker-compose>

Nach erfolgreichem Erstellen der YAML Datei für Docker-Compose startete das Guacamole System erfolgreich. Leider ohne TOTP-Auth, also nur ein Teilerfolg.

Hilfe brachte ein Issue in selbigem Git, der das Einbinden der TOTP Authentifizierung beschreibt.

Guacamole.TOTP: <https://github.com/boschkundendienst/guacamole-docker-compose/issues/18>

Die notwendigen TARs finden sich im Downloadbereich der Guacamole Seite.

Siehe **Anhang Part 2** mit weiterführenden Tipps.

Remote-Desktop mit Apache Guacamole:

Um zu validieren, daß meine Notizen zum Ablauf Sinn ergeben, und da das bisher genutzte Ubuntu mit 40 GB etwas groß war, habe ich ein schlankeres Linux mit GUI gesucht.

Meine Wahl fiel letztendlich auf Linux Mint Xfce. Durch die Ubuntu Nähe konnten alle Kommandos einfach weitergenutzt werden.

Guacamole Docker-Stack Installation:

Hier gilt es sorgsam zu arbeiten, um zu gewährleisten, daß alle Dateien am richtigen Ort liegen.

Als User angemeldet arbeite ich dabei unter /home/mae1cum77/ für die Guacamole-Installation. Im Weiteren also alle derartigen Pfade an den eigenen User anpassen. Ich weise an entsprechender Stelle noch einmal darauf hin.

Als nächstes werden das Git geklont und die Rechte für den neuen Ordner gesetzt. Danach das Skript prepare.sh ausgeführt für die Nginx- und PostGRES-Einrichtung und die Erstkonfiguration.

```
[CODE=Bash]
git clone "https://github.com/boschkundendienst/guacamole-docker-compose.git"
sudo chmod +x /usr/local/bin/docker-compose
cd guacamole-docker-compose
sudo ./prepare.sh
[/CODE]
```

TOTP: Für ein an sich sensibles System und mit dem Gedanken das Ganze bei Bedarf vom Internet aus zugänglich zu machen, empfiehlt es sich 2FA TOTP-Authentifizierung zu integrieren. Wer das nicht will oder braucht kann zum YAML-Part vorspringen.

Ich nutze dafür unter Android andOTP (ermöglicht verschlüsselte Backups, die auf andere Geräte übertragen werden können). [<https://github.com/andOTP/andOTP>]

Die benötigten Extensions als TAR.GZ gibt es hier: <https://guacamole.apache.org/releases/>

Hier die *guacamole-auth-header-1.4.0.tar.gz*, *guacamole-auth-totp-1.4.0.tar.gz* und *guacamole-auth-quickconnect-1.4.0.tar.gz* herunterladen und entpacken. Bei mir kann der Total Commander die Archive entpacken. Gibt reichlich Anleitungen für Linux.

Gebraucht werden die jeweiligen JARs aus den Archiven.

Unterordner erstellen mit (alle mae1cum77 mit dem eigenen User ersetzen):

```
[CODE=Bash]
mkdir /home/mae1cum77/guacamole-docker-compose/guachome
mkdir /home/mae1cum77/guacamole-docker-compose/guachome/extensions
mkdir /home/mae1cum77/guacamole-docker-compose/extensions
[/CODE]
```

Die JARs kommen in den Ordner (geht mit installierten Open-VM-Tools per Drag `n Drop):

```
[CODE]
/home/mae1cum77/guacamole-docker-compose/extensions
[/CODE]
```

Im Bash befinden wir uns nachwievor im /guacamole-docker-compose Ordner.

Braucht nun noch eine Konfig-Datei für die TOTP-Konfiguration, dafür erstellen wir per Nano eine guacamole.properties mit:

```
[CODE=Bash]
sudo nano /home/mae1cum77/guacamole-docker-compose/guachome/
guacamole.properties
[/CODE]
```

Im Bash öffnet sich die leere Datei, einfach per Copy `n Paste folgenden Inhalt einfügen:

```
[CODE]
# guacamole.properties
enable-environment-properties: true
http-auth-header: REMOTEUSER
guacd-hostname: localhost
guacd-port: 4822
[/CODE]
```

Das Ganze mit Strg+O, gefolgt von ENTER, speichern und mit Strg+X beenden.

Fast geschafft, fehlt noch die YAML (docker-compose.yml im /guacamole-docker-compose Ordner) für Docker-Compose. Hat einige Tests gebraucht, aus den Beispielen auf der Seite

und der im Git-Clone enthaltenen Originaldatei, die richtige Integration für TOTP herauszufinden.

Daher hier meine Version. Das Skript nutzt relative Pfade, daher müssen nur die beiden *POSTGRES_PASSWORD*: Einträge unter *#postgres* und *#guacamole* mit eigenen Passwörtern ersetzt werden:

```
[CODE=Bash]
```

```
version: '2.0'
```

```
# networks
```

```
# create a network 'guacnetwork_compose' in mode 'bridged'
```

```
networks:
```

```
  guacnetwork_compose:
```

```
    driver: bridge
```

```
# services
```

```
services:
```

```
  # guacd
```

```
  guacd:
```

```
    container_name: guacd_compose
```

```
    image: guacamole/guacd
```

```
    networks:
```

```
      guacnetwork_compose:
```

```
    restart: always
```

```
    volumes:
```

```
      - ./drive:/drive:rw
```

```
      - ./record:/record:rw
```

```
  # postgres
```

```
  postgres:
```

```
    container_name: postgres_guacamole_compose
```

```
    environment:
```

```
      PGDATA: /var/lib/postgresql/data/guacamole
```

```
      POSTGRES_DB: guacamole_db
```

```
      POSTGRES_PASSWORD: 'password_here'
```

```
      POSTGRES_USER: guacamole_user
```

```
    image: postgres:13.4
```

```
    networks:
```

```
      guacnetwork_compose:
```

```
    restart: always
```

```
    volumes:
```

```
      - ./init:/docker-entrypoint-initdb.d:ro
```

```
      - ./data:/var/lib/postgresql/data:rw
```

```
# guacamole
```

```
guacamole:
```

```
  container_name: guacamole_compose
```

```
  depends_on:
```

```
    - guacd
```

```
    - postgres
```

```
  environment:
```

```
    GUACD_HOSTNAME: guacd
```

```
    POSTGRES_DATABASE: guacamole_db
```

```
    POSTGRES_HOSTNAME: postgres
```

```

    POSTGRES_PASSWORD: 'password_here'
    POSTGRES_USER: guacamole_user
    # set custom GUACAMOLE_HOME within container
    GUACAMOLE_HOME: /guachome
volumes:
- ./guachome:/guachome
- ./extensions:/guachome/extensions
image: guacamole/guacamole
links:
- guacd
networks:
  guacnetwork_compose:
ports:
- 8080/tcp
restart: always

# nginx
nginx:
  container_name: nginx_guacamole_compose
  restart: always
  image: nginx
  volumes:
  - ./nginx/ssl/self.cert:/etc/nginx/ssl/self.cert:ro
  - ./nginx/ssl/self-ssl.key:/etc/nginx/ssl/self-ssl.key:ro
  - ./nginx/nginx.conf:/etc/nginx/nginx.conf:ro
  - ./nginx/mysite.template:/etc/nginx/conf.d/default.conf:ro
  ports:
  - 8443:443
  links:
  - guacamole
  networks:
    guacnetwork_compose:
  # run nginx
  command: /bin/bash -c "nginx -g 'daemon off;'"
[/CODE]

```

Ich habe die Originaldatei im Ordner mit obiger Version ersetzt. Es kann aber auch Nano in der CLI genutzt wrden, je nach Geschmack.

Am Besten einen letzten Snapshot machen. Man weiß ja nie.

Eine letzte Kontrolle, ob alle Dateien am richtigen Ort und die Konfig und YAML korrekt angekommen sind kann nicht schaden.

Im Bash sollten wir uns immer noch im /guacamole-docker-compose Ordner, ansonsten per cd Befehl dorthin wechseln.

Zeit den Compose-Prozeß zu starten. Das geht mit:

```

[CODE=Bash]
sudo docker-compose up -d
[/CODE]

```

Am Ende sollte der Prozess die Erstellung von 4 Volumes mit ‚Done‘ quittieren.

Wir checken den Zustand mit:

```
[CODE=Bash]
sudo docker ps -a
[/CODE]
```

Das sollten dann folgendermaßen aussehen:

```
[CODE]
CONTAINER ID   IMAGE                  COMMAND                  CREATED        STATUS
PORTS         NAMES
c7d0573ec024   nginx                 "/docker-entrypoint...." 20 hours ago   Up About a minute
80/tcp, 0.0.0.0:8443->443/tcp, :::8443->443/tcp   nginx_guacamole_compose
0885bedc302f   guacamole/guacamole   "/opt/guacamole/bin/..." 20 hours ago   Up About
a minute      0.0.0.0:49153->8080/tcp, :::49153->8080/tcp
guacamole_compose
6ec59c9db1c4   postgres:13.4         "docker-entrypoint.s..." 20 hours ago   Up About a
minute       5432/tcp      postgres_guacamole_compose
361e7a0bfe78   guacamole/guacd       "/bin/sh -c '/usr/lo..." 20 hours ago   Up About a
minute (health: starting) 4822/tcp      guacd_compose
[/CODE]
```

Sollte der guacamole_compose Container nicht ‚Up‘ sein, ist etwas mit den Extensions nicht sauber.

Einrichtung:

Gehobene Schwierigkeit aber keine Magie wie man sieht. Und ein gutes Gefühl auf jeden Fall.

Im Browser nun unter [\[iCODE\]https://<IP or Domain>:8443\[/iCODE\]](https://<IP or Domain>:8443/) die Weboberfläche aufrufen. Da ein selbstgezeichnetes Zertifikat genutzt wird taucht eine Warnung auf, die über ‚Erweitert‘ akzeptiert werden muß. Kein Beinbruch.

Das Login ist beides *guacadmin* und sollte dann in den Settings schnellstens geändert werden.

Als nächstes sollte ein QR-Code auftauchen und die übliche TOTP-Verknüpfung mit dem entsprechenden Mobilgerät.

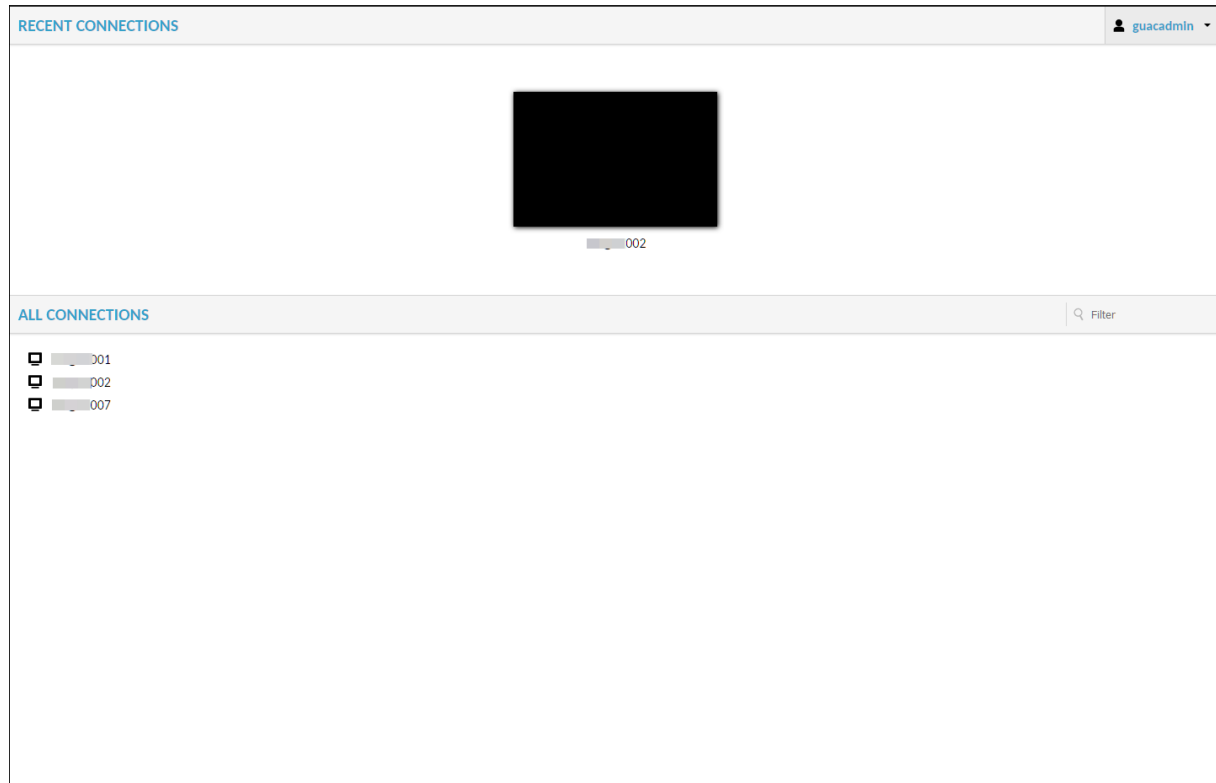
Geschafft! Nun noch die RDP-Verbindung konfigurieren und es kann losgehen.

Rechts oben im Drop-Down-Menü die Settings auswählen und unter *Preferences* als erstes ein neues Passwort vergeben. Danach unter *Connections* die Verbindungen anlegen.

Das geht via *New Connection*. Unter *EDIT CONNECTION* einen Namen vergeben und das Protokoll auf *RDP* setzen.

Nun unter *PARAMETERS* weiter. *Hostname* und *Port* vergeben, Standard-Port ist *3389*. *Username* und *Password* des gewünschten Accounts eintragen und bei *Security mode* die *NLA (Network Level Authentication)* wählen. Ich habe noch die Option für das Ignorieren des Server-Zertifikats angehakt.

Das wars schon. Ein Test sollte Aufschluß geben.



Nextcloud

Wie schon bei Guacamole gibt es mehrere Möglichkeiten. Und damit die selben Probleme. Meine Präferenz war eindeutig ein eigenständiges System in seiner eigenen VM. Wie gesagt soll der Guacamole Server nur bei Bedarf gestartet werden, außerdem wollte ich mögliche Konflikte einer parallelen Dockerinstallation unter dem Ubuntu von vornherein umgehen.

Die Konfiguration der eigenen Domain inklusive DynDNS ist alles andere als trivial. Hier kann man einiges falsch machen, gerade wenn geplant ist, die Cloud von außen erreichbar zu machen.

Einen Artikel von Golem.de mit dem Hinweis auf Yunohost im Hinterkopf, holte ich mir die ISO und bereitete eine neue VM vor.

Yunohost: <https://yunohost.org/#/>

Die Vorteile für mich sind auf jeden die geführte Installation mit Ersteinrichtung und eine hilfreiche Diagnosefunktion. Letztere hilft bei fehlkonfigurierten DNS-Einträgen und anderen Einstellungen nach kurzer Einarbeitung auch Anfängern schnell die Fehler zu beheben. Auch die Verwaltung von Let's Encrypt Zertifikaten ist mit wenigen Klicks geregelt.

Das Ganze basiert auf Debian Buster und benötigt recht wenig Ressourcen. 99% der Administration ist über die Weboberfläche realisierbar.

Zusätzlich integriert das System nützliche Features wie Firewall und Fail2Ban.

Die Nextcloud-Installation ist schnell, unspektakulär und mit wenigen Klicks eingerichtet. TOTP in den Nextcloud-Einstellungen aktiviert und via Total Commander WebDAV-Plugin alle Geräte mit Zugriff ausgestattet.

Siehe **Anhang Part 3** mit weiterführenden Tipps.

Yunohost und Nextcloud:

Um erfolgreich testen zu können, wird eine funktionierende DynDNS-Domain benötigt.

Die Installation ist wie erwähnt selbsterklärend.

Nachdem das getan ist, in die Weboberfläche einloggen. Dabei wird ein Benutzer erstellt, dieser hat nachher Zugriff auf das entsprechende Portal mit den installierten Apps.

Während der Ersteinrichtung findet eine Diagnose statt. Diese ist äußerst nützlich. Sie zeigt den Ist-Zustand und wie es konfiguriert werden muß. Gerade für Portweiterleitungen und einige DNS-Einstellungen sind damit schnell überprüft und notfalls korrigiert.

Ich habe die Ports explizit für die VM geöffnet.

Wenn mehrere Apps installiert werden sollen empfiehlt es sich die entsprechenden (Sub-)Domains gleich hinzuzufügen und eine erneute Diagnose anzustoßen. Dies ist notwendig für die Zertifikate.

Dazu unter *Domains* die entsprechende auswählen und auf *SSL certificate* gehen. Zeigte die Diagnose keine Auffälligkeiten, ist die *Let's Encrypt* Option verfügbar und kann einfach aktiviert und bestätigt werden.

Tuning:

Weiterführend kann das User-Portal noch individualisiert werden. Zum Kopieren der Dateien bietet sich ein SFTP-Client wie das SFTP-Plugin von Total Commander, FileZilla oder WinSCP an, ich nutze auf allen Geräten ersteres.

In der Yunohost-CLI, ich nutze komfortabel SSH, nun im `/home/admin` Ordner einen Ordner anlegen, auf diesen haben wir Zugriffsrechte. Vorher checken wir noch die vorhandenen Themes.

```
[CODE=Bash]
cd /
sudo ls /usr/share/ssowat/portal/assets/themes/
sudo mkdir /home/admin/mae1cum77
[/CODE]
```

Ich werde das Cloud-Theme als Basis nutzen und nur Hintergrund und Logo tauschen.

```
[CODE=Bash]
sudo cp -r /usr/share/ssowat/portal/assets/themes/clouds/* /home/admin/mae1cum77/
[/CODE]
```

Konfigurationsdatei editieren.

```
[CODE=Bash]
sudo nano /etc/ssowat/conf.json.persistent
[/CODE]
[CODE]
{
  "redirected_urls": {
    "nextcloud.xxxxx.selfhost.co/": "nextcloud.xxxxx.selfhost.co/nextcloud",
    "xxxxx.selfhost.co/": "xxxxx.selfhost.co/nextcloud"
  },
  "theme": "mae1cum77"
}
[/CODE]
```

Hier nur den Themenamen anpassen. Dann die background.jpg und cloud.png (das Logo) mit eigenen Dateien ersetzen.

Als letztes noch einen Themes-Ordner anlegen und die Dateien in den neuen Ordner kopieren.

```
[CODE=Bash]
sudo mkdir /usr/share/ssowat/portal/assets/themes/mae1cum77
sudo cp -r /home/admin/mae1cum77/*
/usr/share/ssowat/portal/assets/themes/mae1cum77/
[/CODE]
```

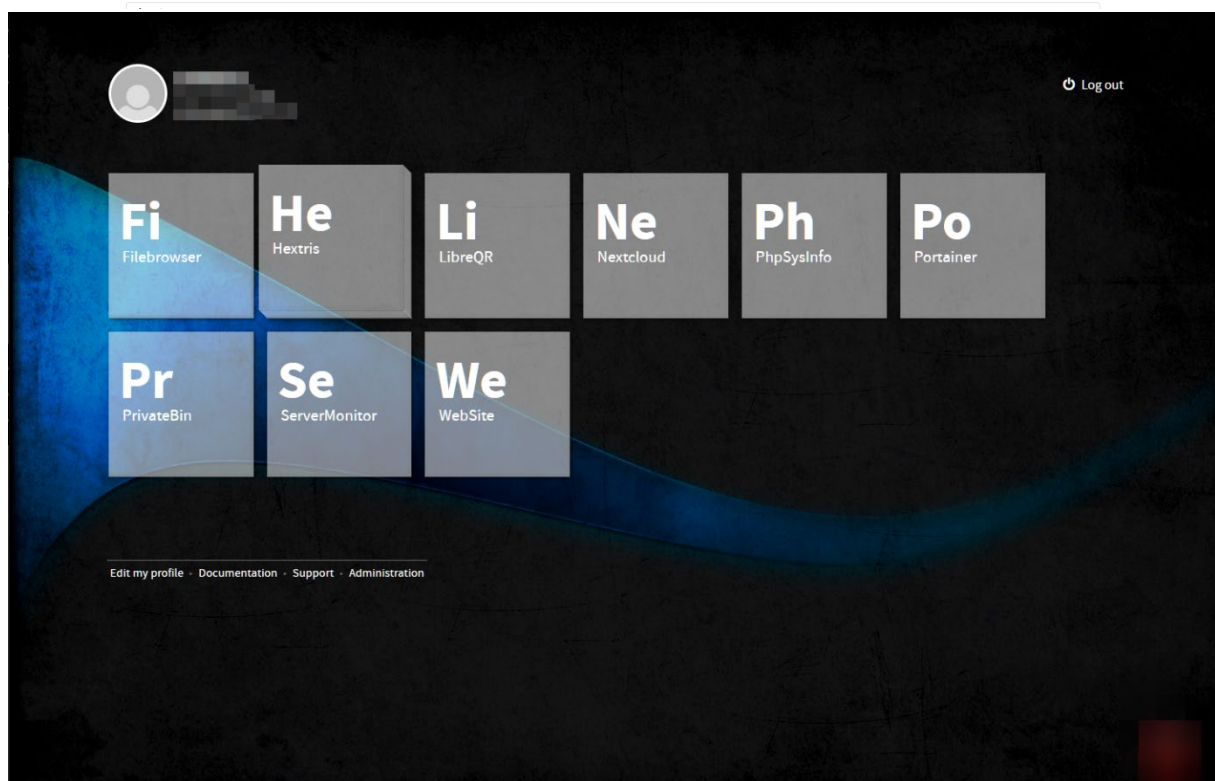
Im Browser mit Strg+F5 die Ansicht erneuern.

[User interface](#)[Logout](#)

Users	>
Domains	>
Applications	>
System update	>
Services	>
Tools	>
Diagnosis	>
Backup	>

[Documentation](#) · [Need help?](#) · [Donate](#)

Powered by YunoHost 4.3.6.3 (stable).



Nextcloud:

Im *Applications* Bereich auf *Install* klicken und unter *Synchronization* die *Nextcloud* finden.

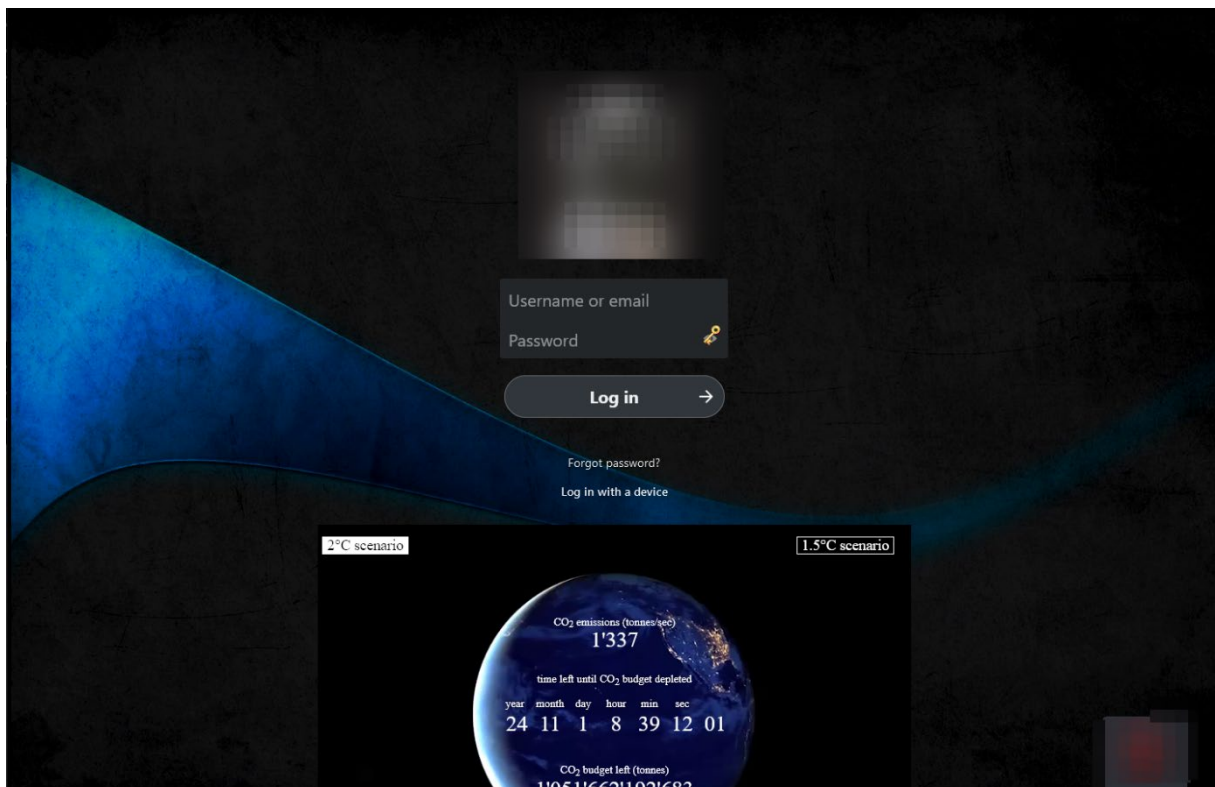
Einen Namen für die Kachel im Benutzerportal festlegen und eine Domain wählen, die Nextcloud nutzen wird. Nutzer festlegen und die Option die Cloud für anonyme Besucher sichtbar zu machen, deaktivieren, sicher ist sicher.

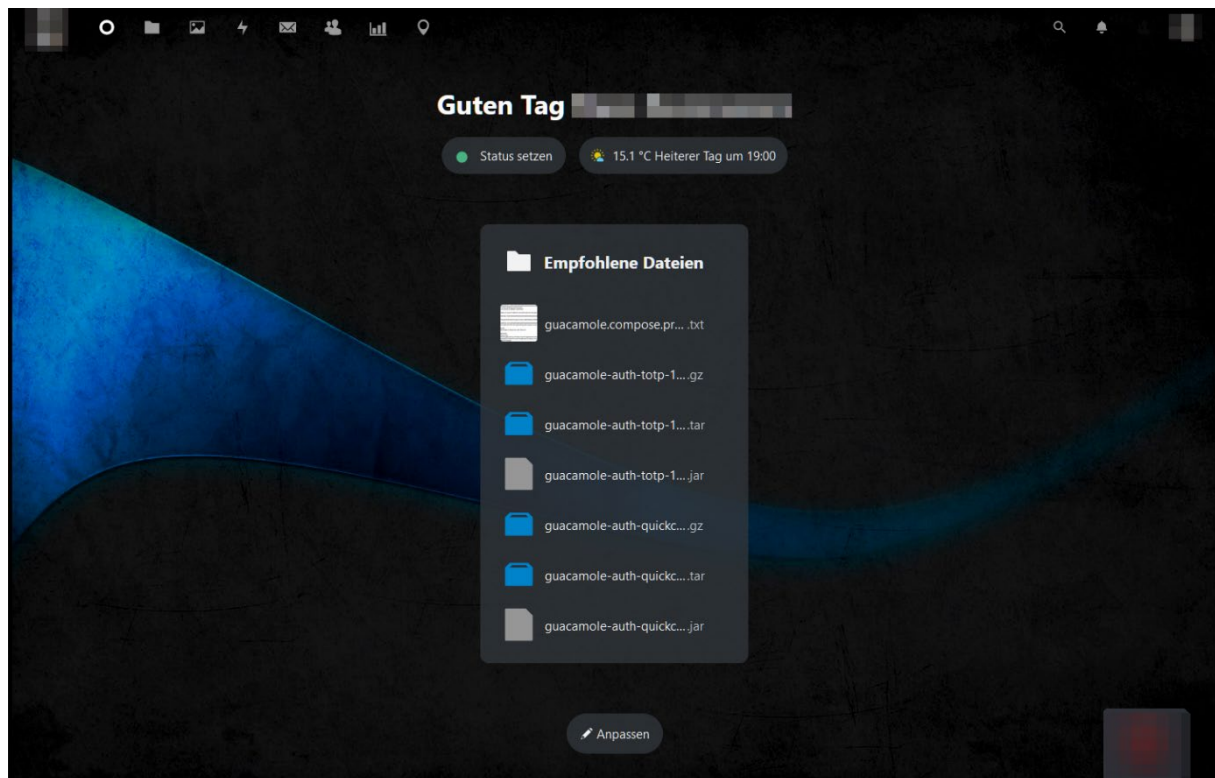
Die Installation starten und den Server machen lassen.

Weiterführende Einstellungen werden in der Weboberfläche durchgeführt.

Hier die Sicherheitseinstellungen überprüfen, 2FA aktivieren. An dieser Stelle lassen sich Apps für direkten WebDAV-Zugriff einrichten. Es existiert eine App aber ich nutze auf allen meinen Systemen Total Commander mit WebDAV-Plugin.

Wenn alles richtig gelaufen ist, wird die Nextcloud via 2FA gesichert und der Yunohost wird von Fail2Ban überwacht. Letzterer Dienst bannt verdächtige IPs für jeweils 10 min, fallen diese mehrfach auf werden sie permanent gebannt. Das läßt sich in der Yunohost-CLI rückgängig machen und feineinstellen, wenn erwünscht.





Portainer:

Yunohost bietet die Möglichkeit auch nicht unterstützte Apps über ein entsprechendes Git zu installieren.

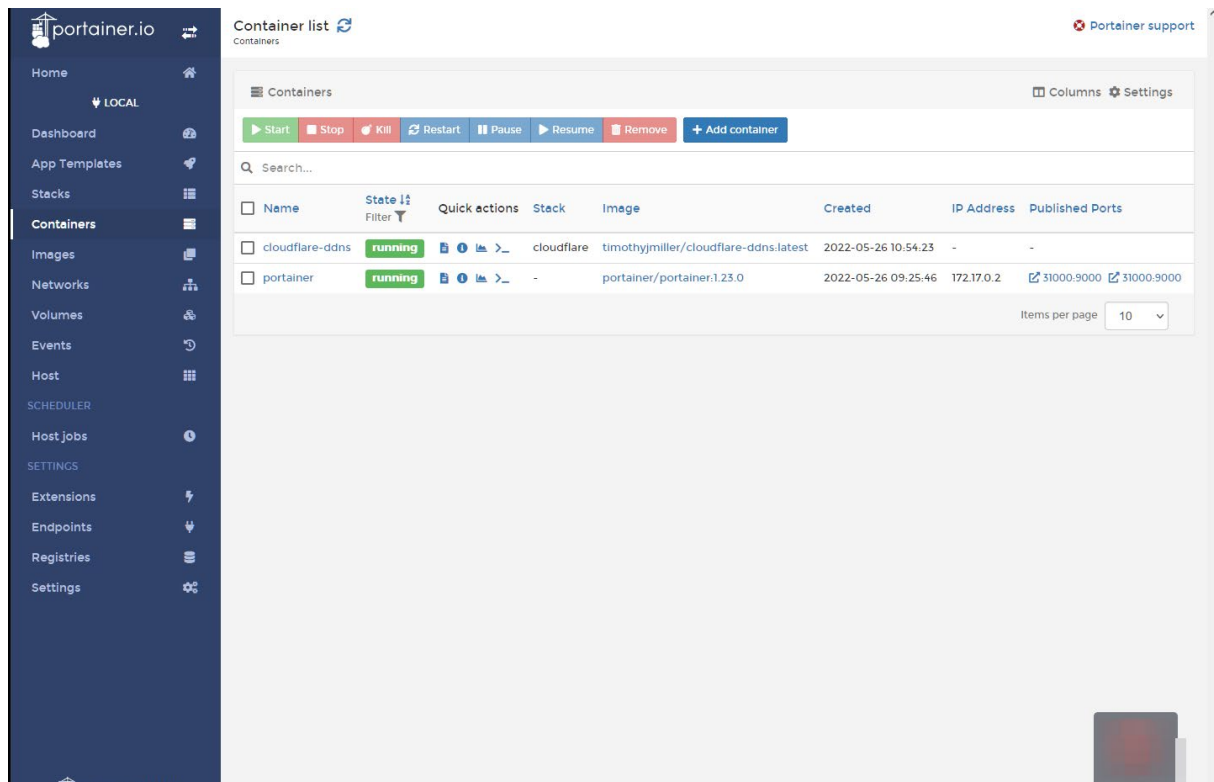
Mir ging es um die Option weitere Funktionen via Docker hinzuzufügen und zu verwalten.

Gibt verschiedene Möglichkeiten, Erfolg hatte ich mit:

https://github.com/DerekJarvis/portainer_ynh.

Bei Applications Install ganz unten den Link einfügen und installieren lassen.

Die Kachel findet sich im Portal.



Cloudflare-Protection-Integration:

War mein Grund für Portainer.

Voraussetzung ist eine nicht-DynDNS Domain bei einem der einschlägigen Anbieter. Ich nutze ein Angebot von 1Blu mit .de Domain, 2 Email Adressen und kleinem Webspace für 11,99€ pro Jahr. Eine überschaubare Investition.

Für solch einen Fall ist der Cloudflare Schutz eine gute Option. Neben DDOS-Schutz wird auch die IP verschleiert. Willkommen in der *Schönen Neuen Welt*.

Ein freier Cloudflare-Account reicht dafür aus.

Die Domain hinzufügen und nach einem kurzen Check der Aufforderung folgen und in den DNS-Einstellungen des Domain-Providers die Name-Server auf die angegebenen von Cloudflare ändern. Die Check-Option aktivieren. Die Benachrichtigung erfolgt via Mail.

Die restlichen Schritte der Einrichtung abschließen. Jetzt heißt es etwas warten, bis Cloudflare alles einrichtet. Wie gesagt kommt eine Mail.

Ist das geschehen in den Einstellungen der Domain bei Cloudflare die restlichen Einträge für Email und co. setzen lassen.

Beim Aufruf der eigenen Seite sollte jetzt der klassische Cloudflare-Screen erscheinen, vor der Weiterleitung.

Cloudflare DDNS-Docker:

Der eingerichtet Cloudflare-Account läßt sich auch für DDNS nutzen. Achtung: dadurch ist die Website nicht erreichbar. Bei mir ist das egal, ich nutze die nicht.

In Cloudflare kann entweder ein DNS-fähiger API-Key erstellt werden, oder man nutzt den Globalen. Desweiteren wird die Zone-ID und das Login-Passwort mit zugehöriger Email benötigt.

Als Root in der Yunohost-CLI anmelden (direkt oder via SSH).

Ich nutze im Folgenden dieses Git: <https://github.com/timothymiller/cloudflare-ddns>

Ins /home/mae1cum77 Verzeichnis wechseln, Git herunterladen, Config erstellen und in Nano öffnen mit:

```
[CODE=Bash]
cd /
cd /home/mae1cum77
sudo git clone https://github.com/timothymiller/cloudflare-ddns
cd /home/mae1cum77/cloudflare-ddns
sudo cp config-example.json config.json
sudo nano config.json
[/CODE]
```

Inhalt der config.json:

Ich nutze den Globalen API-Key mit meiner Email. Wird das API-Token genutzt, dieses darüber zwischen den Anführungszeichen einfügen und die Platzhalter für den Key löschen.

Selbiges gilt für die Domains, diese sind anzupassen, überflüssige Platzhalter zu löschen.

```
[CODE=Bash]
{
  "cloudflare": [
    {
      "authentication": {
        "api_token": "",
        "api_key": {
          "api_key": "Global_API_Key_here",
          "account_email": "mymail@mail.com"
        }
      },
      "zone_id": "d4357742b60bed2192eb457bff705a4c",
      "subdomains": [
        "domain1.ddnss.org",
        "domain2.ddnss.org",
        "domain3.ddnss.org"
      ],
      "proxied": false
    }
  ],
  "a": true,
  "aaaa": true,
}
```

```
"purgeUnknownRecords": false
}
[/CODE]
```

Weiter geht es in Portainer. Einfach aus dem Portal heraus starten. Zu Stacks wechseln und einen neuen erstellen. Default ist der Webeditor. Diesen gilt es mit dem Docker-Compose Skript zu füttern. Einen Namen für das Stack wählen.

Wieder den Usernamen anpassen nicht vergessen.

```
[CODE=Bash]
version: 2
```

```
services:
  cloudflare-ddns:
    image: timothyjmiller/cloudflare-ddns:latest
    container_name: cloudflare-ddns
    security_opt:
      - no-new-privileges:true
    network_mode: "host"
    environment:
      - PUID=1000
      - PGID=1000
    volumes:
      - ./home/mae1cum77/cloudflare-ddns/config.json:/config.json
    restart: unless-stopped
[/CODE]
```

Prozess starten und warten.

Als Letztes in der Yunohost-CLI das Synchronisations-Skript starten mit:

```
[CODE=Bash]
sudo ./start-sync.sh
[/CODE]
```

Wenn alles richtig läuft, tauchen die angegebenen Domains im Cloudflare-Account auf und lassen sich aktivieren.