

Ein Server vor einem NAS

Jochen Paul

7. November 2020

Inhaltsverzeichnis

1 Motivation	3
2 Aktueller Zustand	4
3 Die Idee	5
4 Zielvorstellung	5
5 Auswahl des Host-OS	6
6 Erstinstallation	7
6.1 Das Host-OS	7
6.2 Vorbereitungen	7
7 Die Verbindung zum NAS	8
7.1 Via NFS Share	9
7.1.1 NFS auf dem NAS	9
7.1.2 NFS auf der ZBox	9
7.1.3 NFS Share mounten	10
7.2 Via SMB Share	11
7.2.1 SMB auf dem NAS	11
7.2.2 SMB auf der ZBox	11
7.2.3 SMB Share mounten	11
7.3 Via iSCSI	12
7.3.1 iSCSI-Target auf dem NAS anlegen	12
7.3.2 iSCSI auf der ZBox	16
7.3.3 Nach dem Reboot	17
7.4 Remote Volumes	18
7.4.1 NFS Volumes	18
7.4.2 SMB Volumes	18
8 Docker Umgebung	18
8.1 Installation von Docker	19
8.2 Installation von Docker Compose	19
8.3 Abhängigkeiten	19
9 Die Container	20
9.1 Containerliste	20
9.2 Portainer	21
9.3 Versionierungssystem	24

9.3.1 Installation des Docker Containers	24
9.3.2 Initiale Konfiguration	25
9.3.3 SCM-Manager Plugins	25
9.4 MariaDB	25
9.5 phpMyAdmin	26
9.6 Mosquitto	27
9.7 Syncthing	28
9.8 NextCloud	30
9.9 Dateiserver	31
9.9.1 Samba Image	31
9.10 ClamAV	32
9.11 restic	32
9.12 MinIO	32
9.13 Pi-Hole	32
10 All in One Docker File	32
11 FAQ	34
Abbildungsverzeichnis	36
Tabellenverzeichnis	37

1 Motivation

Das [NAS DS411Slim](#) von Synology bietet zahlreiche Features. Unter anderem kann die Funktionalität als reines NAS erweitert werden, zum Beispiel mit

- Einer [Maria DB](#)
- Einem [MQTT Broker](#)
- Einem [Subversion Server](#)
- Einem [VirensScanner](#)

Diese Erweiterungen werden von mir eingesetzt, und zwar zusätzlich zu den ab Werk installierten Features wie

- Einem [DLNA Server](#)
- Einem [Dateiserver](#)

Mittlerweile ist das NAS etwas in die Jahre gekommen – und wird von Synology seit der [Version 6.2.1-23824](#) nicht mehr unterstützt. Es gibt, falls überhaupt, nur noch Sicherheitsupdates. Neue Features oder eine neue Variante des [Diskstation Manager \(DSM\)](#), dem Betriebssystem von Synology für ihre NAS-Produkte, sind ausgeschlossen.

Zudem ist die Hardware nicht leistungsfähig genug, um aktuelle Bedürfnisse zu befriedigen. Der [Prozessor](#) ist laut [Dokumentation](#) ein [Marvel Kirkwood](#), welcher eine [ARMv5](#) Architektur hat. Dazu kommt, dass das NAS lediglich 256 MB RAM hat.

Deswegen möchte ich meine [ZBox ID41+](#) als Server vor dem NAS verwenden. Die ZBox hat einen [Intel Atom D525](#) Prozessor. Diesem stehen zudem 4 GB RAM zur Seite – das sollte mehr Performance bieten, als es dem NAS jemals möglich wäre.

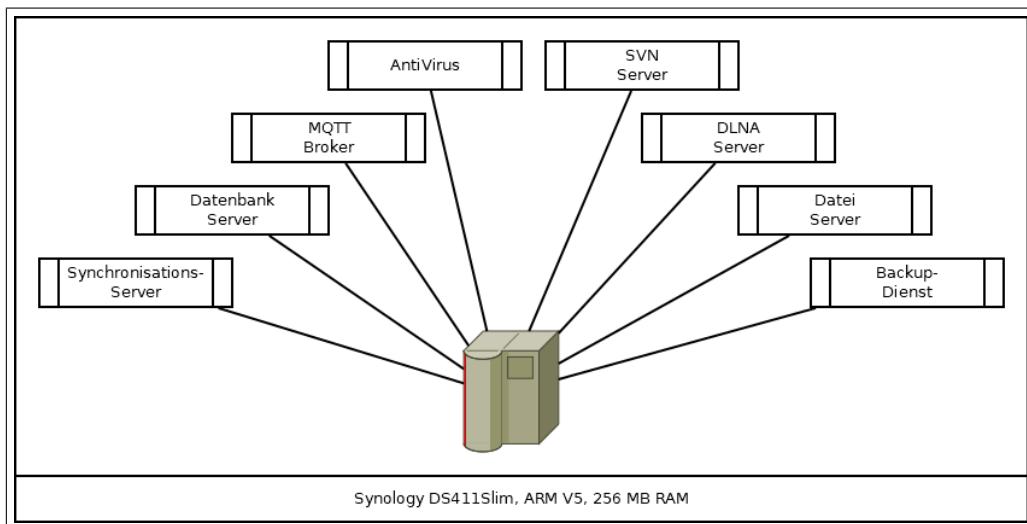
Zum Vergleich:

	Synology DS411Slim	ZBox ID41+
CPU	Marvel Kirkwood	Intel Atom D525
Kerne	1	2
Architektur	ARMv5	x86
RAM	256MB	4GB

Tabelle 1 – Vergleich CPU/RAM

2 Aktueller Zustand

Aktuell ist ausschließlich das NAS in Verwendung. Da dabei ziemlich viele Dienste darauf laufen, kommt die Hardware ziemlich schnell an ihre Grenzen. Grob skizziert sieht das in etwa so aus:

**Abbildung 1 – Aktueller Zustand**

Konkret sind das folgende Pakete auf dem NAS:

- Synchronisations-Server – Cloud Station Server von Synology
- Datenbank Server – MariaDB von Synology
- MQTT Broker – Mosquitto von der Synology Community
- AntiVirus – Antivirus Essential von Synology
- SVN Server – SVN von Synology

- DLNA Server – Von Synology
- Datei Server – Samba, spezielle Version von Synology
- Backup-Dienst – Hyper Backup von Synology

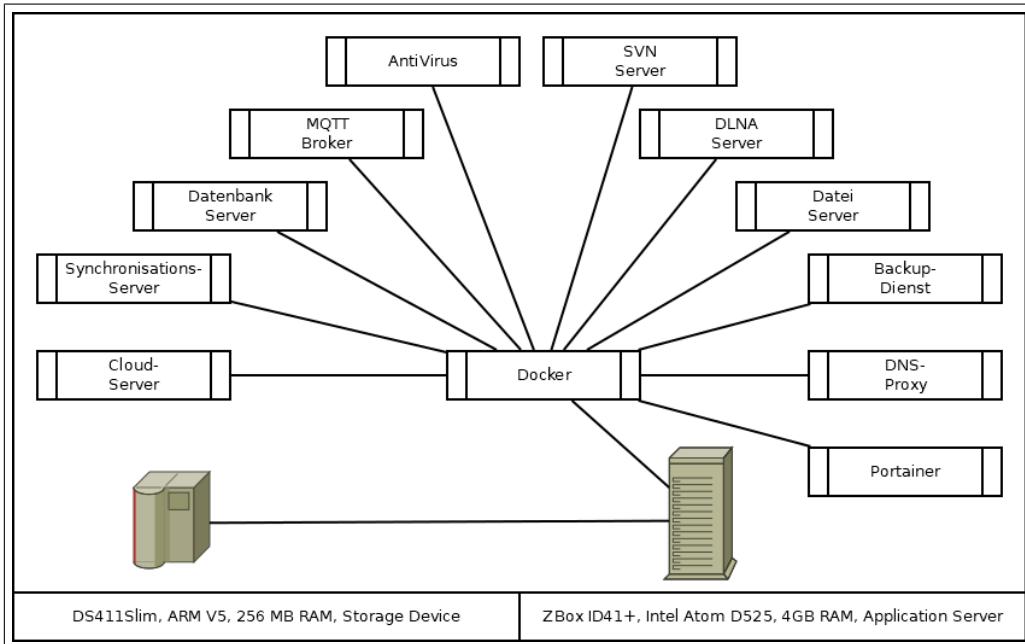
3 Die Idee

Die Idee ist, auf der ZBox ein Host-OS einzurichten, und zwar für [Docker](#). Die oben erwähnten Applikationen sollen dann in Docker Containern laufen. Als Datenspeicher dient nach wie vor das NAS, jedoch wird der Festplattenplatz des NAS irgendwie in das Host-OS eingebunden und die Docker Container legen darauf ihre Daten ab. Irgendwie deswegen, da es ja verschiedene Möglichkeiten gibt:

- Als Filesystem
 - [SMB Share](#)
 - [NFS Share](#)
- Als Blockdevice

4 Zielvorstellung

Aus der Idee ergibt sich dann, grob skizziert, folgende Zielvorstellung:

**Abbildung 2 – Zielvorstellung**

5 Auswahl des Host-OS

Das Projektes beginnt am 31. Dezember 2018. Grundsätzlich könnte jede Linux Variante als Host Betriebssystem gewählt werden. Aber Linux wäre ja nicht Linux, wenn es nur eine Distribution für das Hosten von Docker Containern gäbe. Zur Auswahl stehen unter anderem:

- [Alpine Linux](#)
- [OpenMediaVault](#)
- [Photon](#)
- [SmartOS](#)

Von jedem OS wurde ein Testsystem in einer [Virtual Box](#) installiert und kurz angetestet. Wichtig ist für mich, daß keine speziellen Befehle zur Verwendung kommen, sondern das sich das System sowie das Handling an die von mir gewohnten [Debian](#) basierten Systemen anlehnt.

Nach verschiedenen Tests diesbezüglich habe ich mich erst einmal

für Alpine Linux entschieden. Nicht nur für Container selbst wird das gerne verwendet; durch seine geringe Anforderungen kann es auch gut als Host-OS eingesetzt werden.

6 Erstinstallation

6.1 Das Host-OS

Zur Installation habe ich das [Standard Image¹](#) verwendet. Davon ist schnell gebootet. Die Anmeldung mit dem Benutzer `root` verlangt kein Kennwort. Den eigentlichen Installationsprozess startet man mit `setup-alpine`. Bei den abgefragten Einstellungen benutze ich im Regelfall den Default-Wert. Abweichend davon sind

- Die Tastatur mit Layout `de` und Variante `de`
- Die Zeitzone mit `Europe/Berlin`
- Die Festplatte `sda` als `sys`

Das System ist dann in weniger als 10 Minuten aufgesetzt. Damit der Server auch immer die gleiche IP bekommt, richte ich das in meinem Router auch entsprechend ein. Damit kann ich innerhalb des Netzwerks immer auf die gleiche IP zurückgreifen. Alternativ könnte man auch eine statische IP auf dem Server selbst einrichten. Allerdings habe ich das bei keinem meiner Geräte so gemacht, sondern alle auf meinem Router entsprechend eingetragen.

6.2 Vorbereitungen

Damit ich nicht mit dem `root` Benutzer arbeiten muss, lege ich einen Benutzer für mich an. Davor installiere ich noch das Programm [`sudo`](#), damit der neue Benutzer Root Berechtigung bekommen kann. Ergänzend kommt noch [`curl`](#) hinzu, um über die Kommandozeile einfache Downloads zu ermöglichen. Außerdem wird mit [`htop`](#) noch ein übersichtlicherer Prozessmanager installiert.

¹Zum Zeitpunkt der Veröffentlichung

```
# Install some tools which are default by other linux
apk add sudo htop curl
adduser admin
visudo
# Add the line at the end
# admin ALL=(ALL) ALL
```

Abbildung 3 – Host Benutzer admin

Damit kann ich mich mit **Putty** remote an dem System anmelden und trotzdem alles mit *sudo* und Root Berechtigungen ausführen. In einem weiteren Schritt schalte ich **IPv6** ab. Die Erläuterung dazu folgt später.

```
vi /etc/sysctl.d/local.conf
# Add following line
net.ipv6.conf.all.disable_ipv6 = 1
```

Abbildung 4 – IPv6 abschalten

Zum Schluss sorgen wir noch dafür, dass die Systemzeit immer aktuell ist. Dazu holen wir einmal am Tag die Uhrzeit von einem Zeitserver. Das geht laut [Anleitung](#) so:

```
vi /etc/periodic/daily/do-ntp
# Add following lines
#!/bin/sh
ntpdate -d -q -n -p pool.ntp.org
# Make file executable
chmod +x /etc/periodic/daily/do-ntp
```

Abbildung 5 – Zeitsynchronisation

7 Die Verbindung zum NAS

Bei meinen Versuchen mit verschiedenen Konstellationen von **NFS**- und **SMB**-Shares habe ich herausgefunden, dass das alles mehr oder weniger suboptimal ist.

Docker selbst legt beim Start von Containern verschiedene mounts an. Damit gab es dann Konflikte mit Zugriffsrechten bei der Verarbeitung von Dateien. So funktionierte zwar **Portainer** ohne Probleme,

aber schon der **SCM-Manager** verweigerte den Start wegen einem Berechtigungsproblem. Auch **MariaDB** konnte ich nicht ans Laufen bringen. Und das völlig unabhängig davon, ob

- Ein Share unter `/var/lib/docker/volume` gemountet wurde,
- Ein Share unter `/mnt/nasshare` gemountet wurde,
- Als Share **NFS** verwendet wurde,
- Als Share **SMB** verwendet wurde,
- Ein bzw. mehrere absolute Pfade angegeben wurden,
- Ein bzw. mehrere **Docker Volumes mit NFS** verwendet wurden,
- Ein bzw. mehrere **Docker Volumes mit SMB** verwendet wurden.

Das Ergebnis war niederschmetternd.

Erst als ich einen Versuch mit **iSCSI** gemacht habe, waren meine Anstrengungen von Erfolg gekrönt. Mit iSCSI wird über das Netzwerk ein Blockdevice zur Verfügung gestellt. Beim Start des HostOS wird das Blockdevice dann wie eine konventionelle Festplatte in das System eingebunden. Docker merkt dann nicht, dass die Festplatte irgendwo im Netzwerk liegt und verwendet sie wie eine lokale Festplatte. Damit haben dann oben genannte Container funktioniert.

7.1 Via NFS Share

7.1.1 NFS auf dem NAS

7.1.2 NFS auf der ZBox

Bevor wir einen NFS Share nutzen können, müssen die dazu notwendigen Tools installiert sein. Dazu loggen wir uns mit dem neuen Benutzer *admin* ein und geben folgenden Befehle ein:

```
sudo apk add nfs-utils
sudo rc-update add nfsmount
sudo rc-service nfsmount start
```

Abbildung 6 – NFS Client einrichten

Damit haben wir jetzt den NFS Client eingerichtet, ganz wie es in der Anleitung dazu [hier](#) beschrieben ist.

7.1.3 NFS Share mounten

Damit dieser Share permanent verfügbar wird, müssen wir ihn auch mounten. Zuerst legen wir das Zielverzeichnis an – hierhin wird der Share gemountet. Dann tragen wir folgendes in die `/etc/fstab` ein:

```
sudo mkdir /var/lib/docker
sudo vi /etc/fstab
# Add the following line
<IP OF NAS>:<PATH OF SHARE> /var/lib/docker nfs
    auto,rw,rsize=8192,wsize=8192,timeo=14,intr 0 0
```

Abbildung 7 – NFS Share eintragen

Das Zielverzeichnis wissen wir, weil später Docker genau dort hin installiert wird. Nach einem Reboot prüfen wir, ob der Share verbunden ist:

```
zbox:~$ sudo df -h
[sudo] password for admin:
Filesystem Size Used Available Use% Mounted on
devtmpfs 10.0M 0 10.0M 0% /dev
shm 1.9G 0 1.9G 0% /dev/shm
/dev/sda3 15.6G 819.4M 14.0G 5% /
tmpfs 393.8M 144.0K 393.6M 0% /run
/dev/sda1 92.8M 22.5M 63.4M 26% /boot
<IP OF NAS>:<PATH OF SHARE>
3.6T 1.9T 1.6T 54% /var/lib/docker
```

Abbildung 8 – NFS Share verbunden

Wenn wir die letzte Zeile finden und der Share auf `/var/lib/docker` gemountet ist, haben wir alles richtig gemacht.

7.2 Via SMB Share

7.2.1 SMB auf dem NAS

7.2.2 SMB auf der ZBox

Zuerst müssen wir die notwendigen Tools installieren:

```
sudo apk add cifs-utils
```

Abbildung 9 – CIFS Utils

7.2.3 SMB Share mounten

Zuerst tragen wir den Benutzernamen und das Passwort für die Verbindung in eine eigene Datei ein. Diese ist nur für den Benutzer *root* sichtbar – die *fstab* kann jeder Benutzer einsehen.

```
sudo vi /root/.docker_smb
# Add following lines
username=<SMB USERNAME>
password=<SMB PASSWORD>
domain=<SMB DOMAIN>
# Make file readonly for root
sudo chmod 600 /root/.docker_smb
```

Abbildung 10 – SMB Zugangsdaten

Damit dieser Share permanent verfügbar wird, müssen wir ihn auch mounten. Zuerst legen wir das Zielverzeichnis */var/lib/docker* an, dann tragen wir folgendes in die */etc/fstab* ein:

```
sudo mkdir /var/lib/docker
sudo vi /etc/fstab
# Add the following line
//<IP OF NAS>/<PATH OF SHARE> /var/lib/docker cifs
    uid=0,gid=0,credentials=/root/.docker_smb,_netdev 0 0
```

Abbildung 11 – SMB Share eintragen

Da der Pfad eventuell nicht ganz klar ist, hier ein Beispiel. Das NAS hat die IP 192.168.71.75. Als Share dient das Home-Verzeichnis des

Benutzers aus der Datei `/root/.docker_smb`, die wir eben angelegt haben.

```
<//192.168.1.75/home> /var/lib/docker cifs  
uid=0,gid=0,credentials=/root/.docker_smb,_netdev 0 0
```

Abbildung 12 – SMB Share Beispiel

Damit das Netzlaufwerk beim Start automatisch verbunden wird, muss man das noch aktivieren. Das geht so:

```
# Enable mount of network shares  
sudo rc-update add netmount
```

Abbildung 13 – SMB beim Starten verbinden

Das Zielverzeichnis wissen wir, weil später Docker genau dort hin installiert wird. Nach einem Reboot prüfen wir, ob der Share verbunden ist:

```
zbox:~$ sudo df -h  
[sudo] password for admin:  
Filesystem Size Used Available Use% Mounted on  
devtmpfs 10.0M 0 10.0M 0% /dev  
shm 1.9G 0 1.9G 0% /dev/shm  
/dev/sda3 15.6G 819.4M 14.0G 5% /  
tmpfs 393.8M 144.0K 393.6M 0% /run  
/dev/sda1 92.8M 22.5M 63.4M 26% /boot  
<//IP OF NAS/PATH OF SHARE>  
3.6T 1.9T 1.6T 54% /var/lib/docker
```

Abbildung 14 – SMB Share verbunden

Und jetzt der [Bind-Mount!](#)

7.3 Via iSCSI

7.3.1 iSCSI-Target auf dem NAS anlegen

Damit auf der ZBox die virtuelle Festplatte eingebunden werden kann, muss sie vorher auf dem NAS angelegt werden. Das geht mit dem iSCSI-Manager.

iSCSI-Target auf dem NAS anlegen

Ein Server vor einem NAS

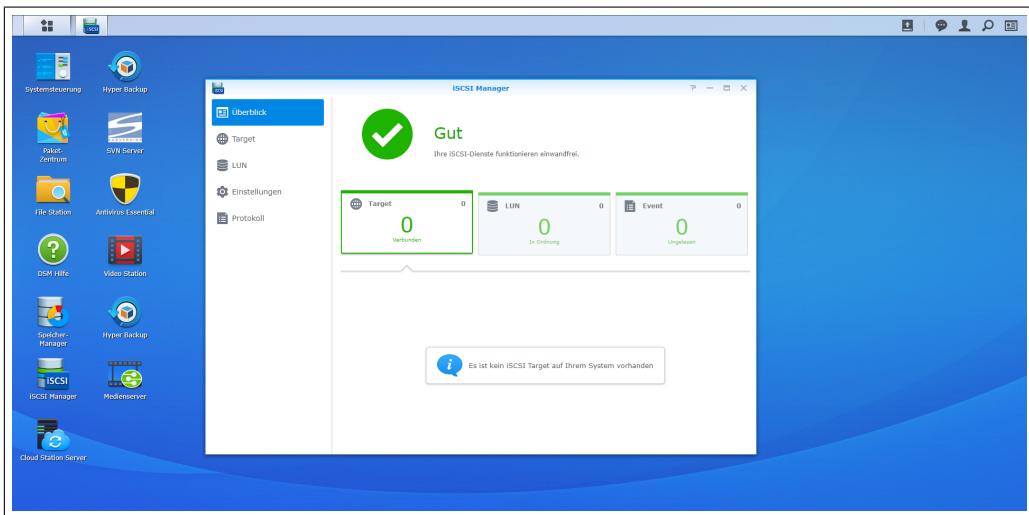


Abbildung 15 – Der initiale Start des iSCSI-Managers

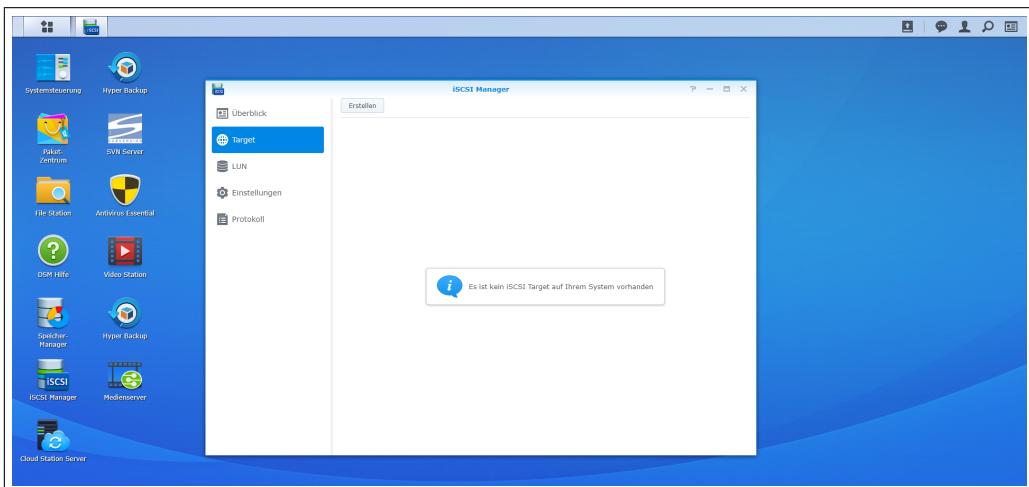


Abbildung 16 – Die iSCSI-Target Liste

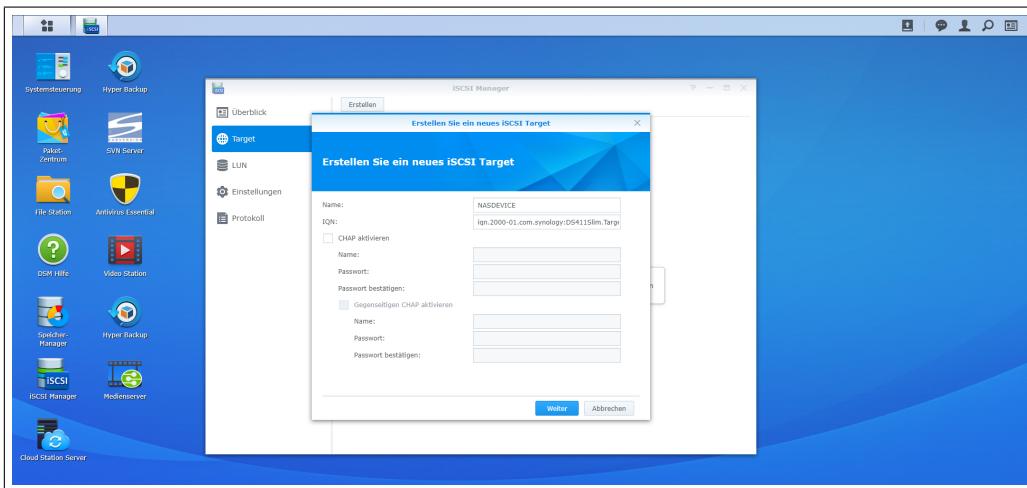


Abbildung 17 – Das iSCSI-Target anlegen

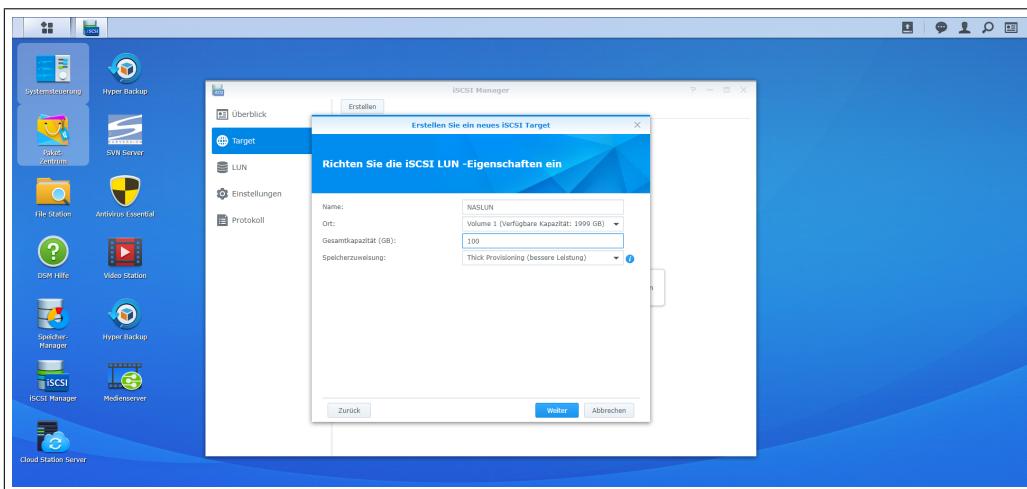


Abbildung 18 – Das dazugehörige LUN mit 100GB anlegen

Die Größe von 100 GB ist hier exemplarisch zu sehen. Je nachdem, wieviel man von der NAS-Kapazität nutzen möchte, muss das natürlich angepasst werden.

iSCSI-Target auf dem NAS anlegen

Ein Server vor einem NAS

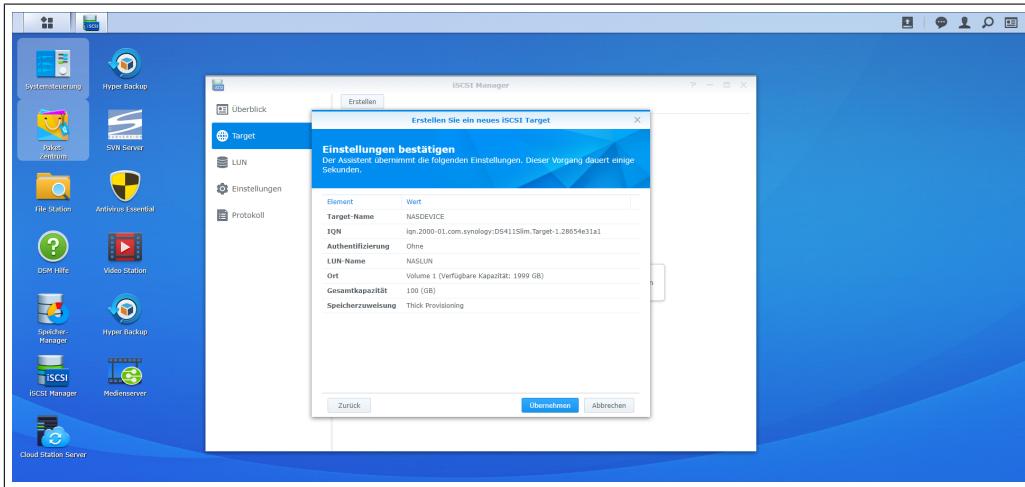


Abbildung 19 – Die Zusammenfassung

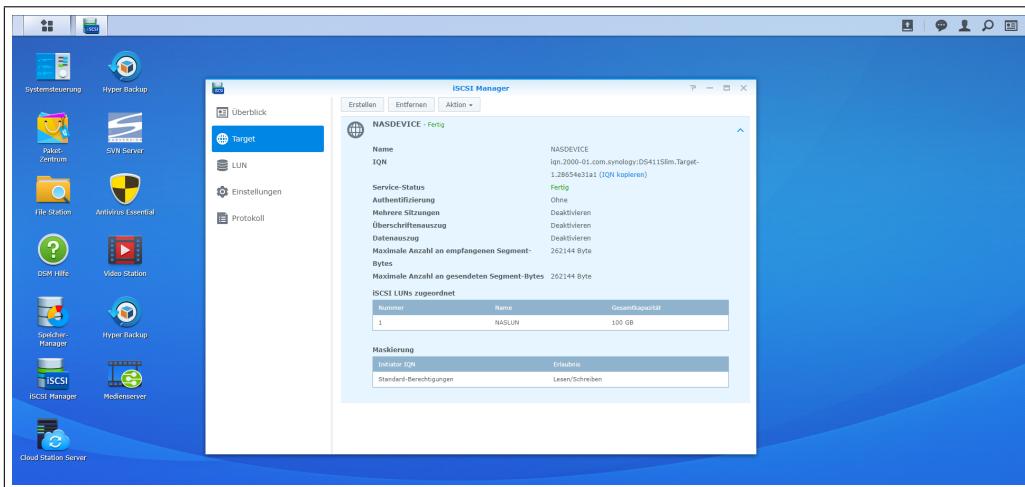


Abbildung 20 – Das Ergebnis

Wichtig ist dann noch, dass prinzipiell nur ein Netzwerkprotokoll aktiv ist. Ansonsten versucht die ZBox, sich auf jede IP zu verbinden. Da das Host OS sowohl **IPv4** als auch **IPv6** verwendet, kann das zu Problemen führen. Ich habe mich dazu entschieden, auf dem NAS ausschließlich **IPv4** zu verwenden.

7.3.2 iSCSI auf der ZBox

Nachdem das iSCSI-Target auf dem NAS angelegt wurde, wird es nun auf der ZBox eingebunden. Das Ergebnis ist eine Mischung aus der [Anleitung für eine Initiator Config](#) sowie diesem [How-To](#).

```
# Install required packages
sudo apk add open-iscsi util-linux

# Modify config to startup automatically
sudo vi /etc/iscsi/iscsid.conf
# Set node.startup = automatic

# Set initiator name
sudo vi /etc/iscsi/initiatorname.iscsi
# Replace the existing line with the name of the target provided by the NAS
# InitiatorName=iqn.2000-01.com.synology:DS411Slim.Target

# Add service to default runlevel and start it
sudo rc-update add iscsid default
sudo rc-service iscsid start
```

Abbildung 21 – Installation von open-iscsi

```
# Discover possible targets
sudo iscsidadm -m discovery -t sendtargets -p <IP OF NAS>
# Login to specific target
sudo iscsidadm -m node -T <TARGET NAME OF NAS> -l
```

Abbildung 22 – Einbinden des iSCSI-Target

```
# Figure out new attached drive
sudo lsblk --scsi
# NAME HCTL TYPE VENDOR MODEL REV TRAN
# sda 0:0:0:0 disk ATA VBOX HARDDISK 1.0 sata
# sdb 3:0:0:1 disk SYNOLOGY iSCSI Storage 3.1 iscsi
# sr0 2:0:0:0 rom VBOX CD-ROM 1.0 ata
# NAS is connected as /dev/sdb

sudo fdisk /dev/sdb
# n - Add new partition
# p - Primary partition
# 1 - Partition number
# <enter> - First sector
# <enter> - Last sector
# w - Write partition table

# Format partition 1
sudo mkfs.ext4 /dev/sdb1
```

Abbildung 23 – Filesystem einrichten

```
# Create mount point
sudo mkdir /var/lib/docker

# Figure out UUID of /dev/sdb1
sudo blkid /dev/sdb1

# Write entry to /etc/fstab
sudo vi /etc/fstab

# Add following line
# UUID=<UUID OF /dev/sdb1> /var/lib/docker ext4 _netdev 0 0

# Start netmount on boot
sudo rc-update add netmount

# Reboot the system
sudo reboot
```

Abbildung 24 – Automount einrichten

7.3.3 Nach dem Reboot

Nach einem Reboot sollte dann das Blockdevice an dem als */dev/sdb1* sichtbar sein. Die Festplatte wurde unter */var/lib/docker* eingebunden. Das sieht in etwa so aus:

```
$ df
Filesystem 1K-blocks Used Available Use% Mounted on
devtmpfs 10240 0 10240 0% /dev
shm 2023296 0 2023296 0% /dev/shm
/dev/sda3 16346492 724524 14771896 5% /
tmpfs 404660 116 404544 0% /run
/dev/sda1 95054 20749 67137 24% /boot
/dev/sdb1 102687640 61464 97366916 0% /var/lib/docker
```

Abbildung 25 – Anzeige der Laufwerke

7.4 Remote Volumes

Man kann auch die Volumes direkt auf das NAS legen.

7.4.1 NFS Volumes

Für NFS müssen wir zuerst die notwendigen Tools installieren:

```
sudo apk add nfs-utils
```

Abbildung 26 – NFS Utilities installieren

7.4.2 SMB Volumes

Für SMB müssen wir zuerst die notwendigen Tools installieren:

```
sudo apk add cifs-utils
```

Abbildung 27 – CIFS Utils

8 Docker Umgebung

Mit Docker wird die Umgebung installiert, um mit Containern arbeiten zu können. Dazu zählt nicht nur Docker selbst, sondern auch Docker Compose.

8.1 Installation von Docker

Docker ist schnell installiert – gibt es doch einen guten [Artikel](#) hierzu in dem Wiki von Alpine Linux. Zuerst muss man in `/etc/apk/repositories` noch die [Quelle](#) für Docker eintragen, dann ein `apk update` ausführen und schon kann man mit `apk add docker` Docker installieren.

Damit Docker auch bei einem Neustart gestartet wird, muss man das noch dem System mit `rc-update add docker default` mitteilen.

```
sudo vi /etc/apk/repositories
# Add the following row without the hash:
# http://dl-cdn.alpinelinux.org/alpine/edge/community
sudo apk update
sudo apk add docker
sudo rc-update add docker default
sudo service docker start
```

Abbildung 28 – Installation von Docker

8.2 Installation von Docker Compose

Um später auch mit [Docker Compose](#) arbeiten zu können, installieren wir das gleich mit. Dazu folgen wir der offiziellen [Anleitung](#).

```
sudo apk add py-pip python3-dev libffi-dev openssl-dev gcc libc-dev make
sudo pip install --upgrade pip
sudo pip install docker-compose
```

Abbildung 29 – Installation von Docker Compose

8.3 Abhängigkeiten

Da sowohl Docker als auch das Netzlaufwerk im Runlevel `default` gestartet werden, kann es unter Umständen zu einem Problem kommen: Docker wird gestartet, bevor das Netzlaufwerk verbunden ist. Das wäre sehr unangenehm, denn dann würde kein Container starten. Also sorgen wir dafür, daß das nicht zu einem Problem wird, indem wir Docker noch die Abhängigkeiten von `iscsid`, dem Daemon für das iSCSI, sowie `netmount`, dem Verbinden von Netzlaufwerken, mitteilen.

```
sudo vi /etc/init.d/docker
# Extend
#
# depend() {
# need sysfs cgroups
# }
#
# With
#
# depend() {
# need sysfs cgroups iscsid netmount
# need sysfs cgroups nfsmount
# }
```

Abbildung 30 – Docker Abhängigkeiten

9 Die Container

Hier wird die Installation von verschiedenen Containern beschrieben, welche ich auf meinem Server verwenden möchte.

9.1 Containerliste

Die Container sollen folgende Dienste zur Verfügung stellen – aufgeführt in der Reihenfolge, wie sie installiert werden:

- Portainer – Oberfläche für Docker.
- SCM-Manager – Oberfläche zu den Versionierungssystemen Git und Subversion.
- MariaDB – Datenbank für diverse Softwareprojekte.
- phpMyAdmin – Oberfläche für die Datenbank.
- Mosquitto – Ein MQTT Broker, ebenfalls für diverse Softwareprojekte.
- Syncthing – Filesynchronisation zwischen PC und dem Server.
- Nextcloud – Eine lokale Cloud.
- Samba – Dateiserver.
- ClamAV – (Anti-) Viren Scanner.
- Datensicherung
 - restic
 - Duplicati
- MinIO – Als Ziel für die Datensicherung.
- Pi-Hole – Werbeblocker für das lokale Netzwerk.

Abbildung 31 – Containerliste

Nicht aufgezählt sind dann die Container, die in der Zukunft noch dazu kommen werden – was auch immer mir da noch einfallen mag...

9.2 Portainer

Damit man sich nicht so mit der Kommandozeile herumschlagen muß, wird Portainer als Frontend zu Docker installiert. Damit sollte es etwas komfortabler sein, die gewünschten Ziele zu erreichen. Damit die Daten/Einstellungen persistent gemacht werden können, braucht es ein Docker Volume.

```
# Local volume
sudo docker volume create --name v_portainer

# Remote volume using cifs
sudo docker volume create \
--driver local \
--opt type=cifs \
--opt device=/192.168.71.10/dockerfs/v_portainer \
--opt o=username=docker,password=cul8er,file_mode=0777,dir_mode=0777 \
--name v_portainer

# Remote volume using nfs
sudo docker volume create \
--driver local \
--opt type=nfs4 \
--opt o=addr=192.168.71.10,rw \
--opt device=/volume1/dockerfs/v_portainer \
--name v_portainer

# Creation of container
sudo docker run -dit \
--restart unless-stopped \
--name portainer \
-p 9000:9000 \
-v /var/run/docker.sock:/var/run/docker.sock \
-v v_portainer:/data \
-d portainer/portainer:latest
```

Abbildung 32 – Installation von Portainer

Wichtig ist hierbei der Teil `--restart unless-stopped` – damit wird der Docker Container automatisch beim Hochfahren des Systems gestartet. Portainer selbst ist eine Webapplikation, die man über den entsprechenden Port (Default ist 9000) aufrufen kann. Bei der ersten Anmeldung muss man einen Account anlegen und ein Passwort dazu vergeben.

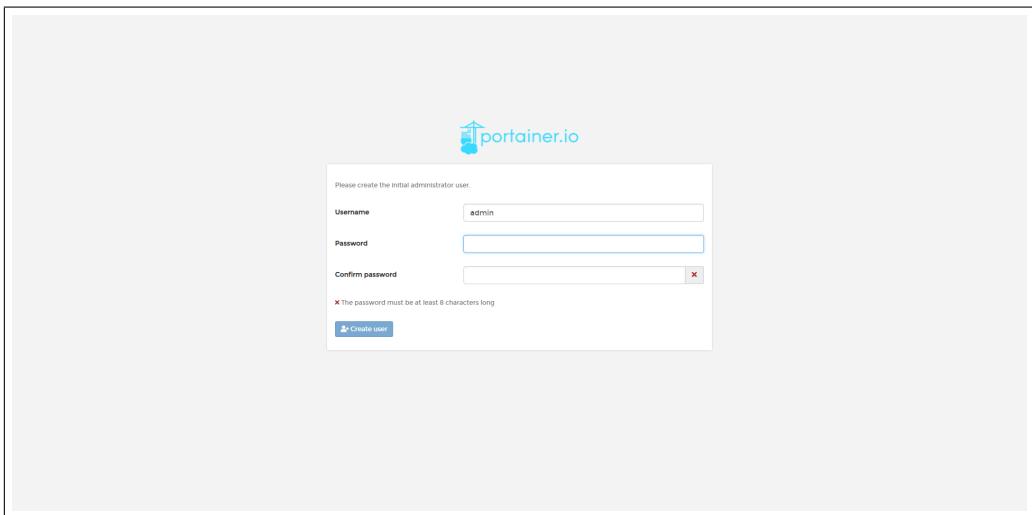


Abbildung 33 – Erste Anmeldung an Portainer

Im zweiten Schritt gibt man an, ob man mit Portainer den lokalen Host oder einen Remote Host verwalten möchte. Hier ist ganz klar der lokale Host gewünscht.

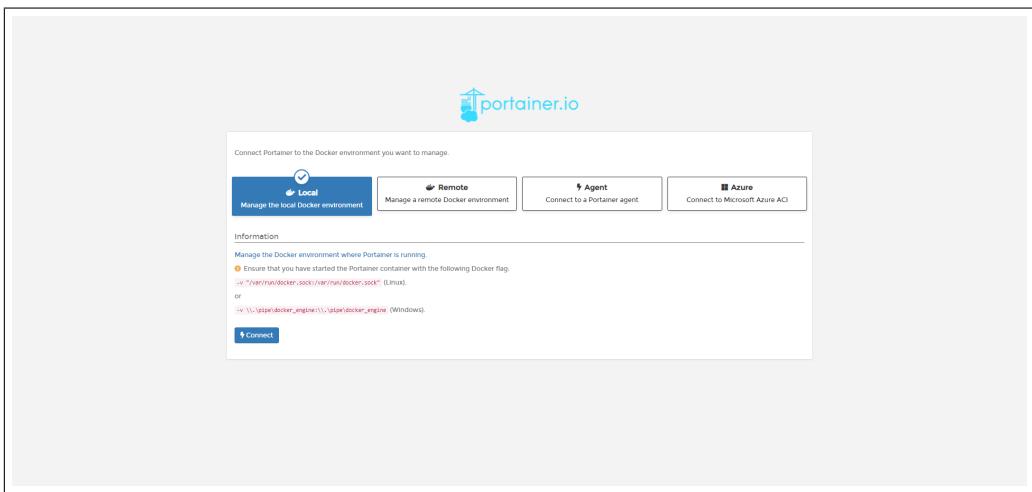


Abbildung 34 – Verbindung von Portainer an Localhost

Nachdem die „Formalitäten“ erledigt sind, kann man Portainer verwenden. Die Startseite sieht für unser System erst einmal so aus:

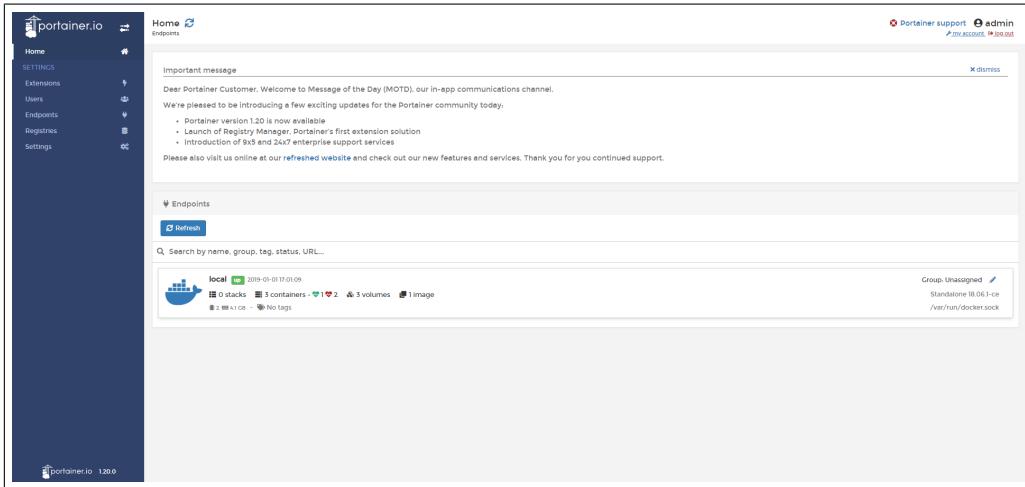


Abbildung 35 – Die Portainer Startseite

9.3 Versionierungssystem

Als **Versionsverwaltung** möchte ich ein Subversion einsetzen. Und zwar mit einer Weboberfläche. **SCM-Manager** scheint hierfür ganz gut geeignet zu sein. Damit habe ich später die Möglichkeit, von Subversion auf Git zu wechseln.

9.3.1 Installation des Docker Containers

Verwendet wird der offizielle **Docker Container** des SCM-Managers.

```

sudo docker volume create --name v_scm_manager

sudo docker volume create \
--driver local \
--opt type=cifs \
--opt device=/192.168.71.10/dockerfs/v_scm_manager \
--opt o=username=docker,password=cul8er,file_mode=0777,dir_mode=0777 \
--name v_scm_manager

sudo docker volume create \
--driver local \
--opt type=nfs4 \
--opt o=addr=192.168.71.10,rw \
--opt device=/volume1/dockerfs/v_scm_manager \
--name v_scm_manager

sudo docker run -dit \
--restart unless-stopped \
--name scm_manager \
-p 8080:8080 \
-v v_scm_manager:/var/lib/scm \
-d sorra/scm-manager:latest

```

Abbildung 36 – Installation von SCM-Manager

9.3.2 Initiale Konfiguration

Nachdem das Volume angelegt ist und der Container gestartet wurde, kann man über *http://<IP OF SERVER>:8080* die Weboberfläche aufrufen. Der Standard Login für den SCM-Manager lautet *scmadmin/scmadmin*. Ein Repository ist schnell aus einem Dump importiert, dazu noch ein Benutzer angelegt und schon kann man über die URL *http://<USER>@<IP OF SERVER>:8080/scm/svn/<REPOSITORY>/* das Repo auschecken und bearbeiten.

9.3.3 SCM-Manager Plugins

Der SCM-Manager lässt sich mit Plugins erweitern. Beispiele für solche Plugins sind:

- **SCM Activity** zur Anzeige der letzten Aktivitäten im Repository.
- **Statistics** zeigt statistische Informationen zum Repository.

9.4 MariaDB

Für verschiedene Dinge wird eine Datenbank benötigt. Der Docker Container, den ich ausgewählt habe, ist [yobasystems/alpine-mariadb](#),

weil damit wirklich einfach eine neue DB-Instanz aufgesetzt werden kann. Das geht ganz schnell, und zwar so:

```
sudo docker volume create \
--driver local \
--opt type=cifs \
--opt device=/192.168.71.10/dockerfs/v_mariadb \
--opt o=username=docker,password=cul8er,file_mode=0777,dir_mode=0777 \
--name v_mariadb

sudo docker run -dit \
--restart unless-stopped \
--name mariadb \
-p 3306:3306 \
-v v_mariadb:/var/lib/mysql \
-e MYSQL_ROOT_PASSWORD=cul8er \
-d yobasystems/alpine-mariadb:latest \
--character-set-server=utf8mb4 \
--collation-server=utf8mb4_unicode_ci \
--innodb-flush-method=fsync
```

Abbildung 37 – Installation von MariaDB

Damit wird das Datenbanksystem mit einem [UTF-8](#) Zeichensatz initialisiert und das *Root-Passwort* wird gesetzt. Alles weitere kann man über die Oberfläche [phpMyAdmin](#) einstellen/erledigen.

9.5 phpMyAdmin

Zur Administration der MariaDB gibt es quasi das Tool schlechthin: [phpMyAdmin](#). Und es gibt einen offiziellen [Docker Container](#) dazu. Diesen möchte ich verwenden, um die Datenbank damit zu administrieren.

```
sudo docker run -dit \
--name phpmyadmin \
--link mariadb:db \
-p 8888:80 \
phpmyadmin/phpmyadmin

sudo docker run \
--name phpmyadmin \
-d -e PMA_HOST=192.168.71.113 \
-p 8888:80 \
phpmyadmin/phpmyadmin

sudo docker run -dit \
--name phpmyadmin \
-p 8888:80 \
--link mariadb:db \
-d phpmyadmin/phpmyadmin:latest
```

Abbildung 38 – Installation von phpMyAdmin

Als Port wurde bewusst *8888* verwendet, da der Port *8080* bereits vom [Versionierungssystem](#) verwendet wird. Zudem wurde darauf verzichtet, den Container immer gestartet zu haben. Dieser soll nur bei Bedarf gestartet und danach wieder beendet werden. Deswegen fehlt das Argument *-restart unless-stopped* im Aufruf. Die Anmeldung am UI erfolgt mit Benutername/Passwort *root/<ROOT PASSWORT>* – dem Passwort, welches bei der Erstellung des [MariaDB](#) Containers angegeben wurde.

9.6 Mosquitto

Da das [IoT](#), also das Internet der Dinge, immer mehr an Bedeutung gewinnt, habe ich mich für diverse Softwareprojekte auch mit diesem Thema auseinander gesetzt. Dazu gehört unter anderem das Protokoll [MQTT](#). [Mosquitto](#) ist ein Broker, quasi das Herzstück dazu. Verwendet wird ist das offizielle [eclipse-mosquitto](#) Image.

```
sudo docker volume create --name v_mosquitto_config
sudo docker volume create --name v_mosquitto_data
sudo docker volume create --name v_mosquitto_log
sudo docker run -dit \
  --restart unless-stopped \
  --name mosquitto \
  -p 1883:1883 \
  -p 9001:9001 \
  -v v_mosquitto_config:/mosquitto/config \
  -v v_mosquitto_data:/mosquitto/data \
  -v v_mosquitto_log:/mosquitto/log \
  -d eclipse-mosquitto:latest
```

Abbildung 39 – Installation von Mosquitto

Damit ist der MQTT Broker einsatzbereit und kann beispielsweise mit dem [MQTT Explorer](#) getestet werden.

9.7 Syncthing

Das Synology NAS bringt von Haus aus eine Software mit, über die lokale Ordner und Dateien auf das NAS synchronisiert werden können. Da das NAS aber nur noch als Fileserver dienen soll, muss dazu eine Alternative verwendet werden. [Syncthing](#) scheint mir dazu gut geeignet zu sein. Vor allem, da man diese Software nicht nur auf dem PC, sondern auch auf dem Smartphone laufen lassen kann.

```
sudo docker volume create --name v_syncthing
sudo docker run -dit \
  --restart unless-stopped \
  --name syncthing \
  -p 8384:8384 \
  -p 22000:22000 \
  -v v_syncthing:/var/syncthing \
  -d syncthing/syncthing:latest
```

Abbildung 40 – Installation von Syncthing

Damit ist Syncthing initial eingerichtet. Die Startseite <http://<IP OF SERVER>:8384> von Syncthing meldet sich dann so:

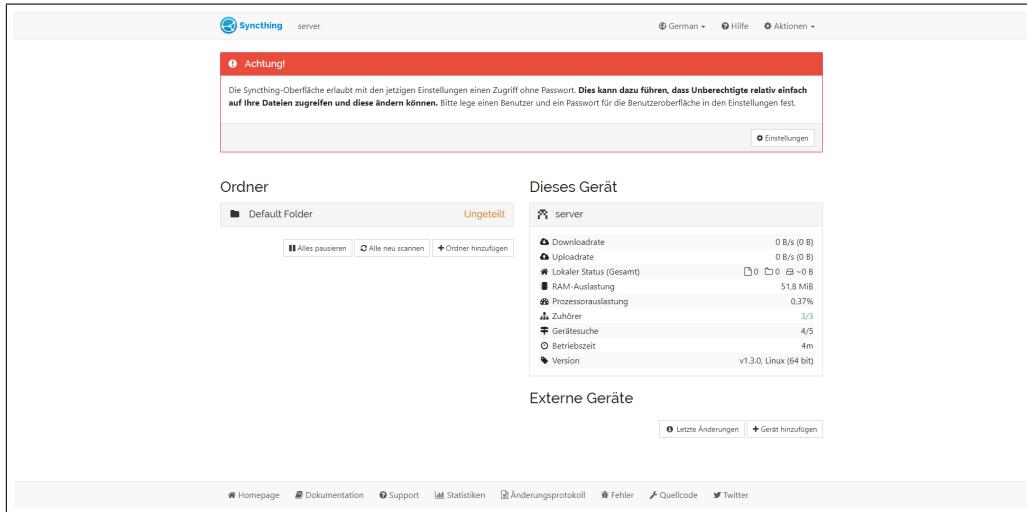


Abbildung 41 – Syncthing, erster Aufruf

Es wird empfohlen, für die Oberfläche ein Kennwort zu vergeben. Das geht über das Menü oben rechts, *Aktionen*, *Einstellungen* und dort den Reiter *GUI*:

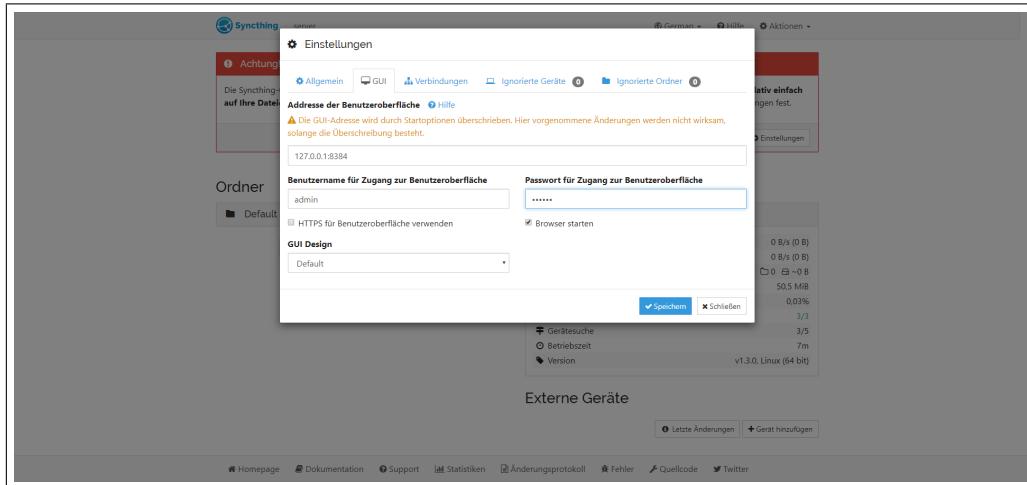
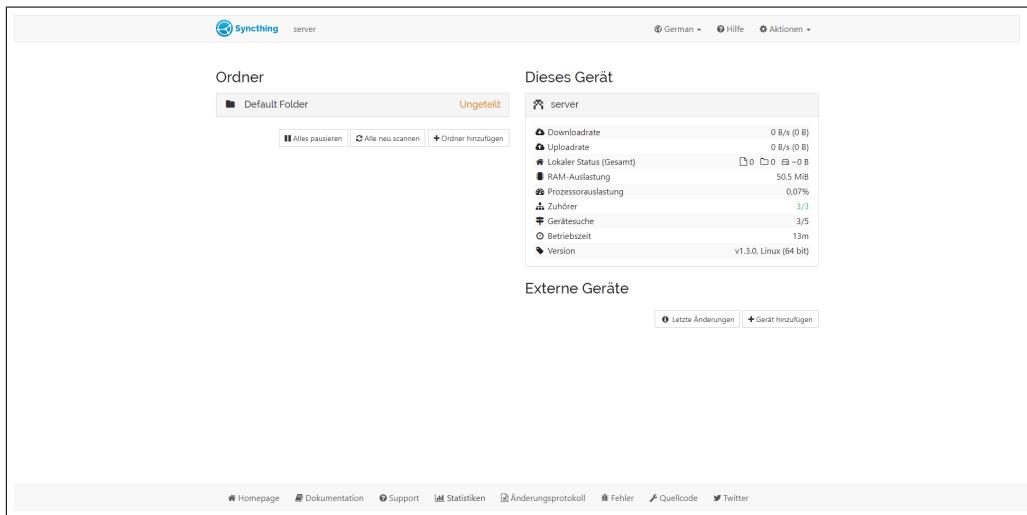


Abbildung 42 – Syncthing, GUI Kennwort

Danach meldet sich Syncthing mit der Standard Startseite. Die weitere Konfiguration bleibt dem Benutzer überlassen.

**Abbildung 43 – Syncthing, Standard Startseite**

Als Client für Windows Systeme wird auf der [Syncthing Seite](#) das Programm [SyncTrayzor](#) empfohlen.

9.8 NextCloud

Die lokale Cloud basiert auf [NextCloud](#). Von NextCloud wird ein offizielles Image zur Verfügung gestellt. Hier möchte ich die bereits installierte Version der Datenbank MariaDB nutzen und dort eine Datenbank für NextCloud zur Verfügung stellen. Deswegen lege ich dort eine Datenbank *nextcloud* an. Damit die Cloud auch Zugriff darauf hat, wird der Benutzer *nextcloud* mit dem Kennwort *<SECRET PASSWORD HERE>* verwendet.

```
sudo docker volume create --name v_nextcloud
sudo docker run -dit \
--restart unless-stopped \
--name nextcloud \
-e MYSQL_DATABASE=nextcloud \
-e MYSQL_USER=nextcloud \
-e MYSQL_PASSWORD=<SECRET PASSWORD HERE> \
-e MYSQL_HOST=<IP OF SERVER>:3306 \
-p 8765:80 \
-v v_nextcloud:/var/www/html \
-d nextcloud:latest
```

Abbildung 44 – Installation von NextCloud

Wenn man das [All in One Docker File](#) verwendet, sollte man vor dem ersten Aufruf von NextCloud natürlich über [phpMyAdmin](#) den Benutzer und die Datenbank angelegt haben.

9.9 Dateiserver

Mit dem Dateiserver habe ich mich lange Zeit schwer getan. Das ist der eigentliche Grund, warum das Projekt so viel Zeit in Anspruch nimmt. Denn normalerweise legen die Docker Container ihre Dateien lokal ab. Meine Datenmenge, also meine Filme, Musik und sonstige Dateien passen nicht lokal auf die SSD der ZBox. Die müssen also nach wie vor auf dem NAS liegen. Durch Experimente habe ich dann den für mich besten Weg gefunden: Die Einbindung mit iSCSI und den Rest mit Docker.

Bisher habe ich verschiedene Features über Docker abgebildet, welche auf meinem NAS laufen. Dazu zählt der [Versionierungssystem](#), die [MariaDB](#), die Administrationsoberfläche [phpMyAdmin](#) dazu, ebenso wie die Administrationsoberfläche für Docker, [Portainer](#).

Den eigentlichen Dateiserver habe ich bis jetzt noch nicht angepackt. Da auf meine NAS bereits ein [Samba](#) lief, bin ich davon ausgegangen, einen Container zu schnappen, die Konfiguration zu übertragen und dann klappt alles. Das war mal wieder ein typischer Fall von „Denkste!“ – so einfach ist das leider nicht.

Das liegt zum einen daran, dass Synology eine eigene Version von Samba einsetzt. Damit sind Teile der Konfigurationsdatei von dem NAS nicht mit einem offiziellen Samba zu verwenden. Zum anderen liegt das aber auch daran, dass Samba eben keine einfache Applikation ist. Somit wird ein flexibler Container benötigt, der auch die Anforderungen erfüllt.

9.9.1 Samba Image

Zum Glück hat [David Personette](#) einen solchen [Container](#) bereits erstellt. Vielen Dank an dieser Stelle. Diesen Container werde ich dann für mein Projekt verwenden. Grundsätzlich wird das eine mehrteilige Aufgabe werden. Zum einen werde ich den Container in meinen Stack integrieren. Zum anderen werde ich verschiedene Shell Skripte erstellen, um Benutzer, Shares und anderes in dem Container zu

konfigurieren. Der Container lässt sich mit dem Image von David Personette ganz einfach einrichten:

```
sudo docker volume create --name v_samba_cache
sudo docker volume create --name v_samba_data
sudo docker volume create --name v_samba_etc
sudo docker volume create --name v_samba_lib
sudo docker volume create --name v_samba_log
sudo docker volume create --name v_samba_run
sudo docker run -dit \
    --name samba \
    -p 139:139 \
    -p 445:445 \
    -e TZ=DE \
    -m 512m \
    -v v_samba_cache:/var/cache/samba \
    -v v_samba_data:/mount \
    -v v_samba_etc:/etc \
    -v v_samba_lib:/var/lib/samba \
    -v v_samba_log:/var/log/samba \
    -v v_samba_run:/run/samba \
    -d dperson/samba:latest
sudo docker exec -i -t samba /usr/bin/samba.sh -w "WLP"
sudo docker exec -i -t samba /usr/bin/samba.sh -g "log level = 2"
sudo docker exec -i -t samba /usr/bin/samba.sh -u "<USER>;<SECRET PASSWORD HERE>"
sudo docker exec -i -t samba /usr/bin/samba.sh -s "public;/share"
sudo docker container restart samba
```

Abbildung 45 – Samba Container

9.10 ClamAV

9.11 restic

9.12 MinIO

9.13 Pi-Hole

10 All in One Docker File

Die Container habe ich zuerst manuell eingerichtet, um die für mich relevanten Einstellungen herauszufinden. Aber bei meinen vielen Versuchen in einer [Virtual Box](#) wurde mir das auf die Dauer zu lästig. Zudem wollte ich auf meiner ZBox nicht ebenfalls alles manuell einrichten. Deswegen habe ich die einzelnen Schritte bzw. Container in einer Docker Compose Datei zusammengefasst. Die sieht dann in etwa so aus:

```

version: '3.7'

services:
  dui:
    container_name: "portainer"
    image: portainer/portainer:latest
    ports:
      - "9000:9000"
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
      - v_portainer:/data
    restart: always

  scm:
    container_name: "scm-manager"
    image: sdorra/scm-manager:latest
    ports:
      - "8080:8080"
    volumes:
      - v_scm_manager:/var/lib/scm
    restart: always

  db:
    container_name: "mariadb"
    image: yobasystems/alpine-mariadb:latest
    ports:
      - "3306:3306"
    volumes:
      - v_mariadb:/var/lib/mysql
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: <SECRET PASSWORD HERE>
    command: [ '--character-set-server=utf8mb4', '--collation-server=utf8mb4_unicode_ci' ]

  dbui:
    container_name: "phpmyadmin"
    depends_on:
      - db
    links:
      - db
    ports:
      - "8888:80"
    image: phpmyadmin/phpmyadmin:latest
    restart: always

  mqtt:
    container_name: "mosquitto"
    image: eclipse-mosquitto:latest
    ports:
      - "1883:1883"
      - "9001:9001"
    restart: always
    volumes:
      - v_mosquitto_config:/mosquitto/config
      - v_mosquitto_data:/mosquitto/data
      - v_mosquitto_log:/mosquitto/log

  st:
    container_name: "syncthing"
    image: syncthing/syncthing:latest
    ports:

```

```

- "8384:8384"
- "22000:22000"
restart: always
volumes:
- v_syncthing:/var/syncthing

nc:
  container_name: "nextcloud"
  environment:
    MYSQL_DATABASE: nextcloud
    MYSQL_USER: nextcloud
    MYSQL_PASSWORD: <SECRET PASSWORD HERE>
    MYSQL_HOST: <IP OF SERVER>:3306
  image: nextcloud:latest
  ports:
- "8765:80"
  restart: always
  volumes:
- v_nextcloud:/var/www/html

volumes:
  v_mariadb:
  v_mosquitto_config:
  v_mosquitto_log:
  v_mosquitto_data:
  v_nextcloud:
  v_portainer:
  v_scm_manager:
  v_syncthing:

```

Listing 1 – All in One Setup

11 FAQ

Q: Wie bekomme ich eine Shell für bzw. in einen Container?

A: Dazu gibt man in der Kommandozeile folgendes ein:

```
# Maybe there is no bash available, then try with /bin/sh
sudo docker exec -i -t <NAME OF CONTAINER> /bin/bash
```

Abbildung 46 – Container Shell

Q: Wie kann ich das MariaDB root-Passwort ändern?

A: Dazu loggt man sich mit dem root-Account und dem bekannten Passwort ein. Danach führt man folgende SQL Statements aus:

```
UPDATE mysql.user SET Password=PASSWORD('<NEW PASSWORD HERE>') WHERE User='root';
FLUSH PRIVILEGES;
```

Abbildung 47 – MariaDB Root Password ändern

Q: Was ist ein Docker Stack?

A: Als einen Docker Stack bezeichnet man einen oder mehrere Container, die mit weiteren Einstellungen in einer *docker-compose.yml* Datei zusammengefasst sind.

Q: Wie kann ich einen Docker Stack starten/anhalten?

A: Dazu führt man auf der Kommandozeile folgendes aus:

```
cd <NAME OF STACK>
# To start the docker stack
# The option '-d' runs the stack in the background
sudo docker-compose up -d
# To stop the docker stack
sudo docker-compose stop
```

Abbildung 48 – Start/Stop Docker Stack

Q: Wie kann ich eine Datei aus einem Docker Container kopieren?

A: Dazu führt man auf der Kommandozeile folgendes aus:

```
sudo docker cp <NAME OF CONTAINER>:<FILE PATH WITHIN CONTAINER> <HOST PATH TARGET>
```

Abbildung 49 – Datei aus Container kopieren

Abbildungsverzeichnis

1	Aktueller Zustand	4
2	Zielvorstellung	6
3	Host Benutzer admin	8
4	IPv6 abschalten	8
5	Zeitsynchronisation	8
6	NFS Client einrichten	9
7	NFS Share eintragen	10
8	NFS Share verbunden	10
9	CIFS Utils	11
10	SMB Zugangsdaten	11
11	SMB Share eintragen	11
12	SMB Share Beispiel	12
13	SMB beim Starten verbinden	12
14	SMB Share verbunden	12
15	Der initiale Start des iSCSI-Managers	13
16	Die iSCSI-Target Liste	13
17	Das iSCSI-Target anlegen	14
18	Das dazugehörige LUN mit 100GB anlegen	14
19	Die Zusammenfassung	15
20	Das Ergebnis	15
21	Installation von open-iscsi	16
22	Einbinden des iSCSI-Target	16
23	Filesystem einrichten	17
24	Autounmount einrichten	17
25	Anzeige der Laufwerke	18
26	NFS Utilities installieren	18
27	CIFS Utils	18
28	Installation von Docker	19
29	Installation von Docker Compose	19
30	Docker Abhängigkeiten	20
31	Containerliste	21
32	Installation von Portainer	22
33	Erste Anmeldung an Portainer	23
34	Verbindung von Portainer an Localhost	23
35	Die Portainer Startseite	24
36	Installation von SCM-Manager	25
37	Installation von MariaDB	26
38	Installation von phpMyAdmin	27

39	Installation von Mosquitto	28
40	Installation von Syncthing	28
41	Syncthing, erster Aufruf	29
42	Syncthing, GUI Kennwort	29
43	Syncthing, Standard Startseite	30
44	Installation von NextCloud	30
45	Samba Container	32
46	Container Shell	34
47	MariaDB Root Password ändern	35
48	Start/Stop Docker Stack	35
49	Datei aus Container kopieren	35

Tabellenverzeichnis

1 Vergleich CPU/RAM	4
-------------------------------	---