Raspberry - Systeme

Softwarelösungen

- Raspberry Pi als PXE-Server
- Raspi-Cam-- SSH-Tunnel--div
- Anpassung PI-Hole
- Docker auf dem Raspberry
- Setup a Raspberry Pi Zero W to run a Web Browser in Kiosk Mode
- Zerocam

Raspberry Pi als PXE-Server

© Erik Reis, 123RF

Booten übers Netz

Friedrich Hotz

Via PXE booten oder installieren Sie auch auf Rechnern ohne Wechselspeicher ein Betriebssystem. Mit dem Raspberry Pi steht Ihnen dafür ein sehr kostengünstiger Server zur Verfügung.

README

PXE erlaubt es, bootbare ISO-Images im Netz bereitzustellen und diese auf den Client-Rechnern zu starten. Der Artikel zeigt Ihnen, wie Sie den RasPi zum PXE-Server aufrüsten.

Anfang 1999 schlug Intel der Internet Engineering Task Force die Gründung einer Arbeitsgruppe vor, die ein Verfahren zum Booten eines Rechners vom Netzwerk über DHCP definieren sollte. Als technische Grundlage dazu präsentierte Intel seine PXE-Technik. Nachdem diese Arbeitsgruppe nicht zustande kam, veröffentlichte Intel im September 1999 im Alleingang die PXE-Spezifikation, die ursprünglich nur für die 32-Bit-Intel-Architektur (IA32) galt. Innerhalb von UEFI wird sie mittlerweile auch für IA64 unterstützt, daneben brachten entsprechende Portierungen PXE mittlerweile auch auf eine Reihe anderer Hardware-Plattformen.

Der PXE-Code befindet sich in einem ROM auf dem Client-Rechner (BIOS oder UEFI). Über eine Reihe von Anfragen an den Server ermittelt der Client dessen IP-Adresse und die zu ladende Startdatei. Letztere, die der Rechner über einen vereinfachten FTP-Transfer beim Server abholt (Abbildung 1), übernimmt dann die weitere Steuerung. Die exakte Vorgehensweise führt die PXE-Spezifikation [1] auf.

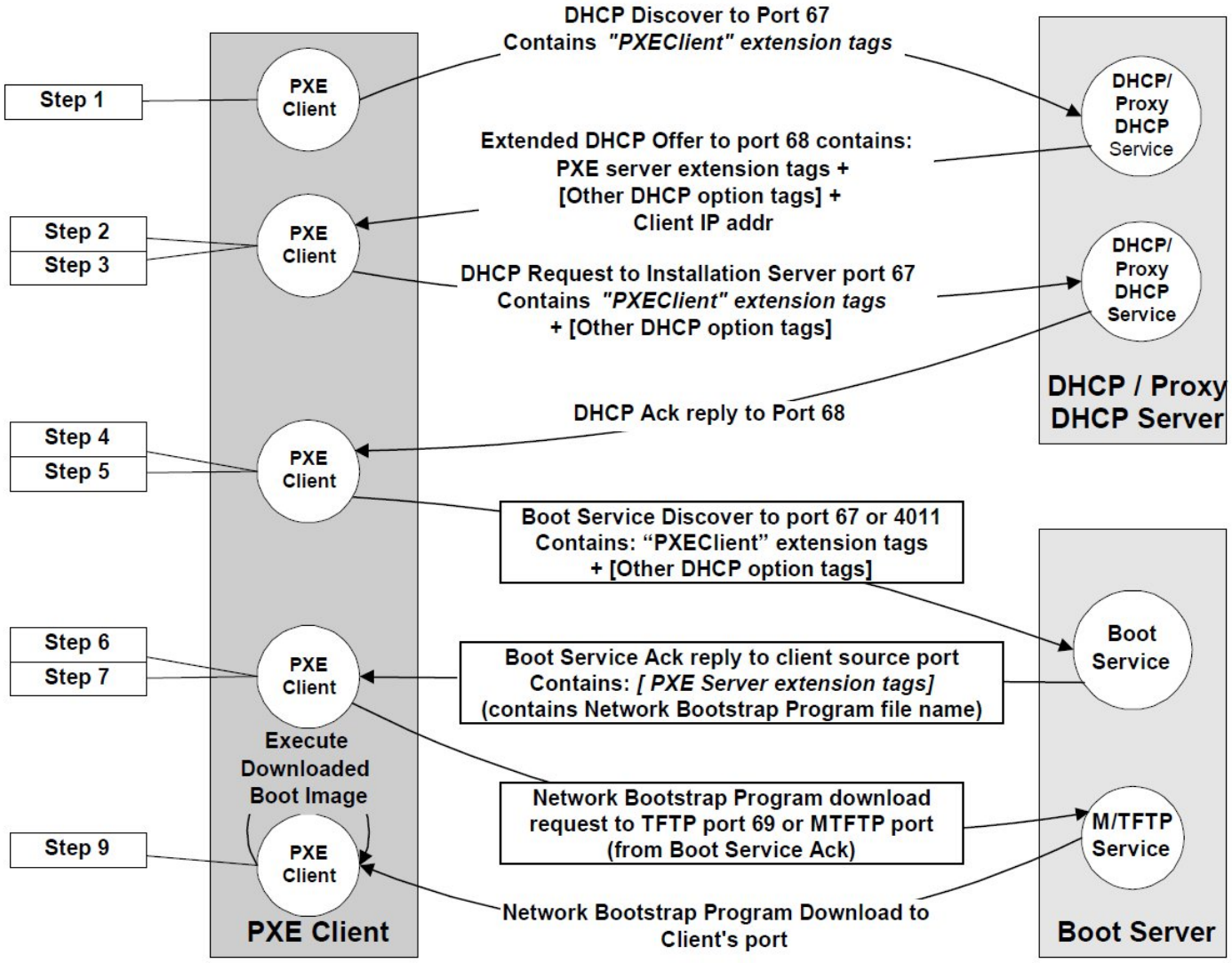


Abbildung 1: Der Boot-Vorgang via PXE im Diagramm. (Quelle: Intel)

DHCP PXE-fähig machen

Auf der Server-Seite, also auf dem Raspberry Pi, benötigen Sie als Grundlage einen installierten und konfigurierten DHCP-Server [2]. Mit dem Zuweisen der IP-Adresse übermittelt dieser DHCP-Server, wo sich ein Boot-Image befindet. Dazu muss auf dem Server ein TFTP-Dienst laufen, der diese Datei zum Client überträgt.

Die DHCP-Konfiguration benötigt nur eine kleine Modifikation, um PXE-Boot zu unterstützen: Im Abschnitt `subnet` der `/etc/dhcp/dhcpd.conf` geben Sie den Hostnamen des TFTP-Servers (in unserem Beispiel `pi.homenet.de`) sowie den Namen der Boot-Image-Datei an (Listing 1). Nach dieser Konfigurationsänderung starten Sie den Server mit dem Befehl `sudo service isc-dhcp-server restart` neu.

Listing 1

```
[...]
#
# add PXE-Boot support
#
  next-server pi.homenet.de;
  filename "pxelinux.0";
}
```

TFTP

Damit der RasPi in der Lage ist, das Image und die Konfigurationsdatei auszuliefern, installieren Sie zunächst mit dem Kommando `sudo apt-get install tftpd-hpa` den TFTP-Server darauf. Der TFTP-Server erwartet in der Voreinstellung einen IPv6-Socket, was zunächst zu einer Fehlermeldung führt. Um dies abzustellen, ergänzen Sie in der Datei `/etc/default/tftpd-hpa` die Zeile `TFTP_OPTIONS` am Schluss um den Eintrag `--ipv4`:

```
TFTP_OPTIONS="--secure --ipv4"
```

Nach dieser Änderung startet der TFTP-Dienst ohne weitere Fehlermeldungen.

Syslinux

Um an die Datei `pxelinux.0` (und einige weitere) zu gelangen, gilt es, das Syslinux-Paket auf dem Raspberry Pi zu installieren. Danach kopieren Sie die benötigten Dateien an den richtigen Ort (Listing 2)

Listing 2

```
$ sudo apt-get install syslinux-common
$ sudo cp /usr/lib/syslinux/chain.c32 /usr/lib/syslinux/menu.c32 /usr/lib/syslinux/vesamenu.c32
$ sudo mkdir --p /srv/tftp/pxelinux.cfg
```

Danach erstellen Sie eine recht einfach gehaltene Isolinux-Menüdatei `/srv/tfpt/pxelinux.cfg/default`, die lediglich die Optionen *Von Festplatte starten* und *Speichertest* als Boot-Optionen anbietet (Listing 3).

Listing 3

```
DEFAULT vesamenu.c32
PROMPT 0
TIMEOUT 300
MENU TITLE PXE Network Boot Menu
LABEL local
MENU LABEL Von Festplatte starten
MENU DEFAULT
LOCALBOOT 0
LABEL memtest
MENU LABEL Speichertest mit memtest86+ v4.20
KERNEL memtest
```

Bootet ein Rechner übers Netz mit dieser Konfiguration, zeigt er einen entsprechenden Bootscreen an (Abbildung 2). Klappt das soweit, können Sie nun weitere Betriebssystem-Images auf den RasPi laden und in die Startdatei eintragen.

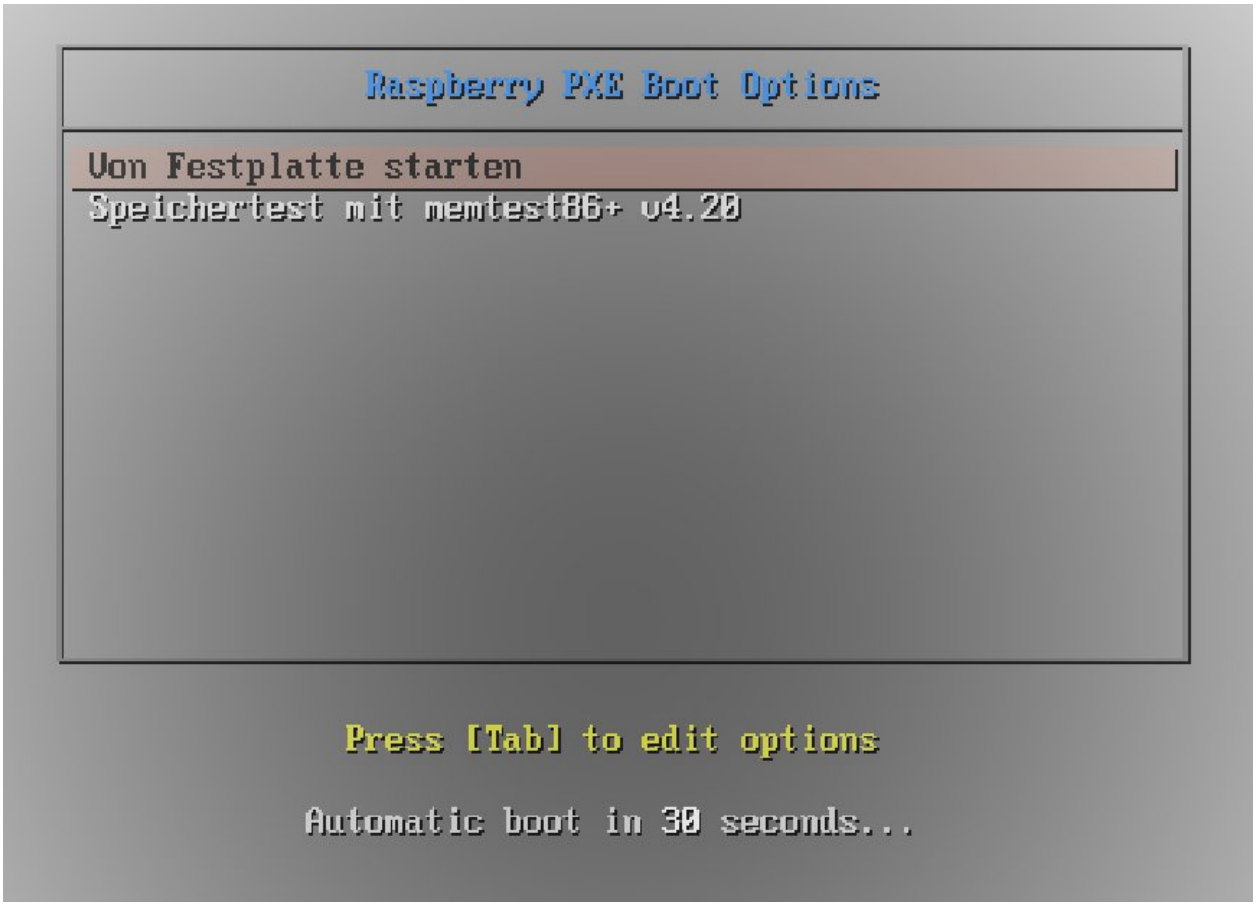


Abbildung 2: Findet Ihr Client-Rechner im Netz via PXE die nötigen Start-Dateien, zeigt er einen entsprechenden Bootscreen an.

PXE-Client einrichten

Aufseiten des Clients müssen Sie für das Booten über PXE lediglich die BIOS-Einstellungen entsprechend anpassen, was eventuell ein manuelles Aktivieren des Boot-RAMs der Netzwerkkarte voraussetzt. Nach der Umstellung startet der Rechner dann dauerhaft via PXE.

Viele Rechner bieten daneben beim Hochfahren auch auf Tastendruck ein Menü für die Auswahl des Bootmediums an, sodass Sie auf diesem Weg gezielt im Einzelfall via PXE booten können. Dell und IBM verwenden dazu in der Regel [F12], Asus nutzt [F8], und HP legt die Funktion meist auf [F9].

MSDOS-ISO booten

Für erste Versuche mit einem bootbaren Betriebssystem-Images eignet sich das kompakte, nur 3 MByte große MSDOS-Bootdisk bestens. Sie finden ein entsprechendes ISO-Image beispielsweise bei Allbootdisks.com [3], laden es dort herunter und kopieren es dann nach `/srv/tftp/DOS/`.

Mithilfe des kleinen Programms Memdisk lädt der Rechner ISO-Images in den Speicher und startet diese von dort. Sie finden das Tool unter `/usr/lib/syslinux/memdisk` und kopieren es von dort aus nach `/srv/tftp/memdisk`. Nun fehlt nur noch ein korrespondierender Eintrag in der Menüdatei `/srv/tfpt/pxelinux.cfg/default`, um den Clients das MSDOS-Image zum Booten anzubieten (Listing 4).

Listing 4

```
[...]
LABEL MSDOS 6.22
MENU LABEL MSDOS 6.22 starten
KERNEL memdisk
APPEND iso raw initrd=DOS/DOS6.22_bootdisk.iso
```

Aktivieren Sie beim Start des Clients den entsprechenden Eintrag am Boot-Prompt, startet DOS 6.22 auf dem Rechner übers Netz.

Ausblick

Auf dem gezeigten Weg lassen sich auch etliche Linux-Distributionen via PXE booten, jedoch bei Weitem nicht alle. In vielen Fällen ist es notwendig, das ISO-Image als Network Block Device einzubinden und das ISO über die aus dem Image extrahierte initiale RAM-Disk zu starten. Wie das funktioniert, zeigt ein Artikel in einer der nächsten Ausgaben von Raspberry Pi Geek.

Glossar

PXE

> Preboot Execution Environment. Eine Methode zum Booten eines Rechners vom Netzwerk über DHCP. PXE nutzt neben DHCP auch noch die Protokolle UDP und TFTP.

UEFI

> Unified Extensible Firmware Interface. Dieser BIOS-Ersatz basiert auf dem ursprünglich von Intel entwickelten EFI, wird aber mittlerweile von einer ganzen Reihe von PC- und BIOS-Herstellern unterstützt, daher das “Unified”.

Infos

1. PXE-Spezifikationen: http://download.intel.com/design/archives/wfm/downloads/pxespec.pdf
2. DHCP-Server einrichten: https://wiki.debian.org/de/DHCP_Server
3. MSDOS-Boot-Image: http://www.allbootdisks.com/download/iso.html

Raspi-Cam-- SSH-Tunnel--div

SSH-Tunnel raspimc: (root crontab)

@reboot sleep 20 &&/usr/bin/mount_all
@reboot autossh -M 0 -f -o ConnectTimeout=10 -o ServerAliveInterval=60 -o ServerAliveCountMax=2 -p 48153 -N -R 8081:localhost:22 root@meet.wkmimnl.de
@reboot autossh -M 0 -f -o ConnectTimeout=10 -o ServerAliveInterval=60 -o ServerAliveCountMax=2 -p 48153 -N -R 8082:localhost:8080 root@meet.wkmimnl.de
@reboot autossh -M 0 -f -o ConnectTimeout=10 -o ServerAliveInterval=60 -o ServerAliveCountMax=2 -p 48153 -N -R 8083:localhost:6875 root@meet.wkmimnl.de

Images auf SD schreiben - von SD ertstellen:

```
$ dd if=/dev/sdc of=raspbian.img bs=1M
$ dd if=raspbian.img of=/dev/sdc bs=1M
$ sync
```

Anpassung PI-Hole

Zwei Dateien sind zu editieren:

Feste IP: /etc/dhcpcd.conf

```
  #Fixe IP
static ip_address=192.168.xxx.xxx/24
static routers=192.168.x.x
static domain_name_servers=127.0.0.1  8.8.8.8
```

IP-Einträge anpassen.

Wlan: /etc/wpa_supplicant/wpa_supplicant.conf

```
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1

network={
    ssid="SSID"
    psk=############wlan-schlüssel

}
```

Einträge anpassen.

Docker auf dem Raspberry

Dies ist Beitrag 1 von 2 der Serie *“Raspberry Pi Docker Basics”*

In dieser Serie lernst du den Umgang mit Docker Containern auf dem Raspberry Pi

In diesem Beitrag lernst du alle Grundlagen der Containerization mit Docker auf dem Raspberry Pi.

Bei Docker handelt es sich um eine noch recht neue Visualisierungstechnologie, die bei vielen großen Unternehmen mittlerweile zum Einsatz kommt. Ich möchte dir in dieser Serie zunächst die Grundlagen von Docker erklären, wie du einfache Anwendungen wie z.B. Heimdall installierst und später dann einen DNS-Filter wie Pi-hole oder AdGuard einrichtest, einen eigenen VPN-Tunnel mit wireguard aufbaust und/oder einen eigenen lokalen GitHub-Klon Gogs und Jira-Alternative Redmine einrichtest. Dadurch lernst du Stück für Stück Docker und seine Eigenarten, sowie Portainer zur Administration von Docker kennen.

INHALTSVERZEICHNIS

Die Hardware für diesem Blog

Vom Prinzip kannst du jeden Raspberry Pi nehmen. Solltest du noch keinen Raspberry Pi besitzen, dann schau dir einmal den Raspberry Pi 400 als Kit an und bestelle auch gleich einen ausreichend

großen USB-Stick dazu. Es sollte aber zumindest ein Raspberry Pi 3, besser noch ein 4er mit mindestens 4GB RAM sein.

WARENKORB

Was unterscheidet Docker von einer virtuellen Maschine

Wenn man etwas über Docker liest, kommt man schnell an den Punkt, wo der Vergleiche zu virtuellen Maschinen aufgezeigt wird. Vom Prinzip verfolgen beide ein ähnliches Prinzip, eine Anwendung möglichst isoliert bereitzustellen.

Als erstes möchte ich dir zeigen, wie sich eine virtuelle Maschine von Docker in ihrer Architektur im Wesentlichen unterscheidet, siehe Abbildung 1.

Das Prinzip der virtuellen Maschine liegt darin, dass Software einen kompletten PC, mit all seiner Hardware emuliert. Die Applikation läuft auf dem Betriebssystem der virtuellen Maschine und ist über diese erreichbar. Problem dabei ist, dass eine virtuelle Maschine enorm hardwarehungrig ist. Meist werden zur Bereitstellung von virtuellen Maschinen, kurz VMs, Serverracks eingesetzt, die in Ihrer Hardware skalierbar sind.

Docker arbeitet da ein bisschen anders. Docker stellt Applikationen in einem Container bzw. durch das Image bereit, welches nur die nötigen Bibliotheken und Programme zur Nutzung der Applikation enthält. Man spricht hier auch von Microservice-Architecture. Dadurch sind die Images extrem klein im Vergleich zu VMs und das Starten, Stoppen und Neustarten der Applikation geht schneller als bei einer VM. Nachteil ist aber, dass Docker den Kernel vom Hostsystem nutzt, was eine VM wiederum nicht macht. Wird also ein Docker-System kompromittiert, können alle Docker-Container kompromittiert werden. Durch die Docker-Images ist es ohne große Probleme möglich, eine Applikation auf jedem beliebigen System auszuführen, was bei einer VM meist schwieriger ist.

Die Frage, wo Docker und eine VM läuft ist schnell geklärt. Ein Docker oder ein VM kann im Prinzip auf jedem System laufen. Gerade bei einer VM ist aber Power gefragt, weswegen auf einem Raspberry Pi eine VM nicht laufen wird. Docker hingegen läuft auf einem Raspberry Pi, jedoch sollte man sich die Frage stellen, ob es klug ist für z.B. ein Produktivsystem Docker tatsächlich einsetzen zu wollen. Privat nutze ich Docker in Kombination mit diversen Applikationen, jedoch gibt es viele Images, die nicht auf den Raspberry Pi ausgelegt sind. Dazu später mehr.

Die Frage, ob man nun lieber Docker oder eine VM nutzen sollte, liegt ganz im Anwendungsbereich! Generell kann Docker eine VM nicht ersetzen und umgekehrt.

Kurze Einführung in die Begriffe / Docker Raspberry Pi

Wenn du mit dem Thema Docker anfängst, wirst du wahrscheinlich schnell mit Begriffen wie Docker-Container, Docker-Image, Docker-Volumen, Bind und vielen mehr konfrontiert. Ich möchte dir an dieser Stelle den Einstieg einfach machen und die wichtigsten Grundbegriffe schon einmal vorab erklären. Wen es ins Detail geht, werde ich das in diesem und allen weiteren Blogabschnitten erklären.

Was ist ein Docker? Streng genommen ist Docker, wie du es ja schon gelesen hast, eine Methode, um eine Applikation zu virtualisieren. Damit das funktioniert, brauchst du ein sogenannten Docker-Container, welcher wiederum aus einem Docker-Image erzeugt wird. Um es einmal einfacher zu erklären, kannst du dir ein Image wie eine DVD vorstellen. Die Daten sind fest auf der DVD bzw. dem Docker-Image hinterlegt und du kannst diese erst einmal nicht modifizieren. Diese DVD legst du in ein DVD-Laufwerk, welches die Daten ausliest und bereitstellt.

Das wäre in dieser Analogie der Docker-Container. Damit ist auch geklärt, warum man öfters liest, dass Daten in einem Docker-Container flüchtig sind bzw. ein Docker-Container vergisst. Stoppe ich den Container oder starte diesen neu, werden alle Daten resettet und du bekommst den ursprünglichen Zustand von dem Docker-Image präsentiert. Damit genau das nicht passiert, kannst du Daten auch in ein Volumen oder ein Bind, ein spezieller Ordner auf dem Hostsystem, auslagern. Dieses Volumen oder Bind musst du dann in den Docker-Container mounten, weswegen der Docker-Container genau diese Daten nutzt, um weiterzuarbeiten.

In der Regel wird bei Starten des Containers geprüft, ob das Volumen oder Bind schon nutzbare Daten hat und wenn nicht, werden Default-Daten angelegt, die dann ebenfalls nach einem Neustart weiterhin vorhanden sind. Welche Pfade du in ein Volumen oder Bind auslagern musst, verrät dir die docker hub – Seite der Applikation. Dann gibt es noch die Enviorments, was vom Prinzip Parameter sind, die ein Docker-Image zwingend braucht. Diese Enviorments können bei einem Image vorhanden sein, müssen es aber nicht. Ob und welche es gibt, bzw. was ihre Funktion ist, findest du bei docker hub meist in der Beschreibung. Damit die Docker-Container mit der Außenwelt kommunizieren können, z.B. eine Datenbank oder eine Webappilkation, musst du die Ports zwischen Host und Docker-Container verknüpfen. Das Interessante dabei ist, du kannst das gleiche Docker-Image auf mehreren Docker-Containern laufen lassen aber durch andere Ports erreichbar machen.

Das klingt erst einmal sehr kompliziert und es könnte sein, dass du gerade das Gefühl hast, dass Docker eher was für IT-Nerds ist, tatsächlich ist es aber schnell gelernt und nach ein paar erstellten Docker-Containern selbsterklärend.

Docker auf dem Raspberry Pi installieren

Um Docker zu nutzen, braucht es wenige Befehle auf der Konsole. Zunächst braucht der Raspberry Pi sein Betriebssystem, meist Raspberry Pi OS, welches schnell auf die SD oder besser einem USB-Device installiert wurde. Hier empfehle ich meinen Blogbeitrag zum Thema „Den Raspberry Pi absichern“, wo ich das genau erklärt habe. Danach öffnest du das Terminal und lädst zunächst ein Skript herunter, welches Docker auf unserem Raspberry Pi installieren wird, siehe Code 1.

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

Code 1: Skript von Docker.com zum installieren

Das Skript selbst ist recht klein, mit gerade einmal 19kByte, siehe Abbildung 2, jedoch wird es gleich beim Ausführen eine Menge an unserem Raspberry Pi verändern bzw. anpassen und installieren.

Abbildung 2: get-Docker-Skript im Ordner Downloads

Zunächst führt das Skript, wenn mit Code 2 zum Starten mit root-Rechten angestoßen, ein Update aller installierten Pakete durch und installiert im Anschluss alle benötigten Pakete für Docker.

```
sudo sh get-Docker.sh
```

Code2: Das Skript get-Docker ausführen

Danach fügt es die Paketquelle von Docker, samt GPG-Schlüssel, dem Paketmanager apt hinzu, um dann Docker zu installieren. Dabei handelt es sich um die Pakete Docker-ce, Docker-ce-cli und containerd.io. Je nach Internetleitung kann die Installation einen kurzen Moment dauern, jedoch solltet ihr am Ende eine Erfolgsmeldung im Terminal sehen, siehe Abbildung 3.

Abbildung 3: Docker auf dem Raspberry Pi installieren

Damit nicht nur der Benutzer **root** Docker ausführen darf, sollte der Benutzer **pi** der Gruppe Docker hinzugefügt werden, siehe Code 3.

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

Code 3: pi der Gruppe Docker hinzufügen

Danach kannst du entweder den Raspberry Pi neustarten oder mit Code 4 die Gruppenrichtlinien neu laden.

```
newgrp docker
```

Code 4: Die Gruppenrichtlinien neu einlesen

Nun solltest du noch prüfen, ob Docker auch funktioniert. Was wäre da also besser als ein Image mit dem treffenden Namen **hello-world**? Dieses startest du mit dem Befehl aus Code 5.

```
docker run hello-world
```

Code 5: hello-world-Container im Docker starten

Wenn alles funktioniert hat, dann solltest du eine Ausgabe wie in Abbildung 4 sehen. Zugegeben, der Container macht erst einmal nicht viel, außer eine Ausgabe in der Konsole zu erzeugen.

Abbildung 4: Ausgeführter hello-world-Container

Doch sollte es keine keinerlei Fehlermeldung bis hierhin gegeben haben, so weißt du, dass Docker ordnungsgemäß installiert wurde und einsatzbereit ist.

Portainer zur Dockerverwaltung

Vom Prinzip kann Docker über die Kommandozeile genutzt werden. Egal ob um neue Container anzulegen, Images herunterzuladen oder Docker Compose zu verwalten. Aber einfacher bzw. schöner wäre doch eine grafische Oberfläche, mit der wir diese Arbeit durchführen. Die Rede ist von Portainer, welches genau dies übernimmt. Portainer ist ein sehr mächtiges Tool und läuft als Docker-Container auf dem Raspberry Pi. Bevor der Container gestartet wird, musst du erst einmal ein Docker-Volumen erstellen. Dies machst du mit dem Befehl aus Code 6.

```
docker volume create portainer_data
```

Code 6: Dockervolumen für Portainer erstellen

Dieses Volumen brauchen wir, da ein Docker-Container beim Neustart seine Daten verliert. Um dies zu umgehen, soll der Docker-Container von Portainer ein Volumen einbinden, welches die angepassten oder erstellten Daten speichert. Im nächsten Schritt soll Portainer gestartet werden, was mit dem Befehl aus Code 7 geschieht.

```
docker run -d -p 8000:8000 -p 9443:9443 --name portainer --restart=always -v
/var/run/docker.sock:/var/run/docker.sock -v portainer_data:/data portainer/portainer-
ce:latest
```

Code 7: Portainer starten

Den oben genannten Befehl, auch wenn du Ihn ggf. schon ins Terminal eingegeben hast, möchte ich noch erklären. Zunächst das Attribut **-d** welches den Docker-Container im Hintergrund ausführt. Mit **-p** verknüpfen wir die Ports vom Container mit dem Port des Hostsystems, also dem Raspberry Pi. In unserem Kommando die Ports 8000 und 9443 mit demselben Port des Raspberry Pis. Letzteres ist die https-Verbindung im Webbrowser. Zuletzt wird mit **-v** die im Container vorhandenen Pfade mit dem Volumen und zuvor noch der Socket von Docker mit Portainer verknüpft. In Fall von Portainer speichern wir die Daten für Portainer im eben angelegten Volumen. Mittels –restart=always wird der Docker-Container immer wieder neugestartet, egal ob durch einen Fehler oder einen Neustart des Raspberry Pi.

Damit Portainer überhaupt laufen kann, muss zunächst das Image von Portainer in der Community Edition heruntergeladen werden, siehe Abbildung 5. Hier brauchst du nichts zu machen, durch unseren Befehl, wird das von selbst durchgeführt.

Abbildung 5: Docker lädt Portainer herunter und führt ihn aus

Portainer wird in der Regel von docker hub heruntergeladen und auf dem Raspberry Pi gespeichert. In unserem Fall, da eine Webadresse angegeben wurde, wird Portainer von portainer.io heruntergeladen. Durch das Tag **latest**, welches mit **:latest** am Ende des Image-Parameters eingetragen wurde, wird die aktuellste Version genutzt. In der Konsole siehst du nach dem Herunterladen und Ausführen erste einmal nichts, aber der Container von Portainer sollte im Hintergrund laufen.

Ob Portainer läuft, können wir mit zwei verschiedenen Methoden herausfinden. Die erste Variante ist über den Befehl aus Code 8.

```
docker ps
```

Code 8: Liste alle aktiven Docker-Container

Dieser Befehl listet alle aktiven Container von Docker auf, siehe Abbildung 6. Jeder Container erhält dabei eine zufällige Container ID und wir sehen diverse Informationen zu dem Container.

Abbildung 6: Zeige aktive Docker-Container

Was direkt ins Auge fallen könnte ist, dass die Ports 8000 und 9443 für IPv4 und IPv6 freigegeben und an den Docker-Container von Portainer weitergeleitet wird.

Die Portainer-Oberfläche für Docker auf dem Raspberry Pi

Die zweite Variante zu prüfen, ob Portainer gestartet wurde, ist einen Webbrowser mit der Adresse vom Pi und dem Port 9443 zu öffnen, siehe Abbildung 7.

Abbildung 7: Portainer beim ersten Start

Denke bitte daran, dass die Adresse wie folgt aussieht **https://**IP-vom-PI**:9443**. Da Portainer das erste Mal gestartet wird, musst du zu Beginn noch ein sicheres Passwort für den Administrator **admin** eintragen. Dieses musst du dir entweder aufschreiben oder merken, weil zurücksetzen kannst du es nicht mehr. Danach landest du in der Home-Ansicht von Portainer, siehe Abbildung 8.

Abbildung 8: Home-Ansicht von Portainer

Auf der rechten Seite, hier markiert mit der rot umrandeten 1, kommst du in die einzelnen Untermenüs, welche ich teilweise gleich noch näher erläutern werde. In der Mitte, rot umrandete 2 oder dem Dashboard, siehe Abbildung 9, siehst du den Status von den Docker-Containern und weitere Informationen wie Images, Volumes oder Networks. Stacks lasse ich an dieser Stelle einmal aus, das wird in einem späteren Beitrag relevant werden.

Abbildung 9: Dashboard von Portainer

Wechseln wir zunächst einmal in den Reiter Containers, mit dem entsprechenden Menüeintrag auf der rechten Seite, siehe Abbildung 10.

Abbildung 10: Portainer-Container Ansicht

Hier sehen wir erst einmal zwei Dinge. Zum einen den Container von dem „hello-world“-Beispiel und den **Portainer**-Container. Bei letzterem könntest du dich nun ein bisschen wundern, da wir ja gerade in Portainer arbeiten und es im ersten Moment keinen Sinn ergibt, dass dieser Container hier erscheint. Tatsächlich liest Portainer über den Docker-Socket alle Container aus und zeigt diese an. Da Portainer auch als Docker-Container läuft, wird daher auch Portainer als solcher angezeigt. Im nächsten Schritt soll nun ein neuer Container erzeugt werden, mit dem du dann auch arbeiten kannst.

Das Applikations-Dashboard Heimdall

Bei dem Namen Heimdall könnten einige nun kurz an die Marvel-Filme von Thor denken. In der nordischen Mythologie ist Heimdall der Götterwächter und bewacht die Brücke, bzw. den Bifröst, zwischen den Welten. Ähnlich kannst du dir auch das Applikations-Dashboard Heimdall vorstellen. Jeder kann sich ein eigenes Profil erstellen und für alle Web-Applikationen einen Button erstellen, der dich dann dorthin bringt. Teilweise haben diese Buttons sogar interessante Zusatzfunktionen. Gerade Heimdall ist ein super Einstieg in Portainer und die diversen Menüs.

Das Docker-Image von Heimdall herunterladen

In den Basics von Docker habe ich geschrieben, dass jeder Container ein Image braucht. Heimdall ist da keine Ausnahme und so muss dieses Image erst einmal gesucht und runtergeladen werden.

Erste Anlaufstelle für Images ist docker hub. Suche hier nach Heimdall, siehe Abbildung 11, wobei du eine Menge an Ergebnissen erhalten wirst.

Wichtig ist, dass du das Tag **arm** siehst, siehe rote Umrandung von Abbildung 11, da sonst das Image auf dem Pi nicht funktionieren würde. Gerade die Community um **linuxserver.io** stellen viele Images für den Raspberry Pi bereit, weswegen ich zum Eintrag **linuxserver/heimdall nur** raten kann. Gleichzeitig sind die Dokumentation und die GitHub-Pflege der einzelnen Images sagenhaft, was dir gerade am Anfang zu einem leichten Einstieg verhilft.

Wenn du nun auf der Seite von linuxserver/heimdall bist, siehst du auf der rechten Seite direkt den Docker-pull-Befehl, siehe Abbildung 12.

Abbildung 12: linuxserver/heimdall pull-Befehl

Weiter unten auf der Seite findet du einen Bereich **Usage** der dir erklärt, wie du mittels Docker-Compose, eine Art Skriptsprache, den Container zum Laufen bekommt. Es werden dabei alle wichtigen Einstellungen für Enviorments, Volume und Ports gezeigt, was wir gleich brauchen werden, siehe Abbildung 13.

Abbildung 13: Wichtige Heimdall-Einstellungen

Zunächst wollen wir aber das Image runterladen. Dazu wechselst du in Portainer ins Menü Images und suchst dort nach dem Image **linuxserver/heimdall**, siehe Abbildung 14.

Abbildung 14: Heimdall-Image runterladen

Mit einem Druck auf den Button **Pull the image**, wird das Image dann bei docker hub gesucht und die Version mit dem Tag **latest** runtergeladen. Theoretisch kannst du, wenn du bei docker hub dir die Tagseite genauer ansiehst mittels :**TAGNAME** aussuchen, welche Version du haben willst. Je nach Größe vom Image und deiner Internetleitung, kann der Download kurz dauern. Danach siehst du aber in der unteren Liste einen weiteren Eintrag mit einem Flag Unused, was das Heimdall-Image ist, siehe Abbildung 15.

Abbildung 15: Portainer zeigt runtergeladenes Heimdall-Image

Aus dem Heimdall-Image wird ein Docker-Container

Der nächste Schritt ist nun, aus dem Image ein Docker-Container zu erzeugen. Dazu wechselst du in den Menüpunkt **Containers** von Portainer und wählst den Button **Add container** aus, siehe Abbildung 16.

Abbildung 16: Neuen Docker-Container für Heimdall erstellen

Jetzt kommt der Moment, wo die Usage-Dokumentation der docker hub – Seite von Heimdall benötigt wird, daher solltest du diese parallel geöffnet haben. Zunächst braucht unser neuer Docker-Container einen Namen, hier Heimdall, sowie die Zuweisung, welches Image genutzt werden soll, hier **linuxserver/heimdall**, siehe Abbildung 17.

Abbildung 17: Container Name und Image übergeben

Als nächstes müssen die Netzwerkports vom Container mit dem Hostsystem, also dem Raspberry Pi, gemappt werden. Im Fall von Heimdall ist das der Port 80 und 443. In der Kartei **Networt ports configuration** drücken wir zweimal den Button publish a new network port und tragen die Ports ein, siehe Abbildung 18.

Abbildung 18: Die Heimdall-Ports mit dem Host verknüpfen

Da ich persönlich beim Öffnen der Website meines Docker-Pis die Heimdall-Seite gezeigt bekommen möchte, sind der Host- und Container-Port identisch. Achte dabei darauf, dass es sich um einen TCP-Port handelt und nicht UDP! Willst du einen anderen Port vom Host nutzen, so musst du später die Webadresse nach dem Schema **http(s)://**IP-vom-PI**:Eingestellter-Port** aufrufen.

Als nächstes musst du die Environments, das Volume und die Restart policy korrekt eintragen. Diese Einstellungen findest du am Ende der Seite. Bevor du damit aber beginnst, solltest du im Tab **Command & logging** bei Console den Punkt Interactive & TTY (-I -t) auswählen, siehe Abbildung 19.

Abbildung 19: Console vom Heimdall-Container aktivieren

Damit haben wir später Zugriff über die Konsole auf den Container und müssen nicht erst umständlich über docker-Kommandos versuchen Zugang zu bekommen.

Nun Wechsel in den Tab **Volumes** und richte mit dem Button **map additional volume** einen neuen Verweis für einen Container-Pfad zum Hostsystem ein, siehe Abbildung 20.

Abbildung 20: Docker-Container-Pfad mit Hostsystem verknüpfen

Hier wird der Ordner **/config** aus dem Image mit dem Ordner **/home/pi/heimdall/config** gemappt. Das kannst du dir wie eine Art Verknüpfung auf einem PC vorstellen, wobei hier der Container-Ordner quasi ausgetauscht wird. Das geht in diesem Fall aber nur, da ich die Option **Bind** anstelle von **Volume** ausgewählt habe. Der Vorteil bei einem Bind ist, dass ich auf die Daten zugreifen kann, was bei einem Container-Volume schwieriger ist. In dem oben gezeigten Beispiel

wird im Arbeitsverzeichnis vom User pi, also /home/pi, der Ordner /heimdall/config erzeugt und in den Container gemountet.

Nun müssen noch die Environments im Tab **Env** eingetragen werden. Drücke dazu dreimal den Button **Add an enviroment variable** und trage die Werte für PUID, PGID und TZ ein, siehe Abbildung 21. Wenn du ein Standard Raspberry Pi OS hast, hat der Benutzer pi jeweils die 1000 für PUID und PGID.

Abbildung 21: Enviorments für Container eintragen

Die entsprechenden Einstellungen kannst du aus Abbildung 21 oder dir die Environments auf der docker hub – Seite von linuxserver/heimdall unter **Usage** ansehen. Da ich mich in der Zeitzone von Berlin befinde, trage ich hier für das Environment **TZ** entsprechend **Europe/Berlin** ein.

Bevor der docker-Container nun deployed werden kann, solltest du noch im Tab **Restart policy** die **Restart** policy von Never auf **Always** umstellen, siehe Abbildung 22.

Abbildung 22: Restart policy auf Always ändern

Im Anschluss kannst du auf den Button **Deploy the container** drücken. Wenn alles richtig eingestellt ist, wirst du beim erfolgreichen deploy, direkt auf die Container-Seite von Portainer weitergeleitet und siehst einen gestarteten Heimdall-Container, siehe Abbildung 23.

Abbildung 23: Container-Seite von Portainer mit Status von Heimdall

An der Stelle musst du ggf. noch ein bisschen warte, da beim ersten Starten vom Heimdall-Container noch einige Keys und Konfigurationen im **/config**-Ordner erzeugt werden müssen. Bei einem Pi 4 mit 4GB dauert der erste Start ungefähr 2 Minuten, danach kannst du aber die Heimdall-Seite erreichen, wobei du vorher das selbsterzeugte SSL-Zertifikat akzeptieren musst, siehe Abbildung 24. Wird der Container neu gestartet, entfällt diese Wartezeit, da die Daten dann schon vorhanden sind.

Abbildung 24: Die Webapplikation Heimdall

Kurzeinführung in Heimdall

Die Webapplikation von Heimdall ist intuitiv und schnell gelernt. Über die Menüleiste am rechten unteren Rand, siehe Abbildung 25, kannst du Heimdall komplett nach deinem Wunsch konfigurieren.

Abbildung 25: Menüeinträge von Heimdall

Ich empfehle am Anfang unter dem Menü Einstellungen die Sprache auf Deutsch zu schalten, dann wird es ggf. ein bisschen leichter bei der Erstellung der Web-Shortcuts. Wie du eine Applikationsliste erzeugst, will ich dir in einem einfachen Beispiel zeigen.

Wechsle zunächst in die Applikationsliste und führe ein Update der App Liste durch, siehe Abbildung 26.

Abbildung 26: Appliste updaten

Über den Button Hinzufügen erstellst du ein neues Anwendungselement, siehe Abbildung 27.

Abbildung 27: Hinzufügen einer neuen Anwendung

In meinem Beispiel erstelle ich nun einen Shortcut zu meiner FritzBox, wozu es auch ein **Application Type** gibt, siehe Abbildung 28. Gerade durch das Applications Type **AVMFritzBox** werden schon einige Bilder und Buttonanpassungen vorgenommen.

Abbildung 28: Einstellungen für FritzBox

Lediglich der Name und die URL muss noch nachgetragen werden. Theoretisch kannst du über den Button **Hochladen einer Datei** auch ein anderes Icon auswählen, sofern es dir nicht gefällt oder du die Applikation nicht in der Liste findest. Nach dem Speichern erscheint der Button in deiner Übersicht, siehe Abbildung 29.

Abbildung 29: Neue Webapplikation in der Übersicht

Je nach Geschmack und Arbeit, kann die Übersicht schon einige Shortcuts zu wichtigen Webapplikationen zeigen, wie in meiner Übersicht, siehe Abbildung 30.

Abbildung 30: Meine Heimdall-Übersicht

Zusammenfassung

Ich habe dir bisher Docker und Portainer versucht näher zu bringen. Ich hoffe die Erklärung zwischen einem Docker-Container zu einer VM ist mir insoweit gelungen, dass dieser Unterschied nun klar ist. Mit deinem ersten Docker-Container Heimdall habe ich dir zudem gezeigt, wie einfach es ist, ein Docker-Image runterzuladen und als Container dann zu starten. Auf viele Eigenheiten von Docker bin ich nicht eingegangen, das hätte hier aber den Rahmen um weiten gesprengt. Nicht zuletzt gibt es von Michael Kofler und Bernd Öggl ein komplettes Buch zu dem Thema Docker, was die Komplexität verdeutlichen soll. Wahrscheinlich hat dich dieser doch recht umfangreiche Beitrag etwas erschlagen, aber ab jetzt werden die Themen in dieser Serie einfacher bzw. kompakter. Gerade die Grundlagen von Docker sind wichtig, damit du später in deinem Test- bzw. Produktivumfeld sicher arbeiten kannst.

Dies ist Beitrag 2 von 2 der Serie *“Raspberry Pi Docker Basics”*

In dieser Serie lernst du den Umgang mit Docker Containern auf dem Raspberry Pi

Genug von Werbung und Trackern? In diesem umfassenden Guide erklären wir dir, wie du Pi-hole auf dem Raspberry Pi mittels Docker installieren und konfigurieren kannst.

Im ersten Teil dieser Serie habe ich dir Docker und Portainer nähergebracht und genau erläutert, worum es sich bei dieser Technik handelt. Gleichzeitig habe ich dir Heimdall als erstes laufendes Projekt vorgestellt. Nun will ich dir den Vorteil von Docker zeigen, indem wir einen zweiten Container mit einem Webserver laufen lassen werden. Anders als zuvor soll der Container nun in unserem Netzwerk etwas tun und uns beim Surfen durchs Internet von Werbung befreien. Die Rede ist von Pi-hole, was ich dir in den Grundzügen auch noch erklären werde. Keine Angst, auch dieses Mal werde ich dich durch die Konfiguration vom Container führen und danach durch die Einstellung von Pi-hole.

INHALTSVERZEICHNIS

Die Hardware für diesem Blog

Vom Prinzip kannst du jeden Raspberry Pi nehmen. Solltest du noch keinen Raspberry Pi besitzen, dann melde dich in unserem kostenlosen BerryBase Maker Club an, wo wir aktuell exklusiv für Mitglieder Boards zur Verfügung haben. Es sollte aber zumindest ein Raspberry Pi 3, besser noch ein 4er mit mindestens 4GB RAM sein.

Die Software für diesem Blog

Damit du diesen Abschnitt vom Blogbeitrag mit durcharbeiten kannst, brauchst du Docker und Portainer auf deinem Raspberry Pi. Wie das geht, erkläre ich im ersten Teil dieser Serie, wobei du die Installation von Heimdall nicht durchführen musst. Pi-hole funktioniert auch ohne Heimdall, was ja auch der Sinn von Docker ist.

Was ist bzw. wie funktioniert Pi-hole

Bevor ich dir erkläre, wie du Pi-hole auf deinem Raspberry Pi als Docker-Container einrichtest, sollte vielleicht vorab geklärt werden, was Pi-hole genau macht. Pi-hole ist ein sogenannter Tracking- und Werbeblocker und kann auch als DHCP-Server eingesetzt werden. Wie der Name schon verrät, ist Pi-hole für unseren Kleinstcomputer, dem Raspberry Pi, entwickelt worden. Vom Prinzip kann Pi-hole über die Konsole bedient werden, aber durch eine grafische Oberfläche, die über einen Webserver bereitgestellt wird, ist die Bedienung einfacher.

Aber wie genau filtert Pi-hole nun Werbung und unterbindet Tracking? Vom Prinzip steckt hinter Pi-hole eine Datenbank, die Abfragen von unserem Computer bzw. Geräte die Pi-hole als DNS-Server verwenden, abgleicht und im Zweifelsfall unterbindet. Ich will nicht zu tief ins Detail gehen, da ich hier keinen Exkurs über Netzwerk-Technik anfangen möchte, dennoch will ich versuchen es einfach zu erklären.

Wenn du mit deinem PC einen Webadresse in die Adresszeile eintippst und Enter drückst, passiert eine ganze Menge. Zunächst wird deine Anfrage über den Router an einen DNS-Server weitergegeben. Diese Abfrage muss gestellt werden, damit im späteren die IP-Adresse der angeforderten Webadresse unserem Computer bekannt ist. Den hinter jeder Webadresse steht im Grunde nur eine eindeutige IP-Adresse zu einem Server. Ähnlich wie bei einem Telefonbuch erhalten wir vom DNS-Server eine Adresse zurück und landen von dort auf unserer gewünschten Website.

Jetzt kommt Pi-hole ins Spiel. Wenn du Pi-hole als lokalen DNS-Server einbindest, werden deine Anfragen zwischen Router und deinem Computer „abgefangen“ und analysiert. D.h. die Anfragen über eine Website laufen zunächst über den Pi-hole und wenn eine geblockte Adresse dabei ist, wird die Anfrage zu der Seite direkt blockiert. Das ist aber erst der erste Teil, den Pi-hole durchführt. Wenn eine Website in deinem Browser geladen wird, dann wird auch meist Zusatzinhalt, wie Werbebanner oder ähnliches nachgeladen. Werbung oder Tracker sind, gerade in der heutigen Zeit, die Einnahmequelle, daher übertreiben es viele Webseiten mit zugeschalteter Werbung oder Trackern. Auch hier greift Pi-hole wieder ein. Wenn sich in dem nachgeladenen Inhalt eine Adresse befindet, die eine geblockte Werbung oder einen Tracker enthält, wird diese Anfrage ebenfalls geblockt. All das passiert über die schon erwähnte Datenbank im Hintergrund.

Aber wie wird diese Datenbank zur Filterung genau befüllt?

Hier bietet verschiedene Dienstleister oder Privatpersonen sogenannte AdLists an, die eine Vielzahl von ungewollten Webadressen enthalten. Eine solche Liste kannst du dir vom GitHub-Verzeichnis von StevenBlack mal ansehen. Die hier bereitgestellte Liste umfasst über 10.000 Einträge. Eine

solche AdList wird heruntergeladen und direkt danach in die Datenbank eingetragen. Je mehr Einträge also vorhanden sind, umso mehr Werbung oder Tracker werden geblockt.

So schön sich das im ersten Moment anhört, gibt es auch einen Nachteil. Durch die vielen geblockten Webadressen kann es auch passieren, dass harmlose Webseiten oder LogIn-Seiten geblockt werden. In meinem Fall hatte Pi-hole zur Folge, dass ich mich mit Autodesk Fusion 360 nicht mehr anmelden konnte oder Nvidia-Logins nicht mehr verfügbar waren. Das ist ärgerlich, aber hier gibt es mit sogenannten WhiteLists, also geduldeten Adressen, eine Möglichkeit diese wieder zu erlauben. Egal ob über eine feste Adresse oder eine sogenannte Wildcard, beides ist zur Freischaltung möglich. Eine Liste aller Anfragen und ob diese geblockt oder freigegeben sind, kann man in einem Log einsehen. Dazu werde ich dir aber im späteren Verlauf dieses Blogartikels mehr erklären.

Pi-hole als Docker-Container auf dem Raspberry Pi einrichten

Wie schon im ersten Blogbeitrag, wird über Portainer ein Docker-Container von Pi-hole erstellt. Das geschieht wieder in zwei Phasen. Als erstes wird das Image heruntergeladen und im Anschluss werden wir aus dem Docker-Image ein Docker-Container erstellen. Damit du das alles nachmachen kannst, musst du dich über die Weboberfläche von Portainer anmelden. Nutze dazu die Login-Daten, die du im ersten Teil der Blogserie extra für Portainer erzeugt hast.

Das Image von Pi-hole herunterladen

Zunächst wird das Docker-Image von Pi-hole benötigt. Dieses finden wir, wie auch schon im ersten Beitrag, bei docker hub. Wenn du in der Suche **pihole** einträgst, gelangst du schon im ersten Suchergebnis auf das offizielle Docker Image der Pi-hole-Macher, siehe Abbildung 1.

Abbildung 1: Sucheergebnis zu pihole bei docker hub

Da es sich, wie schon oben erwähnt, um das offizielle Image von pi-hole.net handelt, sollte auch genau das heruntergeladen werden. Dazu öffnest du die docker hub – Seite von pihole und kopierst dir aus dem **Docker Pull Command** den entsprechenden Pfad, siehe Abbildung 2.

Abbildung 2: Den Pfad für Pi-hole aus dem Docker Pull Command kopieren

Lass am besten die Seite gleich offen, wir werden Sie gleich noch brauchen, wenn es zu der Konfiguration des Pi-hole Docker Container auf dem Raspberry Pi kommt. Zunächst loggst du dich bei Portainer ein und Wechsel zu Menü **Images**, siehe Abbildung 3.

Abbildung 3: Image von Pi-hole in der neusten Version runterladen

Auch in diesem Fall wollen wir die neuste Version von Pi-hole haben, daher geben wir am Ende von **pihole/pihole** das Tag **:latest** an. Direkt danach laden wir das Image über den Button **Pull the image** herunter. Das Image ist knapp 250MB groß, daher kann es je nach Internetleitung einen Moment dauern, am Ende sollte aber das Image mit dem Flag **Unused** in der Imageliste erscheinen, siehe Abbildung 4.

Abbildung 4: Das Docker-Image von Pi-hole in der Image-Liste

Das Image ist damit schon einmal auf unserem Raspberry Pi verfügbar und es kann ein Docker-Container daraus erzeugt werden.

Den Pi-hole Docker Container auf dem Raspberry Pi erzeugen

Damit aus dem Docker-Image ein Docker-Container wird, musst du nun in den Menüpunkt **Containers** und drücken auf den Button **Add container**, siehe Abbildung 5.

Abbildung 5: Den Pi-hole-Container erstellen

Damit du später den Docker-Container auf dem Raspberry Pi wiederfindest, braucht dieser erst einmal einen eindeutigen Namen und das Image muss angegeben werden. Dies passiert, wie du schon sicher weißt, am Anfang der Konfiguration, siehe Abbildung 6.

Abbildung 6: Container einem Namen geben und Image auswählen

Bevor es nun weitergeht, sollten wir einen Blick in die Dokumentation werfen. Hier erhalten wird interessante Informationen über den Container, wie z.B. die Ports oder die Environments. Ersteres ist dabei für unseren Tracker- und Werbungsfilter vom besonderen Interesse, da wir einen alternativen DNS-Server dafür verwenden, siehe Abbildung 7.

Abbildung 7: Information zum Pi-hole-Container

Der Grund dafür ist, dass die Ports für DNS und dem Bootstrap Protocol standardisiert sind und auch welche Art von Protokoll. In den meisten Fällen kommt das sogenannte Transmission Control Protocol, kurz TCP, zum Einsatz. In manchen Fällen aber auch das User Datagram Protocol, kurz UDP. Die Unterschiede möchte ich hier nicht erläutern, wichtig ist an dieser Stelle erst einmal, dass die Ports für unseren alternativen DNS-Server korrekt gesetzt werden, dies sind die Ports 53 und 67. Beim Port 80 handelt es sich um den Standardport vom Webserver, bei Pi-hole kommt der lighttp zum Einsatz, der die grafische Oberfläche von Pi-hole bereitstellt. Also erzeugst du erst einmal neue Netzwerkports und stellst diese korrekt ein, siehe Abbildung 8.

Abbildung 8: Netzwerkports für Pi-hole

Achte darauf, dass du TCP und UDP korrekt hinter den Ports einstellst, da andernfalls Pi-hole nicht funktionieren könnte. Solltest du Heimdalll aus dem ersten Teil verwenden, so musst du dir einen anderen Port für den Webserver aussuchen. Typisch ist, dass man Port 8080 als http-Alternative verwendet. Es geht in der Theorie aber jeder andere Port, der nicht schon vom Host belegt ist.

Im nächsten Schritt stellst du zunächst im Tab **Command and logging** bei Console den Wert auf **Interactive && TTY**.

Abbildung 9: Interactive Konsole & TTY nutzen

Warum, werde ich dir im Bezug auf das Updaten der Filterdatenbank näher erklären. Daher an dieser Stelle erst einmal nur setzen. Nun soll dem Container ein Volumen bzw. ein Bind zugeordnet werden, auch hier hilft die Doku der Docker Hub – Seite. In meinem Fall nutze ich die Bind-Methode, siehe Abbildung 10, damit man später relativ einfach auf die Daten vom Container zugreifen kann.

Abbildung 10: Bind für den Pi-hole-Container

Mit den **Environments** legst du zum einen die Zeitzone fest und kannst auch ein von dir gewünschtes Webadmin-Passwort definieren, siehe Abbildung 11. Beides musst du im Tab **ENV** vornehmen.

Abbildung 11: Einstellung der Enviroment

Die Zeitzone, als Name **TZ**, braucht als Variablenwert **Europe/Berlin**. Vergibst du der Variable **WEBPASSWORD** kein Value oder legst diese nicht an, wird ein zufälliges Passwort generiert. Im Tab **Restart policy** setzt du die Neustart-Regelung auf **Always**, siehe Abbildung 12.

Abbildung 12: Die Neustart-Regelung setzen

Bevor du nun den Pi-hole Docker Container auf dem Raspberry Pi ausführst, muss das Kernel-Modul **NET_ADMIN** unter dem Tab **Capabilities** aktiviert werden, siehe Abbildung 13**.**

Abbildung 13: Das Modul NET_ADMIN aktivieren und Container starten

Hast du bis hier hin alles genau befolgt, sollte der Container von Pi-hole nun ohne Fehlermeldung gestartet worden sein und nach einer kurzen Zeit in deiner Containers-Liste mit dem Status **running** oder sogar schon **healthy** zu finden sein, siehe Abbildung 14.

Abbildung 14: Pi-hole-Container gestartet

Abbildung 14: Pi-hole-Container gestartet

Healthy ist ein Status bei Docker, was durch ein internes Skript geprüft wird. Genauere Infos dazu findest du bei Docker direkt in der Manual-Seite.

Pi-hole administrieren und benutzen

Der Docker-Container ist gestartet und Pi-hole ist nun einsatzbereit. Jetzt muss Pi-hole noch angepasst werden und am Ende sollte der Pi-hole auch zwischen PC und Router als alternativer DNS-Server eingebunden werden.

Gestatten Pi-hole

Eine komplette Übersicht von Pi-hole kann ich in diesem Blog nicht geben, jedoch ist die Hilfe-Seite von Pi-hole dir einmal näher ansehen. Ich zeige dir hier, wie du weitere AdLists einträgst, Whitelists erzeugst und was es noch für interessante Einstellungen gibt. Zunächst musst du aber erst einmal die Weboberfläche von Pi-hole über deinen Browser aufrufen. Wenn du bis hierhin den Blog genau befolgt hast, ist die Adresse ***IP-Deines-Pi*:8080/webadmin.** Gibst du nur die IP mit dem Port an, landest du auf einer Seite, die dich automatisch zur Webadmin-Seite von Pi-hole weiterleitet, siehe Abbildung 15.

Abbildung 15: Weiterleitung zu Pi-hole Webadmin-Oberfläche

Dort angekommen, siehst du schon im nicht eingeloggten Zustand wichtige Informationen. Das Dashboard ist für die Statistik der perfekte Ort, siehe Abbildung 16.

Abbildung 16: Dashboard von Pi-hole

Neue Blacklists in Pi-hole auf dem Raspberry Pi hinzufügen

Aktuell wirst du nicht viel sehen, da der Pi-hole nicht zum Filtern verwendet wird. Interessant ist aber, dass Pi-hole schon über 100.000 Domains in der Blocklist hat. Das wollen wir nun etwas erweitern, wofür du über die Login-Seite dich mit deinem Passwort einloggst. Wechsel danach in der linken Menüleiste zu **Group Management -> AddLists**, siehe Abbildung 17.

Abbildung 17: Zu AdLists wechseln

Jetzt trägst du in dem Feld **Address** eine gültige Adresse ein. In unserem Fall ist das zu Demozwecken die Adresse https://easylist.to/easylist/easylist.txt und bestätigen mit **Add**, siehe Abbildung 18.

Abbildung 18: Eine Adresse hinzufügen

Damit ist zwar die Liste nun eingetragen, die Informationen in der Liste sind aber noch nicht in unserer Datenbank hinterlegt. Daher folgt nun der Schritt, dass Pi-hole sich aus den angegebenen Adressen die Informationen zieht. Dabei bietet Pi-hole zwei Varianten an:

1. Über ein Untermenü in der Weboberfläche
2. Mittels einem Terminal-Command

An dieser Stelle, auch weil ich in der Vergangenheit ein paar Mal Probleme hatte, nutze ich lieber das Terminal-Command. Der zweite Grund ist, dass du nun auch erfährst, warum in den Container-Einstellungen in Portainer immer bei den **Command & logging** -Einstellungen, die **Console** auf **Interactive & TTY** steht, siehe dazu Abbildung 9. Du wechselst nun wieder zu Portainer in den Reiter Containers und wählst das Symbol **Exec Console,** Abbildung 19**.**

Abbildung 19: Die Konsole des Pi-hole-Containers via Portainer öffnen

Im darauffolgenden Menü kannst du alle Einstellungen so belassen und direkt auf Connect drücken, siehe Abbildung 20.

Abbildung 20: Mit Pi-hole-Konsole verbinden

Jetzt bist du in der Konsole vom Pi-hole-Container mit dem Account root, siehe Abbildung 21.

Abbildung 21: Als Benutzer root auf der Pi-hole-Konsle angemeldet

Nun soll der Befehl für das Herunterladen und Aktualisieren der Blacklists erfolgen. Dazu nutzt du das Command aus Code 1.

```
pihole -g
```

Code 1: Pi-hole Blacklist aktualisieren

Je nach Anzahl von Adressen, kann der Vorgang ein bisschen dauern, da hier aber nur eine weitere Liste hinzugekommen ist, wirst du schnell eine Erfolgsmeldung sehen, siehe Abbildung 22.

Abbildung 22: Blacklists aktualisiert

Eine Liste macht noch nicht viel aus, daher soll nun noch ein bisschen erweitert werden. Hilfe zu nützlichen URLs mit Blacklists finden wir bei firebog.net. Keine Angst, du musst nun nicht jede einzelne Adresse eingeben, hier hat Pi-hole eine nette Funktion eingebaut. Mehrere URLs können Leerzeichengetrennt in die Address-Zeile eingetragen werden. Du kopierst also von firebog.net ein paar URLs und fügst diese dann in die Adresszeile ein, siehe Abbildung 23.

Wieder zu Portainer in der Konsole, führst du das Command aus Code 1 aus. Das wird diesmal ein bisschen länger dauern, da knapp 60.000 neue Einträge hinzukommen. So kannst du bequem die Liste erweitern. Je nachdem wie viele URLs du einträgst, können bis zu 1 Millionen Einträge hinzukommen.

Den Pi-hole nun auf einem Arbeitsrechner einbinden

Aktuell wird der Pi-hole unerwünschten Traffic noch nicht blocken, da Pi-hole in Netzwerk noch gar nicht verwendet wird. Das Stichwort für euer Betriebssystem heißt **hier Bevorzugter DNS-Server** , der je nach Betriebssystem an einer anderen Stelle eingetragen werden muss. Im Fall von Windows geht man in die Netzwerkeinstellungen und wählt den passenden Netzwerkadapter aus. Mit einem Doppelklick auf Internetprotokoll, Version 4 (TCP/IPv4) kannst du im nachfolgenden die IP-Adresse von deinem Pi-hole eintragen, siehe Abbildung 24.

Abbildung 24: Bevorzugte IP für DNS-Server eintragen

Es gibt aber auch die Möglichkeit, direkt Pi-hole auf dem Router laufen zu lassen. Diese Art möchte ich dir aber hier nicht vorstellen, kann aber einfach im Internet nachgelesen werden. Auch behandle ich nur den alten Standard TCP/IPv4 und nicht das immer mehr kommende TCP/IPv6.

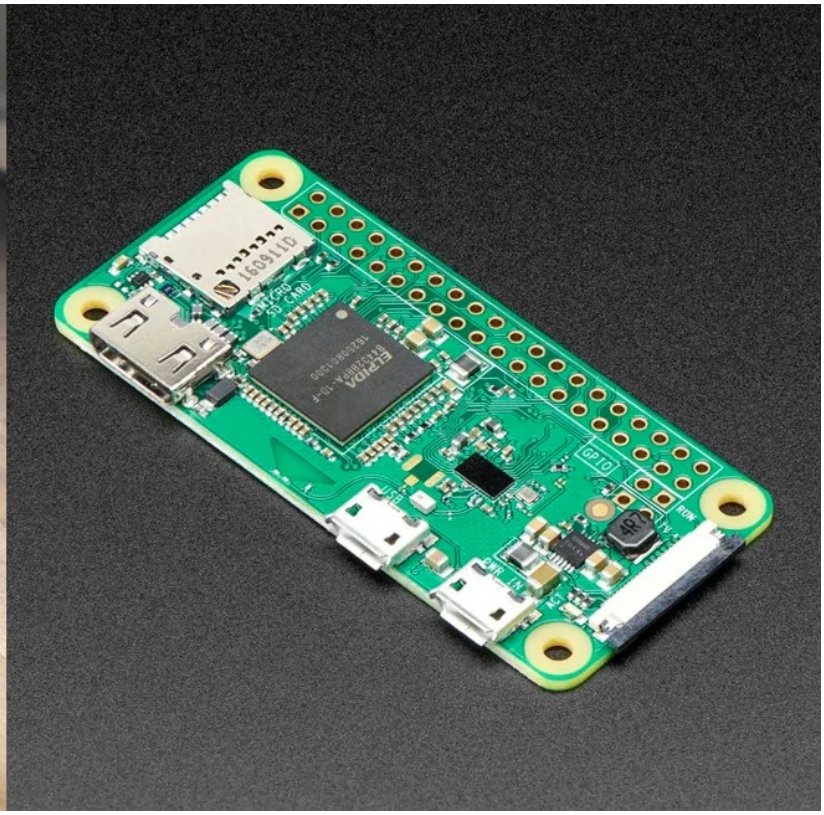
Zusammenfassung

Ich habe dir mit diesem Blogbeitrag den DNS-Filter Pi-hole vorgestellt und wie dieser als Docker-Container mit Portainer eingerichtet wird. Auch habe ich dir gezeigt, wie du mittels Portainer auf die Konsole deiner Container zugreifen kannst. Pi-hole ist aber so viel mehr, als das was ich in diesem Beitrag zeigen konnte! Mit Whitelists, Wildcards und diversen Einstellungen, kann Pi-hole noch deutlich mehr! Wenn dir also Pi-hole gefällt oder du dein Wissen vertiefen willst, dann kann ich dir das Forum von Pi-hole nur empfehlen. Hier findest du eine große Wissensdatenbank, viele helfende Leute und auch Anleitungen und Ankündigungen rund um den Pi-hole.

Gerade im Bezug auf das Thema Sicherheit, kann ich die Integration von Pi-hole ins Netzwerk nur empfehlen.

Setup a Raspberry Pi Zero W to run a Web Browser in Kiosk Mode

with Dakboard



Download Raspbian Buster Lite and write it to the SD card

https://www.raspberrypi.org/downloads/raspbian/

First login

`u:pi`

`p:raspberry`

Configuration

`sudo raspi-config`

1. change pi password
2. change hostname
3. set wifi ssid and pass
4. set timezone
5. inferface opt -> SSH enable
6. Boot Options -> Desktop / CLI -> Console Autologin
7. finish -> reboot

Applying latest versions

`sudo apt-get update && sudo apt-get upgrade -y`

Creating a Minimum X Server Environment for the Chromium Browser

`sudo apt-get install --no-install-recommends xserver-xorg x11-xserver-utils xinit openbox -y`

`sudo apt-get install --no-install-recommends chromium-browser -y`

Remove Rainbow Screen (OPTIONAL)

`sudo nano /boot/config.txt`

```
# Disable rainbow image at boot


disable_splash=1
```

Create an free account on Dakboard and get your custom board url

https://www.dakboard.com/

Configure Openbox.

`sudo nano /etc/xdg/openbox/autostart`

```
# Disable any form of screen saver / screen blanking / power management

xset s off

xset s noblank

xset -dpms
```

```
# Allow quitting the X server with CTRL-ATL-Backspace

setxkbmap -option terminate:ctrl_alt_bksp
```

```
# Start Chromium in kiosk mode

sed -i 's/"exited_cleanly":false/"exited_cleanly":true/' ~/.config/chromium/'Local State'

sed -i 's/"exited_cleanly":false/"exited_cleanly":true/;
s/"exit_type":"[^"]\+"/"exit_type":"Normal"/' ~/.config/chromium/Default/Preferences

chromium-browser --disable-infobars --noerrdialogs --incognito --check-for-update-interval=1
--simulate-critical-update --kiosk '[https://DAKBOARD-CUSTOM-URL-HERE]'
```

Start X automatically on boot

`sudo nano .profile`

```
[[ -z $DISPLAY && $XDG_VTNR -eq 1 ]] && startx -- -nocursor
```

Vertical screen (OPTIONAL)

`sudo nano /boot/config.txt`

`display_rotate=1`

Refreshes

`sudo apt-get install xdotool -y`

`sudo nano keyF5`

```
export display=:0,0

xdotool keydown F5; xdotool keyup F5 &

exit
```

`sudo chmod +x keyF5`

`sudo chown pi:pi keyF5`

crontab -e

```
#Refesh every 30 min

0 */30 * * * /home/pi/keyF5

#Shutdown at 11PM  (OPTIONAL)

* 23 * * * sudo shutdown -h
```

Zerocam

Scripte:

#/bin/bash

raspistill -t 86000000 -tl 10000 --rotation 90 -q 80 -o /home/pi/Photo/image%04d.jpg

---

#/bin/bash

raspistill -t 0 -tl 10000 -w 1920 -h 1080 --annotate 12 --rotation 90 -o /home/pi/Photo/image%04d.jpg

---

rc.local:

```
#!/bin/sh  -e
#
# rc.local
#
# This script is executed at the end of each multiuser runlevel.
# Make sure that the script will "exit 0" on success or any other
# value on error.
#
# In order to enable or disable this script just change the execution
# bits.
#
# By default this script does nothing.
#aktivieren...
#/home/pi/10sec-still_FullHD.sh
#/home/pi/camstream.sh
# Print the IP address
_IP=$(hostname -I) || true
if [ "$_IP" ]; then
  printf "My IP address is %s\n" "$_IP"
fi

exit 0
```

Streaming: http://zerocam2:8080/?action=stream

/bin/bash

v4l2-ctl -c compression_quality=85
v4l2-ctl -c brightness=55
v4l2-ctl -c rotate=90
v4l2-ctl -c contrast=30
v4l2-ctl -c saturation=20
v4l2-ctl -c scharpness=60
mjpg_streamer -i "input_uvc.so -d /dev/video0 -r 1280x720 -f 25" -o "output_http.so -w /usr/local/share/mjpg-streamer/www -p 8080"

Anleitung:

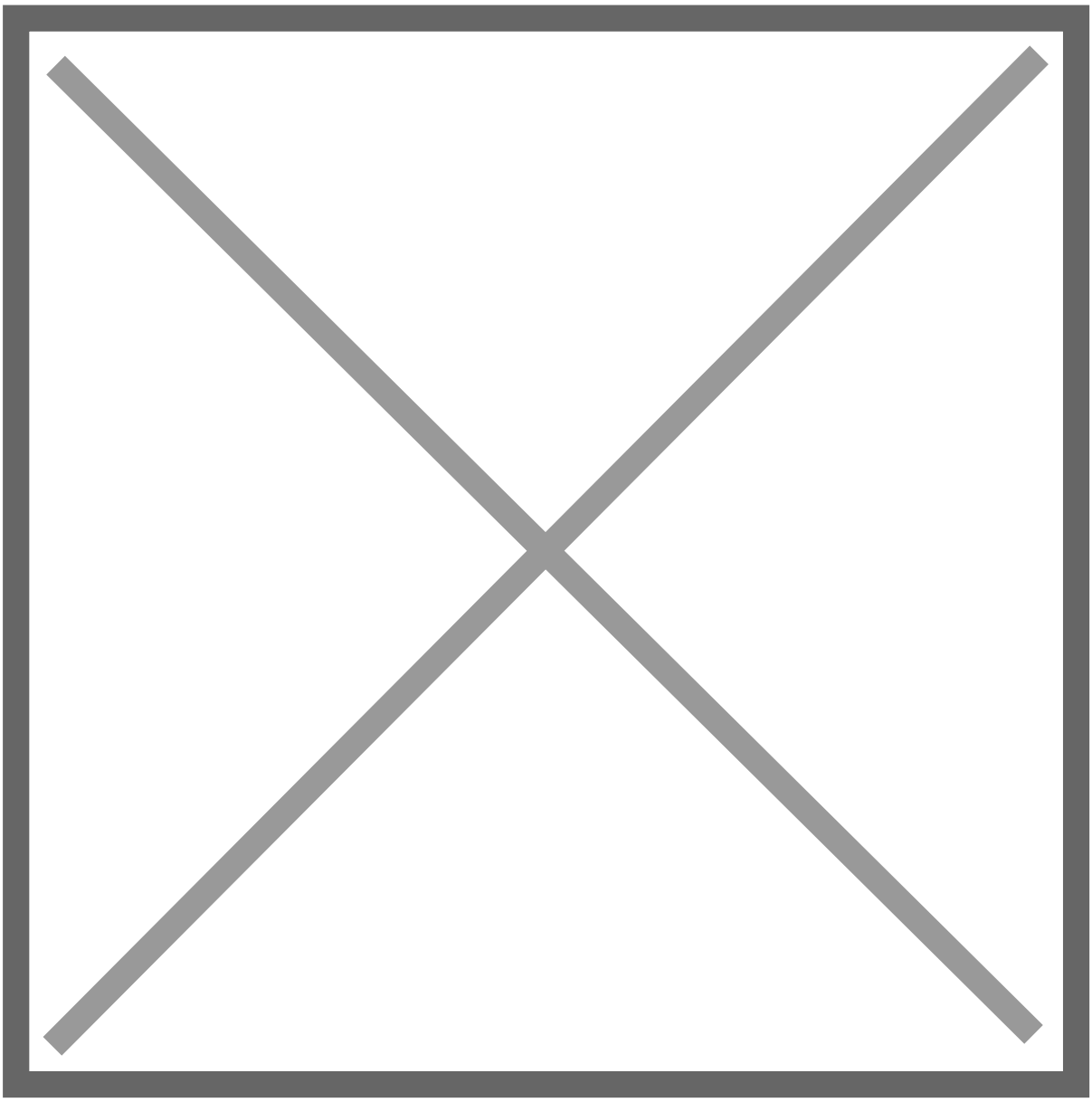
Let’s test by trying to run mjpg-streamer against the input_uvc.so plugin, since I”m not using a Raspi-Camera, but a USB one instead. (**See notes at the bottom about Raspberry Pi Bullseye and Raspi-Cameras**). I’ll also output using the http plugin. So my testing command line looks like this:

```
/usr/local/bin/mjpg_streamer -i "input_uvc.so -f 15 -r 640x480" \
 -o "output_http.so -w /usr/local/share/mjpg-streamer/www"
```

Executing that line gives us a dump of data – here’s the relevant info:

```
MJPG Streamer Version: git rev: 5a6e0a2db163e6ae9461552b59079870d0959340
 i: Using V4L2 device.: /dev/video0
 i: Desired Resolution: 640 x 480
 i: Frames Per Second.: 15
 i: Format............: JPEG
 i: TV-Norm...........: DEFAULT
 o: www-folder-path......: /usr/local/share/mjpg-streamer/www/
 o: HTTP TCP port........: 8080
 o: HTTP Listen Address..: (null)
 o: username:password....: disabled
 o: commands.............: enabled
```

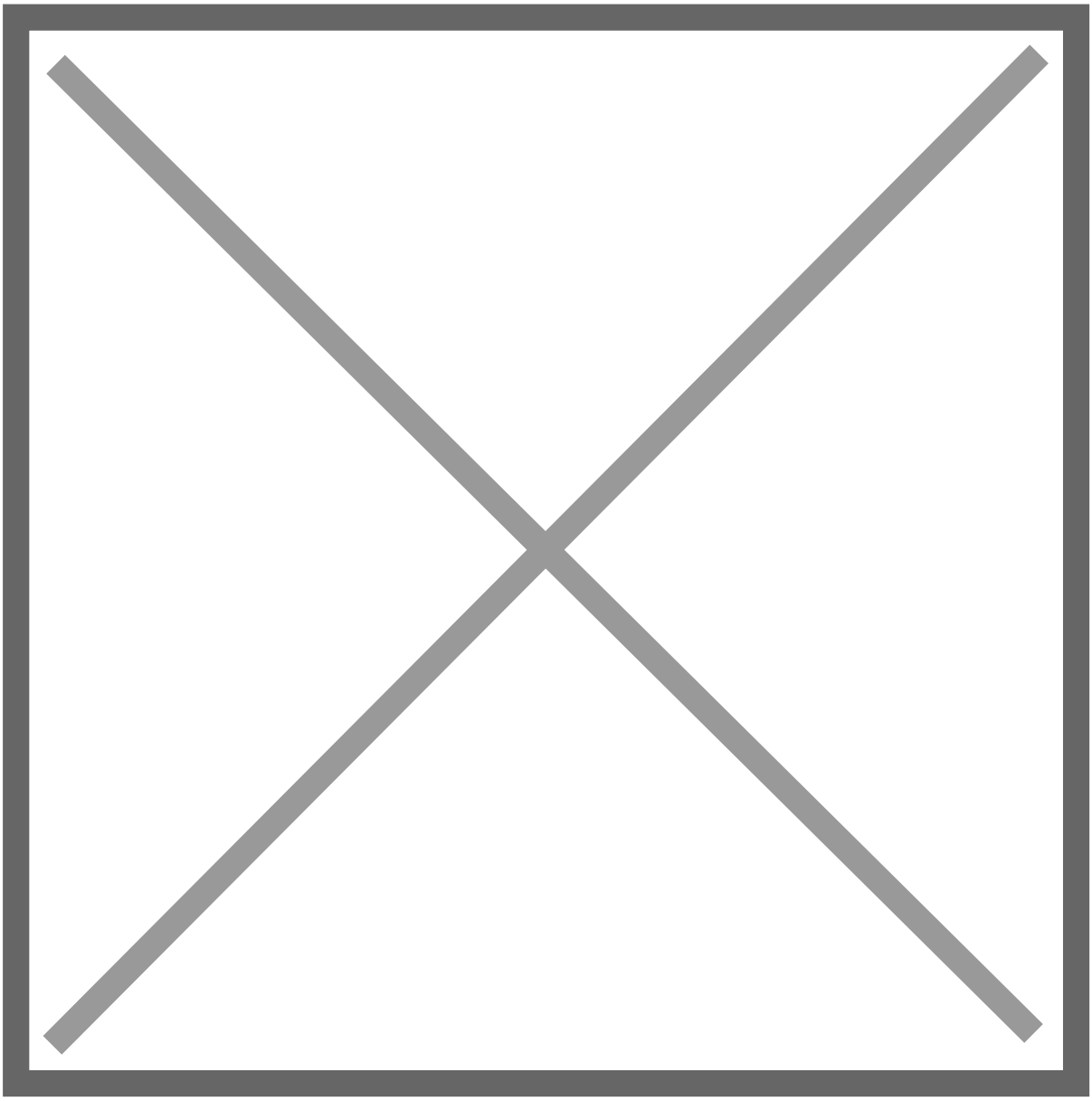
Our key variables to tweak are **-f for frame rate**, and **-r for resolution**. I’ll change those later, but for a quick test I head on out to my browser and point it at the www server at http://cam-pi-zero.local:8080/



I always feel like… somebody’s watching me…

Not only do we see the web interface, but a snapshot of the web cam, which was, clearly, laying down on a workbench behind me while I tried this out.

Validating the stream worked (by clicking ‘Stream’) and turning around, I could see myself moving:



Hello you... come here often?

You can now hit CTRL-C in your SSH window (or terminal) and quit the stream. After this point, I could delete the build files I downloaded.

Tweaking configuration

I wanted to run my camera at its intended resolution and at least a frame rate of 30fps. To do that I modified my command line:

```
#30fps, 1080 HD
/usr/local/bin/mjpg_streamer -i "input_uvc.so -f 30 -r 1920x1080" \
 -o "output_http.so -w /usr/local/share/mjpg-streamer/www"
```

I then loaded my own stream up in VLC by opening the direct network path to the stream:

http://cam-pi-zero.local:8080?action=stream

That worked, and I get about a 1-2 second delay from Pi to VLC. So I wouldn't use this with audio feeds unless I was planning on creating a sync delay. I'm using this to monitor 3D prints, or watch birds outside, so my use case does **not** demand low latency. Is it exactly 30 frames per second? No…. NO it's not.

At 1280×720, it seemed closer, but at 1920×1080, when set at 30fps, it definitely wasn't even close. I'd guess more like 15.

The PI wasn't maxed out on CPU, but it could just be the nature of USB 2.0 at this point. I'm not sure how I can tell if it's overloaded there or not, but, once again, this is a light monitoring video stream, I don't care too much about latency. It could be WiFi as well. Someday I may dig in a bit more and find out where the bottleneck is.

I think you could probably choose either 1920×1080 15fps or 1280×720 30fps routes and be okay.

Be careful about frames and resolution – Watch your CPU, but also make sure you don't see messages about fps being 'coerced' down or up. Your mileage may vary, and you may have to tweak format, fps, and resolution settings further, check out the mjpg-streamer headquarters on their git hub page (see references down below). You can also see the different modes available to your camera by using this command:

```
1 v4l2-ctl --list-formats-ext
```

Running On Startup

I liked Jacob Salmela's script – all I did was change it for my command line on the stop and restart section for my resolution and frame-rate. Save the following script by doing a sudo vi:

```
1 sudo vi /etc/init.d/livestream.sh
```

```
#!/bin/sh
# /etc/init.d/livestream.sh
### BEGIN INIT INFO
# Provides:          livestream.sh
# Required-Start:    $network
# Required-Stop:     $network
# Default-Start:     2 3 4 5
# Default-Stop:      0 1 6
# Short-Description: mjpg_streamer for webcam
# Description:       Streams /dev/video0 to
http://IP/?action=stream
### END INIT INFO
f_message(){
        echo "[+] $1"
}

# Carry out specific functions when asked to by the
system
case "$1" in
        start)
                f_message "Starting mjpg_streamer"
                /usr/local/bin/mjpg_streamer -b -i
"input_uvc.so -f 15 -r 1920x1080" -o "output_http.so -
w /usr/local/share/mjpg-streamer/www"
                sleep 2
                f_message "mjpg_streamer started"
                ;;
        stop)
                f_message "Stopping mjpg_streamer…"
                killall mjpg_streamer
                f_message "mjpg_streamer stopped"
                ;;
        restart)
                f_message "Restarting daemon:
mjpg_streamer"
                killall mjpg_streamer
                /usr/local/bin/mjpg_streamer -b -i
"input_uvc.so -f 15 -r 1920x1080" -o "output_http.so -
w /usr/local/share/mjpg-streamer/www"
                sleep 2
                f_message "Restarted daemon:
mjpg_streamer"
                ;;
        status)
                pid=`ps -A | grep mjpg_streamer |
grep -v "grep" | grep -v mjpg_streamer. |
awk '{print $1}' | head -n 1`
                if [ -n "$pid" ];
                then
                        f_message "mjpg_streamer is
running with pid ${pid}"
                        f_message "mjpg_streamer was
started with the following command line"
                        cat /proc/${pid}/cmdline
; echo ""
                else
                        f_message "Could not find
mjpg_streamer running"
                fi
                ;;
        *)
                f_message "Usage: $0
{start|stop|status|restart}"
                exit 1
                ;;
esac
exit 0
```

Then we enable this on startup by performing:

```
sudo chmod 755 /etc/init.d/livestream.sh
sudo update-rc.d livestream.sh defaults
```

Now you can reboot your pi, or use commands like ‘sudo service livestream start’ I rebooted my pi, and my fed was running, and running a status command on my service yields:

```
pi@cam-pi-zero:~ $ sudo service livestream status
● livestream.service - LSB: mjpg_streamer for webcam
   Loaded: loaded (/etc/init.d/livestream.sh; generated)
   Active: active (running) since Thu 2020-05-14 11:46:09 CDT; 2min 51s ago
     Docs: man:systemd-sysv-generator(8)
  Process: 411 ExecStart=/etc/init.d/livestream.sh start (code=exited, status=0/SUCCESS)
   Memory: 2.2M
   CGroup: /system.slice/livestream.service
           └─416 /usr/local/bin/mjpg_streamer -b -i input_uvc.so -f 15 -r 1920x1080 -o output_http.so

May 14 11:46:07 cam-pi-zero.local mjpg_streamer[416]: MJPG-streamer [416]: TV-Norm...........: DEFAUL
May 14 11:46:08 cam-pi-zero.local mjpg_streamer[416]: MJPG-streamer [416]: www-folder-path......: /us
May 14 11:46:08 cam-pi-zero.local mjpg_streamer[416]: MJPG-streamer [416]: HTTP TCP port........: 808
May 14 11:46:08 cam-pi-zero.local mjpg_streamer[416]: MJPG-streamer [416]: HTTP Listen Address..: (nu
May 14 11:46:08 cam-pi-zero.local mjpg_streamer[416]: MJPG-streamer [416]: username:password....: dis
May 14 11:46:08 cam-pi-zero.local mjpg_streamer[416]: MJPG-streamer [416]: commands............: ena
May 14 11:46:08 cam-pi-zero.local mjpg_streamer[416]: MJPG-streamer [416]: starting input plugin inpu
May 14 11:46:08 cam-pi-zero.local mjpg_streamer[416]: MJPG-streamer [416]: starting output plugin: ou
May 14 11:46:09 cam-pi-zero.local livestream.sh[411]: [+] mjpg_streamer started
May 14 11:46:09 cam-pi-zero.local systemd[1]: Started LSB: mjpg_streamer for webcam.
```

Conclusion

Using the Raspberry Pi for video streams is good enough if we’re looking for low frame rate monitoring without audio. I’ve yet to find anything that really gives me the frame rate and audio (regardless of latency) that a standard USB webcam directly into my OBS machine would give. Maybe if NDI ever makes it on to the raspberry Pi, or if there is ever SLDP support.

Have you tried streaming across the network on a Raspberry Pi with a USB webcam? Did you fare better than I? Let me know!

Nevertheless, with that, I'm done! I can now embed this into OBS via the VLC media source or anything that can handle an HTTP Motion JPEG video stream. I won't have audio, but that's okay for what I'm using this for.

Raspi-Cameras (libcamera is in, raspivid is out in Bullseye)

Mjpg-Streamer won't have the raspicam input module anymore if you compile on Bullseye, because I believe the headers don't exist as the Bullseye version is going the way of libcamera. While I was able to hack around that and build it, I got HORRIBLE artifacts using a raspberry pi camera with the h264 codec and libcamera. I tried switching codecs to MJPEG instead of H264 and had less artifacts, but worse frame rate. So for now I decided to go with gstreamer instead of mjpg-streamer at 1280×720 for a nice 30fps solution.

For me, what worked was to enable the legacy raspi-camera support (so we're not using libcamera, we're using v4l2).

Here are the rough steps:

- Started out with Bullseye, latest install with sudo apt update/sudo apt upgrade executed.
- Edit /boot/config.txt and change the following:
  - start_x=1 //add this line
  - gpu_mem=128 //probably no higher
  - #camera_auto_detect=1 //Comment this line out
- Restart
- Installs:

```
1
2
```

```
sudo apt-get install gstreamer1.0-tools gstreamer1.0-plugins-base \
    gstreamer1.0-plugins-good gstreamer1.0-plugins-bad
```

Then execute this on the command line:

```
1
```

```
/usr/bin/gst-launch-1.0 v4l2src ! video/x-h264,width=1280,height=720,framerate=30/1 ! h264parse config-interval=1 ! matroskamux streamable=true ! tcpserversink host=::0 port=9000 sync=false sync-method=2
```

Once that's up and running, I ran VLC against it using :network-caching=250 and a URL of tcp://<yourhostname>:9000 and was able to get a decent stream. That's about the extent I've been pushing the raspi-cameras as I typically use the USB side. I did notice it was nice and fluid, along with less power consumption by a 100 milliamps or so, so I think I'll use this as a portable battery powered wireless camera to use around the house on streams. Hmmm, I may need to look at getting a microphone on here somehow and streaming audio with gstreamer as well.