Server - Anleitung zum selber kompilieren

FabAccess kann auf einer ganzen Reihe von Systemen zu Laufen gebracht werden, zum Beispiel:

Betriebssysteme

- auf Linux/Unix-Basis
  - Ubuntu/Kubuntu
  - Arch Linux
  - Raspberry OS
  - Fedora
  - CentOS
  - NixOS → Luca Lutz vom Hackwerk e.V. fragen
  - Synology
  - WSL (Windows Subsystem for Linux)
- FreeBSD
- MacOS

Container/Virtualisierung

- Docker
- Portainer
- Kubernetes → Luca Lutz vom Hackwerk e.V. fragen
- Moby
- Proxmox
- LXC
- runc
- Containerd
- VirtualBox
- Boxes

Empfehlungen für Hardware

Allgemeine Empfehlungen sind immer relativ schwer zu treffen. Je nach Größe der Institution und der vorhandenen Software- und Personallandschaft gibt es unterschiedlichste Auffassungen davon, was benötigt wird und wie es mit anderen Systemkomponenten zusammenspielen soll bzw. muss.Grundsätzlich versuchen wir, FabAccess Server mit möglichst wenig Ressourcenverbrauch zu installieren. Grundsätzlich benötigt:

- Festplattenspeicher: ≥16 GB
  - BFFH schreibt u.U. fleißig Log-Files (Audit). Außerdem werden ggf. weitere Systeme installiert, wie z.B. Scripts oder Monitoring-Services. Wir empfehlen deshalb pauschal 16 GB oder mehr (für Betriebssystem, BFFH, Services, Log Files und Sonstige + Puffer)
  - idealerweise eine schnelle SSD oder Industrial Grade SD-Karte (mit erhöhter Lebensdauer)
- Arbeitsspeicher: ≥ 1,5 GB (zum Kompilieren der Binary)
- CPU Kerne: ≥ 1
- OS Architektur: 64 Bit
- u.U. sinnvoll: USV (unterbrechungsfreie Stromversorgung)
- **stabiles** Wifi-Netzwerk (viele Aktoren werden u.U. lediglich per WLAN angebunden!)

Setup auf Raspberry Pi 3 mit Raspberry Pi OS (64 Bit)

Wir flashen auf einem beliebigen Rechner eine neue SD-Karte mit Hilfe von `rpi-imager`.

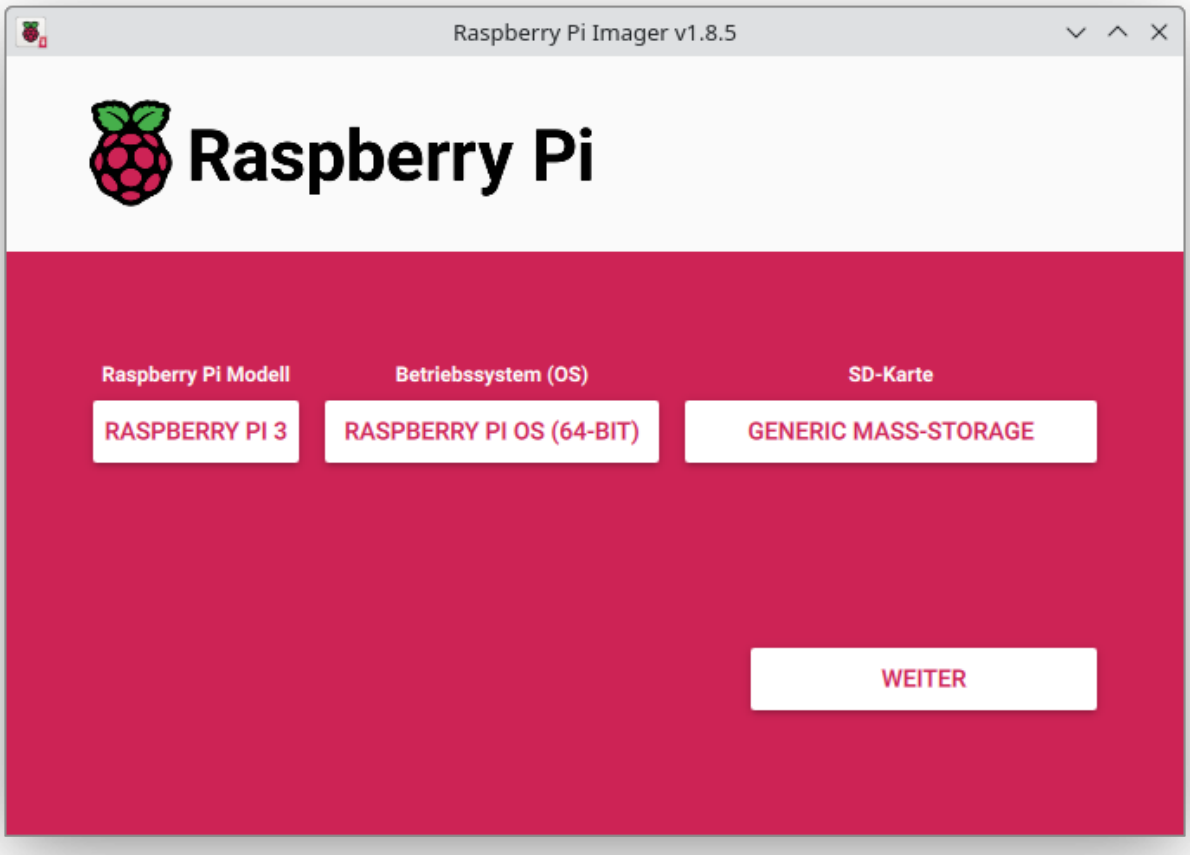
Siehe https://www.raspberrypi.com/software. Unser Raspberry Pi 3 Model B+ ist vergleichsweise ziemlich alt und hat folgende Specs:

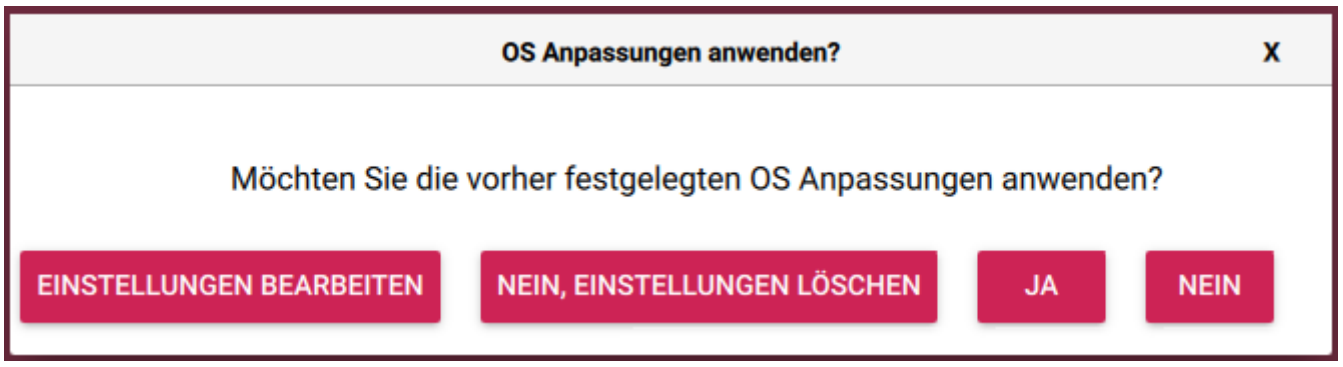
- Broadcom BCM2837B0, Cortex-A53 (ARMv8) 64-Bit-SoC @ 1,4GHz
- 1GB LPDDR2 SDRAM
- 2,4GHz und 5GHz IEEE 802.11.b/g/n/ac Wireless LAN, Bluetooth 4.2, BLE
- Gigabit Ethernet über USB 2.0 (maximaler Durchsatz 300 Mbps)
- Erweiterte 40-polige GPIO-Stiftleiste
- HDMI® in voller Größe
- 4 USB 2.0-Anschlüsse
- CSI-Kamera-Port für den Anschluss einer Raspberry Pi-Kamera
- DSI-Display-Port für den Anschluss eines Raspberry Pi-Touchscreen-Displays
- 4-poliger Stereoausgang und Composite-Video-Anschluss
- Micro-SD-Anschluss zum Laden Ihres Betriebssystems und Speichern von Daten
- 5V/2.5A DC Stromeingang
- Power-over-Ethernet (PoE)-Unterstützung (separater PoE-HAT erforderlich)

Imaging / Provisioning

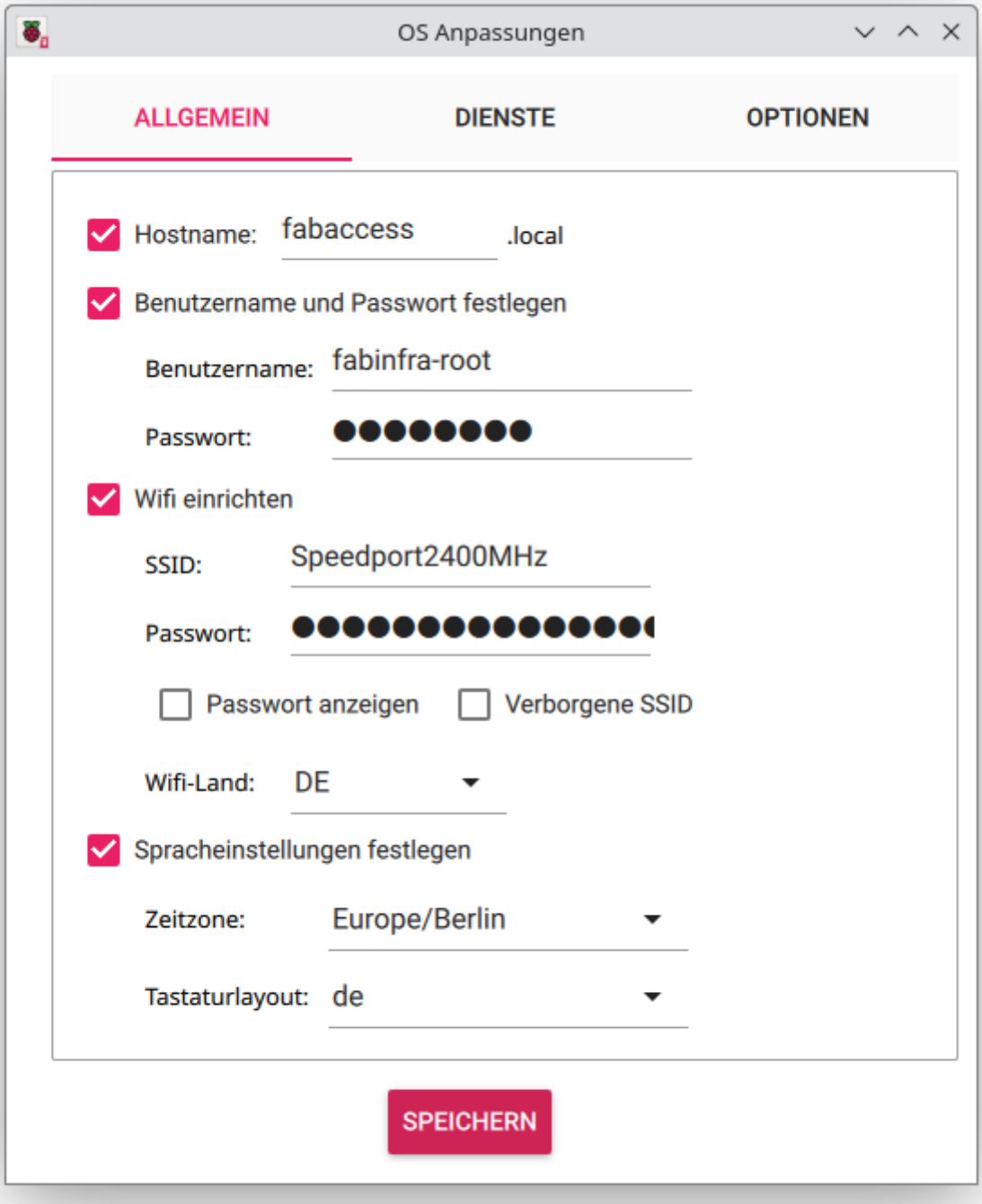
```
sudo apt install rpi-imager
```

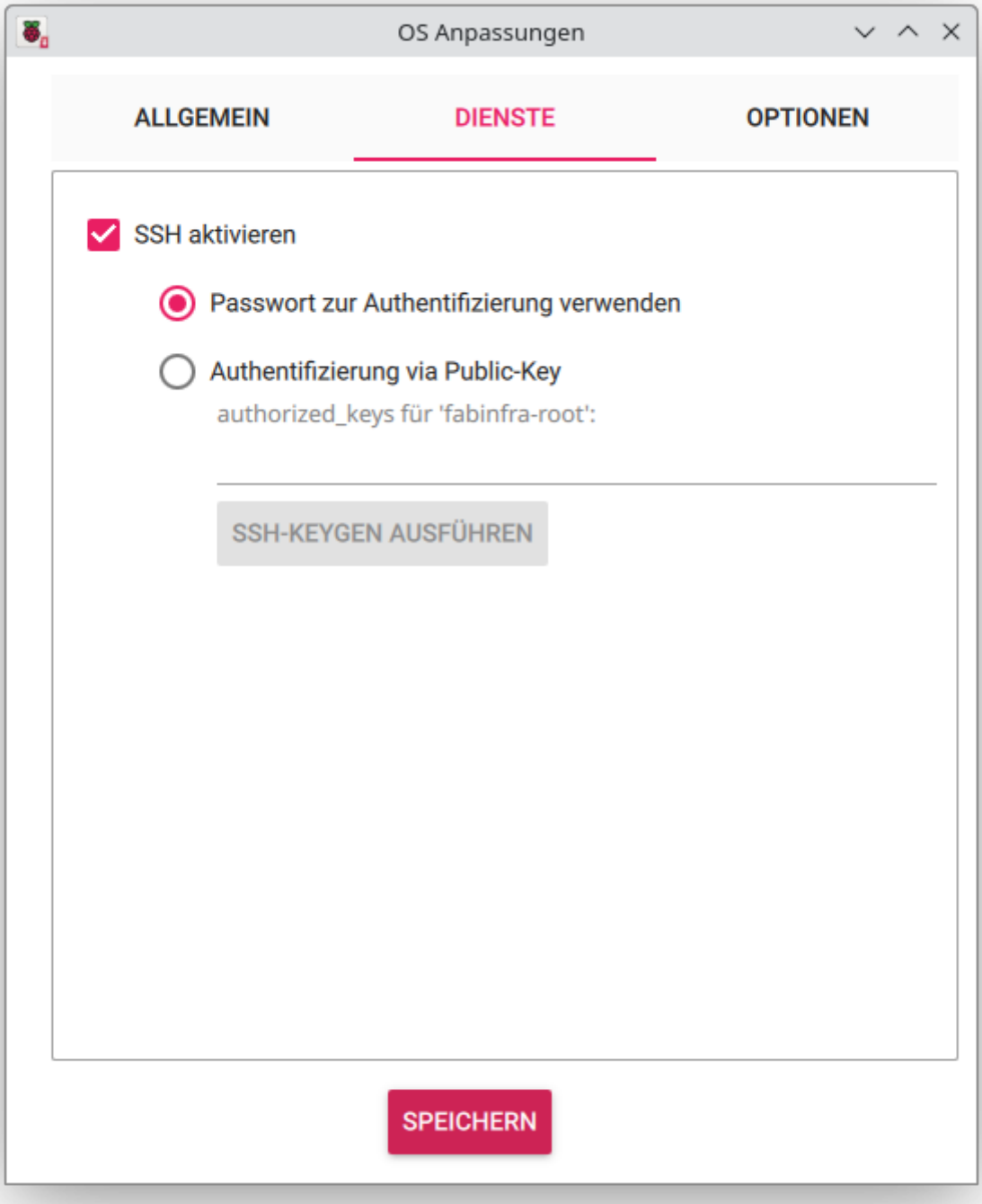
Setup-Schritte mit Screenshots zeigen (aufklappen)

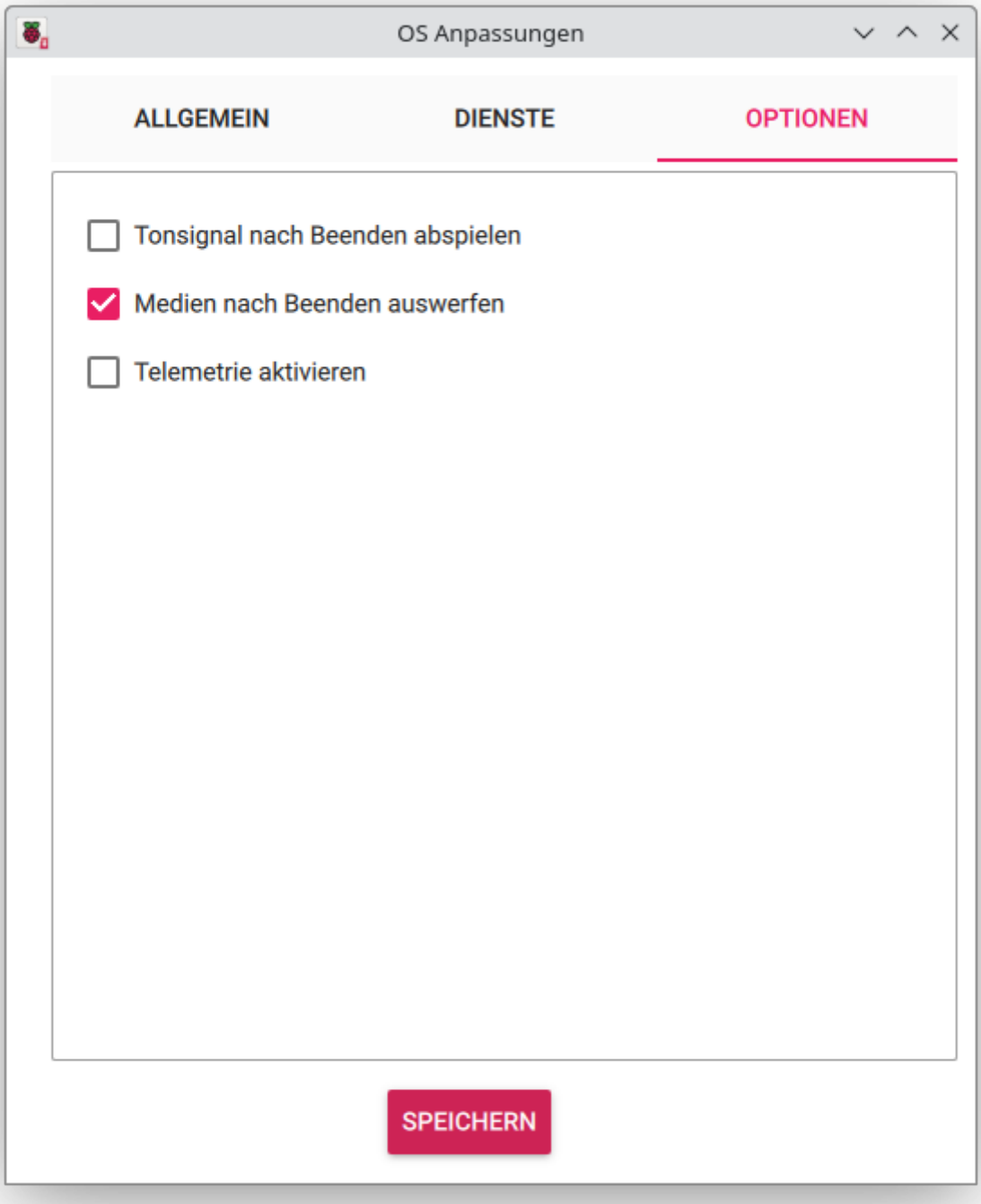




Wir vergeben als Benutzername `fabinfra-root` und das Passwort `vulca`


OS Anpassungen
ALLGEMEIN
DIENSTE
OPTIONEN
Hostname:
fabaccess
.local
Benutzername und Passwort festlegen
Benutzername:
fabinfra-root
Passwort:
Wifi einrichten
SSID:
Speedport2400MHz
Passwort:
Passwort anzeigen
Verborgene SSID
Wifi-Land:
DE
Spracheinstellungen festlegen
Zeitzone:
Europe/Berlin
Tastaturlayout:
de
SPEICHERN


OS Anpassungen
ALLGEMEIN
DIENSTE
OPTIONEN
SSH aktivieren
Passwort zur Authentifizierung verwenden
Authentifizierung via Public-Key
authorized_keys für 'fabinfra-root':
SSH-KEYGEN AUSFÜHREN
SPEICHERN



Unser Image: Debian Bookworm Linux `MP PREEMPT Debian 1:6.6.51-1+rpt3 (2024-10-08) aarch64`

Standardpakete installieren

Wir installieren ein paar Defaults. Einige der folgenden Pakete sind unter Umständen bereits standardmäßig vorinstalliert. Wir geben sie der Vollständigkeit halber trotzdem an.

```
sudo apt update && sudo apt upgrade
sudo apt install -y vim gsasl libgsasl7-dev libssl-dev libclang-dev build-essential cmake clang capnproto cargo
mosquitto mosquitto-clients
```

Die richtige Rust Version installieren (1.66.1)

```
rustc --version
cargo --version

rustc 1.63.0
cargo 1.65.0

# Da wir mit rust 1.63.0 Fehler beim Kompilieren von bffh bekommen, downgraden wir es!
apt purge rustc libstd-rust-1.63 libstd-rust-dev

# Wir installieren das aktuelle Rust (als normaler Nutzer). Jedoch mit dem Unterschied, dass dieses das Tool den
Befehl "rustup" mitbringt, womit wir letztlich die korrekte, alte Zielversion 1.66.1 installieren können
curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs > rust.sh
chmod +x rust.sh
./rust.sh -v #install rust - dauert ca. 10 Minuten - wir installieren die Standardvariante 1)
rm rust.sh

#cargo in .bashrc einfügen
echo 'source "$HOME/.cargo/env"' >> ~/.bashrc
source ~/.bashrc

#zu alter Version downgraden:
rustup install 1.66.1
rustup default 1.66.1

#Version erneut checken
rustc --version
cargo --version

rustc 1.66.1 (90743e729 2023-01-10)
cargo 1.66.1 (ad779e08b 2023-01-10)
```

> Folgenden Fehler erhalten wir später beim Kompilieren von bffh, wenn rust zu neu ist, sodass wir gleich vorbeugen:

```
(signal: 11, SIGSEGV: invalid memory reference)
```

Mosquitto vorbereiten

```
sudo vim /etc/mosquitto/mosquitto.conf
```

Wir lassen Mosquitto auf IPv4 und IPv6 lauschen und binden es an alle Interfaces (`0.0.0.0` bzw `::`), da unsere MQTT Aktoren im Netzwerk sonst den Server nicht erreichen. Aus Sicherheitsgründen erlauben wir keine anonymen Zugriffe (`allow_anonymous false`). Außerdem legen wir eine Passwortdatei an, die mindestens einen Nutzer + Passwort enthält, gegen den sich sowohl der bffh Server, als auch die MQTT Geräte später authentifizieren.

```
# Place your local configuration in /etc/mosquitto/conf.d/
# /usr/share/doc/mosquitto/examples/mosquitto.conf.example

# note: :: does not automatically configure 0.0.0.0
listener 1883 ::
listener 1883 0.0.0.0

allow_anonymous false
password_file /etc/mosquitto/pw.file
pid_file /run/mosquitto/mosquitto.pid
persistence true
persistence_location /var/lib/mosquitto/
#log_dest file /var/log/mosquitto/mosquitto.log
#log to journald
log_dest syslog
include_dir /etc/mosquitto/conf.d
```

Passwortdatei anlegen

Wir erzeugen einen Standardnutzer und nennen ihn `fabaccess-defaultuser` (kann beliebig genannt werden) und vergeben ein Passwort. Wir verwenden als Passwort den Wert `default`

```
sudo mosquitto_passwd -c /etc/mosquitto/pw.file fabaccess-defaultuser
sudo chmod 400 /etc/mosquitto/pw.file
```

> Es können beliebig viele Nutzer in die Passwortdatei geschrieben werden. Das hängt vom gewünschten Setup ab!

Berechtigungen anpassen

```
sudo chown -R mosquitto:mosquitto /etc/mosquitto/
```

Mosquitto Dienst starten

```
sudo systemctl restart mosquitto.service
sudo journalctl -f -u mosquitto.service
```

Je nach Konfiguration von Mosquitto finden wir die Log-Einträge nun entweder per journalctl im syslog oder in einer separaten Datei.

Swap File erhöhen

BFFH benötigt zum Kompilieren mehr als 1 GB RAM. Ein Raspberry Pi 3 hat nur 1 GB. Dadurch stürzt er beim Kompilieren mit OOM-Fehlern ab. Deshalb erhöhen wir den Swap temporär auf 2 GB. Dieser Trick basiert auf

https://www.alibabacloud.com/blog/optimization-on-memory-usage-during-rust-cargo-code-compiling_601189

```
sudo su
dphys-swapfile swapoff
echo "CONF_SWAPSIZE=2048" > /etc/dphys-swapfile
dphys-swapfile setup
dphys-swapfile swapon

# Überprüfen von "Swp" mit htop
htop
```

BFFH Nutzer anlegen

Wir legen einen gesonderten Nutzer zum Ausführen von BFFH an. Es ist ein Systembenutzer ohne eigenes Homeverzeichnis (`-s` Parameter).

```
sudo useradd -m -s /bin/bash bffh
```

Damit der Nutzer bffh allerdings Python-Scripts ausführen darf, benötigt er weitere Gruppenberechtigungen

BFFH Installation (Version "interfacer" - entspricht v0.4.3 Alpha - vom 17.03.2023)

Nach dem Installieren von Mosquitto können wir uns unserem BFFH Server widmen. Wir arbeiten nachfolgend im Verzeichnis `/opt/fabinfra/`. Es kann aber jedes andere Verzeichnis verwendet werden

```
mkdir -p /opt/fabinfra/
cd  /opt/fabinfra/
git clone https://gitlab.com/fabinfra/fabaccess/bffh.git --recursive
cd /opt/fabinfra/bffh/

#wir wechseln den Branch auf die aktuellste Entwicklerversion. Siehe https://gitlab.com/fabinfra/fabaccess/bffh/-/commits/feature%2Fclaims_api -> das lassen wir, denn es klappt nicht mit "cargo build --release"
#git checkout feature/claims_api

#wir wechseln zum Branch "development"
git checkout development

# Submodules updaten
git submodule update --remote

# BFFH kompilieren
cargo build --release #dauert ca. 25-35 Minuten

# Standard-Dateien kopieren
cp -R /opt/fabinfra/bffh/examples/ /opt/fabinfra/bffh/target/release/
```

Zertifikat erzeugen (self signed)

Das Zertifikat benötigen wir für den bffh Server, da wir es in der Konfigurationsdatei angeben müssen.

```
sudo mkdir -p /opt/fabinfra/bffh-data/cert/
sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout /opt/fabinfra/bffh-data/cert/bffh-selfsigned.key -out /opt/fabinfra/bffh-data/cert/bffh-selfsigned.crt
```

Das Ergebnis der folgenden Prüfung muss lauten: `Version: 3 (0x2)`

```
sudo openssl x509 -in /opt/fabinfra/bffh-data/cert/bffh-selfsigned.crt -noout -text | grep "Version"
```

Konfiguration (bffh.dhall) anlegen

Wir legen ein neues Verzeichnis an und kopieren die Beispielkonfiguration dort hin.

```
mkdir -p /opt/fabinfra/bffh-data/config/
cp /opt/fabinfra/bffh/examples/bffh.dhall /opt/fabinfra/bffh-data/config/bffh.dhall
```

```
vim /opt/fabinfra/bffh-data/config/bffh.dhall
```

Wir passen die Konfiguration nun an unsere Wünsche an. Die Referenz für die Konfiguration findest du hier.

Nutzerdatenbank (users.toml) anlegen und importieren

```
cp /opt/fabinfra/bffh/examples/users.toml /opt/fabinfra/bffh-data/config/users.toml
```

```
vim /opt/fabinfra/bffh-data/config/users.toml
```

Hier geht's zur Dokumentationen der users.toml

Nach dem Anlegen der Nutzerdatenbank importieren wir diese in bffh mit folgendem Kommando (dabei ist auch die Hauptkonfiguration `bffh.dhall` anzugeben!):

```
/opt/fabinfra/bffh/target/release/bffhd --config /opt/fabinfra/bffh-data/config/bffh.dhall --load /opt/fabinfra/bffh-data/config/users.toml
```

> Bei jeder Änderung der `users.toml` Datei muss ein Import durchgeführt werden, da bffh die Änderungen nicht automatisch einliest!

Berechtigungen und Besitzer anpassen

Damit nur Nutzer und Gruppe, nicht aber Externe auf die Dateien zugreifen können, beschränken wir den Zugriff auf unseren Ordner `/opt/fabinfra`. In diesem Verzeichnis befinden sich nur Dienste und Configs, die von `bffh` ausgeführt werden sollen. Sollten Dateien nachträglich hinzugefügt oder modifiziert werden ist darauf zu achten, dass der Eigentümer und die Berechtigungen entsprechend angepasst werden.

```
BFFH_DIR="/opt/fabinfra/"
sudo find $BFFH_DIR -type d -exec chmod 750 {} +
sudo find $BFFH_DIR -type f -exec chmod 640 {} +
chown -R bffh:bffh $BFFH_DIR
```

Achtung beim Naming

> `bffhd` ist unser bffh **Daemon** und impliziert mit seinem Namen, dass das Programm permanent laufen soll. Wir verwenden nachfolgend systemd zum Installieren von `bffhd` als Service. Den Service nennen wir jedoch `bffh.service` - so, wie der Serverdienst heißt.

Server starten (manueller Test)

Wir starten den Dienst `bffhd` manuell und dann prüfen wir, ob er auf dem richtigen Port und Interface lauscht:

```
/opt/fabinfra/bffh/target/release/bffhd --verbose --config /opt/fabinfra/bffh-data/config/bffh.dhall --log-format Pretty


netstat -an | grep "LISTEN" | grep -v "LISTENING"
```

> Hinweis: `bffhd` erzeugt oder verwendet beim Starten zwei Dateien, und zwar eine Lock-Datei namens `bffh-lock` und eine interne Datenbankdatei `bffh`, die die Machine States und die Benutzer enthält, die aus `users.toml` stammen.

systemd Service anlegen und bffh automatisch starten

Wenn unser manueller Test geklappt hat, dann sollten wir bffh als Dienst installieren und uns die Arbeit abnehmen lassen:

```
vim /opt/fabinfra/bffh-data/bffh.service
```

```
[Unit]
Description=FabAccess BFFH Service
After=network.target

[Service]
Type=simple
User=bffh
Group=bffh
ExecStartPre=/opt/fabinfra/bffh/target/release/bffhd --check --config /opt/fabinfra/bffh-data/config/bffh.dhall
#BFFH_LOG levels: trace,debug,warn,info,error
Environment="BFFH_LOG=warn"
#log-format: Full, Compact, Pretty
ExecStart=/opt/fabinfra/bffh/target/release/bffhd --verbose --config /opt/fabinfra/bffh-data/config/bffh.dhall --log-format Pretty
Restart=on-failure
RestartSec=30
ExecStopPost=/usr/bin/bash /opt/fabinfra/scripts/bffh-state.sh $EXIT_CODE

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

Dienst aktivieren, starten und prüfen

```
sudo ln -sf /opt/fabinfra/bffh-data/bffh.service /etc/systemd/system/bffh.service
sudo systemctl daemon-reload
sudo systemctl enable bffh.service --now
journalctl -f -u bffh.service
```

Log-Größe von journalctl beachten

Je nach Konfiguration des Log Levels von BFFH und der Anzahl der aktiven Nutzer kann die Logdatei innerhalb von 1-2 Wochen auf über 5 GB und mehr anwachsen und insbesondere bei Systemen auf Basis von SD-Karten zum Defekt des Schreibmediums führen, denn diese altern durch die vielen Schreibzyklen sehr schnell. im produktiven Fall ist deshalb zum einen darauf zu achten, dass die SD-Karte stets überlebt (Log Level auf `Info` setzen), und dass das System nicht voll läuft. Hier ein paar Kommandos, um das Journal zu leeren (dies schützt nicht vor den Schreibzyklen, aber erlaubt eine bessere Übersicht).

```
sudo journalctl --rotate
sudo journalctl --vacuum-time=1s
```

Auch syslog läuft unter Umständen voll und kann geleert werden. Das ist allerdings ein sehr unschöner Trick und sollte nur zu absoluten Testzwecken angewendet werden:

```
sudo truncate -s 0 /var/log/syslog
```

Arch Linux

```
sudo pacman -Syu
sudo pacman -S make cmake clang gsasl
sudo pacman -S git rust capnproto
```

> Todo ...

CentOS

```
sudo yum update
sudo yum install curl && curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh
sudo yum install epel-release && sudo yum install capnproto
sudo yum install https://packages.endpointdev.com/rhel/7/os/x86_64/endpoint-repo.x86_64.rpm && sudo yum install git
sudo yum install centos-release-scl && yum install llvm-toolset-7 && scl enable llvm-toolset-7 bash (Change bash to youre shell)
sudo yum install gcc-c++ libgsasl-devel openssl-devel cmake
```

> Todo ...

Version #52
Erstellt: 23 Oktober 2024 22:56:38 von Mario Voigt (Stadtfabrikanten e.V.)
Zuletzt aktualisiert: 13 Dezember 2024 17:03:41 von Mario Voigt (Stadtfabrikanten e.V.)