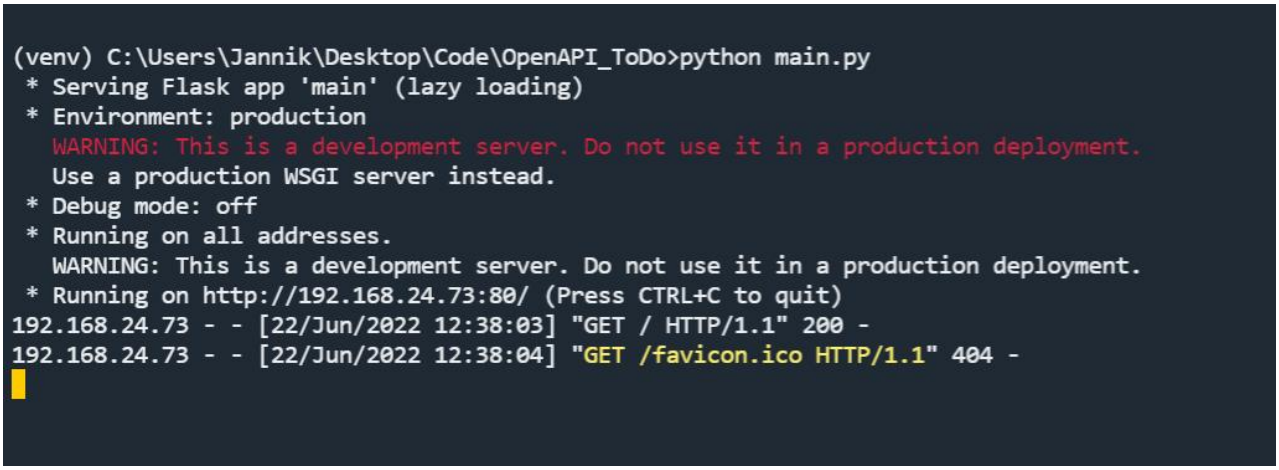
Deployment einer Web-Applikation



Jannik Möbius, Armin Spöllmann
LF9: Netzwerke und Dienste bereitstellen
Herr Wichmann
BBS-Brinkstraße
IFA02

Deployment einer Webanwendung

Von Armin Spöllmann und Jannik Möbius
https://github.com/ItZzMJ/ToDoList_OpenAPI

Inhaltsverzeichnis

Vorgehen

1. Netzwerkkonfiguration

Damit beim nächsten start des Raspberry PIs die IP Adresse gleich bleibt, muss zuerst eine statische IP Adresse eingerichtet werden.

1.1 Statische IP Festlegen

Zuerst muss die Datei **/etc/dhcpcd.conf** wie folgt editiert werden um eine statische IP zu erhalten. Um die Datei zu editieren wird ein Editor benötigt. Hierzu wird der VI Editor verwendet.

```
vi /etc/dhcpd.conf
```

```
#static IP configuration LAN
interface eth0
static ip_address=192.168.24.134/24
static routers=192.168.24.254
static domain_name_servers=192.168.24.254

#static IP configuration WLAN
interface wlan0
static ip_address=192.168.24.164/24
static routers=192.168.24.254
static domain_name_servers=192.168.24.254
```

Mit der Tastatureingabe **:wq** wird die Datei gespeichert und der VI Editor geschlossen.

```
:wq
```

Damit die Änderungen wirksam werden muss der Daemon neugestartet werden.

```
sudo systemctl restart dhcpcd
```

1.2 WLAN-Verbindung herstellen

Dazu muss die Datei **/etc/wpa_supplicant/wpa_supplicant.conf** editiert werden. Bei **SSID** muss der WLAN Name eingetragen werden und unter **PASSWORD** das Passwort.

```
ctrl_interface=DIR=/var/run/wpa_supplicant
GROUP=netdev
country=de
update_config=1
network={
    ssid="<SSID>"
    psk="<PASSWORD>"
}
```

Auch hier muss der Service neugestartet werden damit die Änderungen wirksam werden.

```
sudo systemctl restart dhcpcd
```

Nun kann die Liste der verfügbaren Netzwerke angezeigt werden.

```
wpa_cli -i wlan0 list_networks
```

Falls keine Netzwerke angezeigt werden, muss ein Netzwerk eingerichtet werden. Dazu muss man die CLI öffnen und die folgenden Befehle ausführen.

```
# CLI öffnen
wpa-cli
# Nach WLAN Netzwerken scannen
scan
# Ergebnis des Scans anzeigen
scan_results
add_network
add_network 0 ssid "SSID"
add_network 0 psk "passphrase"
enable_network 0
save_config
quit
```

Nun ist die Anmeldung im WLAN möglich.

```
wpa_cli -i wlan0 select_network 0
```

2. Benutzer anlegen und konfigurieren

Benutzer werden mit dem adduser Befehl hinzugefügt.

```
sudo adduser benutzer72
sudo adduser fernzugriff
```

Fernzugriff soll sudo Rechte bekommen also wird er zur sudo group hinzugefügt

```
sudo adduser fernzugriff sudo
```

Da Sudo Benutzer automatisch SSH Rechte haben, müssen diese nicht manuell eingerichtet werden.

---

3. Firewall Konfiguration

Um nur Verbindungen zu dem Server über bestimmte Ports zu erlauben, muss zuerst eine Firewall installiert werden.

```
sudo apt-get install ufw
```

Um sich mit dem Server per SSH zu verbinden sobald die Firewall aktiv ist, muss der Port 22 freigegeben werden. Dieser soll aber nur aus dem lokalen Netzwerk erreichbar sein, also wird die Regel wie folgt gesetzt.

```
sudo ufw allow from 192.168.24.0/16 to any port 22
```

Damit die Webseite von außen auch erreichbar ist muss der Port 80 für HTTP und optional auch noch der Port 443 für HTTPS freigegeben werden.

```
sudo ufw allow 80
sudo ufw allow 443
```

Damit die Änderungen aktiv werden, muss die Firewall noch aktiviert werden.

```
sudo ufw enable
```

4. ToDo-Listen-Verwaltung deployen

Um das Hochladen der ToDo-Listen-Verwaltung auf den Server zu vereinfachen, wird der Code aus dem Git-Repository geklont. Dazu wird zuerst Git auf dem Server installiert.

```
sudo apt install git
```

Nun kann das Repository einfach geklont werden.

```
git clone https://github.com/ItZzMJ/ToDoList_OpenAPI.git
```

Als nächstes müssen die Dependencies der App installiert werden. Dafür wird zuerst in das Verzeichnis der App gewechselt.

```
cd ToDoList_OpenAPI
```

Und dann mit folgendem Kommando die Dependencies aus **requirements.txt** installiert.

```
pip install -r requirements.txt
```

Anschließend kann die App gestartet werden.

```
python main.py
```

Nun ist die ToDo-Listen-Verwaltung über die IP 192.168.24.164 erreichbar.

4.1 Autostart

Damit die App bei Serverneustart oder bei einem Error wieder automatisch startet, wird ein **Supervisor** installiert.

```
sudo apt install supervisor
```

Um den Supervisor zu konfigurieren wird zu erst für die ToDo-Listen App zuerst eine Konfigurationsdatei in **/etc/supervisor/conf.d/** erzeugt.

```
sudo nano /etc/supervisor/conf.d/todolist.conf
```

In dieser Konfigurationsdatei wird dem Programm zuerst ein Name gegeben.

```
[program:flask_app]
```

Danach muss das Kommando festgelegt werden, welches überwacht werden soll.

```
command = python main.py
```

Außerdem muss das Verzeichnis der App festgelegt werden.

```
directory = /home/pi/code/ToDoList_OpenAPI/
```

Zuletzt muss noch der Autostart festegelegt werden.

```
autostart = true
autorestart = true
```

Am Ende sollte die Konfigurationsdatei wie folgt aussehen:

```
[program:flask_app]
command = python main.py
directory = /home/pi/code/ToDoList_OpenAPI/
autostart = true
autorestart = true
```

Damit die Konfigurationsdatei vom Supervisor erkannt wird, wird folgendes Kommando ausgeführt.

```
sudo supervisorctl reread
```

Falls eine bestehende Supervisor-konfiguration verändert wurde, sollte der Supervisor aktualisiert werden.

```
sudo supervisorctl update
```

Nun wird der Supervisor für die ToDo-Listen-Verwaltung gestartet.

```
sudo supervisorctl start flask_app
```

4.2 Mit Docker und einem Nextcloud-Container deployen

Um die ToDo-Listen-Verwaltung als Container mit Docker zu deployen, muss zuerst Docker, Docker-Compose und weitere Dependencies installiert werden.

```
sudo apt install docker-ce docker-ce-cli containerd.io docker-compose-plugin
```

Damit Docker auch als nicht Root-User verwenden zu können, muss der aktuelle Benutzer zur Docker Gruppe hinzugefügt werden.

```
sudo groupadd docker
sudo usermod -aG docker pi
```

Um die Anwendung in ein Dockerimage zu verpacken wird zuerst eine **Dockerfile** benötigt die den Aufbau des Images beschreibt. Als Basis wird ein Alpine-Container mit vorinstalliertem Python verwendet. In dieser Dockerfile wird das Arbeitsverzeichniss festgelegt und die benötigten Dependencies installiert. Außerdem wird der Port 80 des Containers geöffnet und der Code der App in den Container kopiert.

```
# syntax=docker/dockerfile:1
FROM python:3.7-alpine
WORKDIR /code
RUN apk add --no-cache gcc musl-dev linux-headers nano bash
COPY ./requirements.txt requirements.txt
RUN pip install -r requirements.txt
EXPOSE 80
COPY . /data
CMD ["python", "/data/main.py"]
```

Um den Container leichter mit Docker-compose zu managen, wird eine **docker-compose.yml** Datei geschrieben.

```
version: '3'
services:
  flask:
    build: .
    ports:
      - "80:80"
      - "8000:5000"
    restart: unless-stopped
```

Bei dieser Gelegenheit bietet es sich an einen weiteren Service einzurichten, Nextcloud. Dafür wird dieses mal keine Dockerfile benötigt da ein vorgefertigtes Image verwendet wird.

```
  nextcloud:
    image: nextcloud:latest
    ports:
      - "8080:80"
    volumes:
      - ./nextcloud:/var/www/html
    restart: unless-stopped
```

Nun werden die Container aus den Images erzeugt.

```
docker-compose build
```

Zu aller Letzt werden die Container gestartet.

```
docker-compose up -d
```

Nun ist die Einrichtung abgeschlossen.