Raspberry Pi 4 (2GB RAM) – Heimserver CoreStack

Ein Wissens- & Schulungsdokument von Olaf Droste Products

Wiki: droste-wiki

URL:https://wiki.droste-home.net

Raspberry Pi 4 (2GB RAM) – Heimserver Setup (RAM-optimiert)

Ziel: Stabiler Betrieb mit **DokuWiki**, **File Browser**, **Cockpit** und **Gogs** (Git) auf einem Raspberry Pi 4 mit 2GB RAM. **Wichtig:** Alle Befehle sind **copy-paste-fertig** – einfach ins Terminal einfügen!

---

1) Terminal-Cheatsheet (Copy & Paste)

Alles hier ist so geschrieben, dass du es **1:1 ins Terminal kopieren** kannst. Unter jedem Block steht kurz, was er macht.

1.1 System-Check (Architektur & RAM)

```
getconf LONG_BIT
```

Ausgabe: 64 → Dein System läuft in **64-Bit** (wichtig für Docker-Images).

```
free -h
```

Zeigt freien RAM an (sollte ~1.5GB frei sein, bevor wir starten).

```
df -h
```

Zeigt freien Speicherplatz (mind. 10GB frei auf / empfohlen).

1.2 System aktualisieren

```
sudo apt update && sudo apt full-upgrade -y
sudo reboot
```

Aktualisiert alle Pakete und startet neu (wichtig nach Kernel-Updates).

Optional (nur Raspberry Pi OS):

```
sudo raspi-config
```

→ **Advanced Options → Expand Filesystem** (nutze die volle SSD-Größe).

1.3 Basis-Tools installieren

```
sudo apt install -y mc git curl wget
```

- **mc** (Midnight Commander) – Textbasierter Dateimanager
- **git** – Wird für Gogs benötigt
- **curl/wget** – Download-Tools

---

2) Digitalcraft Ordnerstruktur (einheitlich!)

Root: `/srv/docker/corestack` **Pro Dienst ein Ordner:**

- `/srv/docker/corestack/wikicore` (DokuWiki)
- `/srv/docker/corestack/filebrowser` (File Browser)
- `/srv/docker/corestack/gogs` (Git-Repository)
- `/srv/docker/corestack/shared` (Gemeinsame Daten)

2.1 Ordner anlegen

```
sudo mkdir -p
/srv/docker/corestack/{wikicore,filebrowser,gogs,shared}
sudo chown -R $USER:$USER /srv/docker/corestack
```

Erklärung:

- `mkdir -p` erstellt alle Ordner inkl. Zwischenverzeichnisse
- chown setzt dich als Besitzer (kein `sudo` mehr nötig)

2.2 In den Stack-Root wechseln

```
cd /srv/docker/corestack
```

Ab hier arbeitest du im CoreStack-Root.

3) Cockpit installieren (Host-System)

```
sudo apt install -y cockpit
sudo systemctl enable --now cockpit.socket
```

Zugriff:

```
https://<PI-IP>:9090
```

Nutze Cockpit für:

- System-Logs
- Dienste verwalten
- Updates
- Speicherplatz überwachen

---

4) Docker + Compose installieren

4.1 Grundtools installieren

```
sudo apt-get install -y ca-certificates curl gnupg
```

4.2 Keyring-Ordner anlegen

```
sudo install -m 0755 -d /etc/apt/keyrings
```

4.3 Docker GPG-Key speichern

```
curl -fsSL https://download.docker.com/linux/debian/gpg | sudo gpg --
dearmor -o /etc/apt/keyrings/docker.gpg
```

4.4 Leserechte setzen

```
sudo chmod a+r /etc/apt/keyrings/docker.gpg
```

4.5 Docker Repo eintragen

```
echo \
  "deb [arch=$(dpkg --print-architecture) signed-
by=/etc/apt/keyrings/docker.gpg]
https://download.docker.com/linux/debian \
  $(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt-get update
```

4.6 Docker installieren

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

4.7 Docker ohne sudo nutzen

```
sudo usermod -aG docker $USER
newgrp docker
docker version
docker compose version
```

Teste Docker:

```
docker run --rm hello-world
```

→ Sollte „Hello from Docker!“ ausgeben.

4.8 Gemeinsames Docker-Netzwerk erstellen

```
docker network create dc-net
docker network ls | grep dc-net
```

Zweck: Alle Container kommunizieren über `dc-net` (z. B. `dc-gogs` statt IP-Adressen).

---

5) Docker-Stacks einrichten

Jeder Dienst hat seinen eigenen Ordner mit `docker-compose.yml`. **RAM-Limits sind**

strikt gesetzt!

Stack 01 – DokuWiki (WikiCore)

Pfad: `/srv/docker/corestack/wikicore/`

1) .env

Datei: `/srv/docker/corestack/wikicore/.env`

```
TZ=Europe/Berlin
PUID=1000
PGID=1000
WIKICORE_PORT=8080
```

Hinweis: PUID/PGID mit `id -u` und `id -g` prüfen.

2) docker-compose.yml

Datei: `/srv/docker/corestack/wikicore/docker-compose.yml`

```yaml
services:
  wikicore:
    image: lscr.io/linuxserver/dokuwiki:latest
    container_name: dc-wikicore
    environment:
      - TZ=${TZ}
      - PUID=${PUID}
      - PGID=${PGID}
    volumes:
      - ./config:/config
    ports:
      - "${WIKICORE_PORT}:80"
    networks:
      - dc-net
    restart: unless-stopped
    deploy:
      resources:
        limits:
          memory: 256M  # RAM-Limit
```

```
networks:
  dc-net:
    external: true
```

3) Start

```
cd /srv/docker/corestack/wikicore
docker compose up -d
docker compose ps
```

Zugriff:

```
http://<PI-IP>:8080
```

Stack 02 – File Browser

Pfad: `/srv/docker/corestack/filebrowser/`

1) Vorbereitung

Viele Docker-Images laufen nicht als root, sondern als normaler Benutzer im Container. Damit der Container Daten speichern kann, müssen die gemounteten Ordner die passenden Rechte haben.

Wir verwenden dafür die Benutzer-ID 1000, die auf den meisten Linux-Systemen dem ersten Benutzer entspricht.

Ordner erstellen

```
sudo mkdir -p /srv/docker/corestack/filebrowser/config
cd /srv/docker/corestack/filebrowser
```

Rechte setzen

```
sudo chown -R 1000:1000 config
```

Damit darf der Container später in diesen Ordner schreiben.

Warum ist das wichtig?
Wenn die Rechte nicht stimmen, erscheint später ein Fehler wie:

```
Permission denied
cp: can't create '/config/settings.json'
```

Nach dem Setzen der Rechte kann der Container problemlos starten.

2) .env

Datei: `/srv/docker/corestack/filebrowser/.env`

```
TZ=Europe/Berlin
PUID=1000
PGID=1000
FILEBROWSER_PORT=8081
```

3) docker-compose.yml

Datei: `/srv/docker/corestack/filebrowser/docker-compose.yml`

```yaml
services:
  filebrowser:
    image: filebrowser/filebrowser:latest
    container_name: dc-filebrowser
    environment:
      - TZ=${TZ}
      - PUID=${PUID}
      - PGID=${PGID}
    volumes:
      - /srv/docker/corestack/shared:/srv  # Gemeinsamer Ordner
      - ./config:/config
    ports:
      - "${FILEBROWSER_PORT}:80"
    networks:
      - dc-net
    restart: unless-stopped
    deploy:
      resources:
        limits:
          memory: 128M  # RAM-Limit

networks:
  dc-net:
    external: true
```

4) Start

```
cd /srv/docker/corestack/filebrowser
docker compose up -d
docker compose ps
```

Zugriff:

```
http://<PI-IP>:8081
```

Einloggen mit:

```
Username: admin
Password: admin
```

Stack 03 – Gogs (Git-Repository)

Pfad: `/srv/docker/corestack/gogs/`

1) .env

Datei: `/srv/docker/corestack/gogs/.env`

```
TZ=Europe/Berlin
GOGS_HTTP_PORT=3000
GOGS_SSH_PORT=2222
```

2) docker-compose.yml

Datei: `/srv/docker/corestack/gogs/docker-compose.yml`

```
services:
  gogs:
    image: gogs/gogs:latest
    container_name: dc-gogs
    environment:
      - TZ=${TZ}
    volumes:
      - ./data:/data
```

```
    ports:
      - "${GOGS_HTTP_PORT}:3000"  # Web-UI
      - "${GOGS_SSH_PORT}:22"     # Git über SSH
    networks:
      - dc-net
    restart: unless-stopped
    deploy:
      resources:
        limits:
          memory: 256M  # RAM-Limit

networks:
  dc-net:
    external: true
```

3) Start

```
cd /srv/docker/corestack/gogs
docker compose up -d
docker compose ps
```

4) Gogs konfigurieren

```
1. Öffne ''http://<PI-IP>:3000'' im Browser
2. **Datenbank:** Wähle **SQLite3** (kein PostgreSQL nötig!)
3. **Domain:** ''<PI-IP>'' (oder Hostname, falls DNS eingerichtet)
4. **SSH-Port:** ''2222'' (falls Git über SSH genutzt wird)
5. **Admin-Benutzer erstellen** und speichern
```

Zugriff:

```
http://<PI-IP>:3000
```

Git SSH:

```
ssh -p 2222 git@<PI-IP>
```

---

6) Start-Reihenfolge (empfohlen)

```
# 1) DokuWiki (leichtgewichtig)
cd /srv/docker/corestack/wikicore && docker compose up -d
```

```
# 2) File Browser
cd /srv/docker/corestack/filebrowser && docker compose up -d

# 3) Gogs (Git)
cd /srv/docker/corestack/gogs && docker compose up -d
```

---

7) Update-Routine (pro Stack)

```
cd /srv/docker/corestack/<stack>
docker compose pull
docker compose up -d
docker image prune -f
```

Erklärung:

- `pull` lädt neue Images
- `up -d` aktualisiert den Container
- `image prune -f` löscht alte Images

---

8) Backup-Quicklist (Pflicht)

Wichtige Pfade sichern:

- `/srv/docker/corestack/wikicore/config/` (DokuWiki)
- `/srv/docker/corestack/gogs/data/` (Git-Repos + Config)
- `/srv/docker/corestack/filebrowser/config/` (File Browser Einstellungen)
- `/srv/docker/corestack/shared/` (Gemeinsame Daten)

Backup-Befehl (Beispiel):

```
cd /srv/docker/corestack
tar -czvf backup_$(date +%Y-%m-%d).tar.gz wikicore/config gogs/data
filebrowser/config shared/
```

---

9) Dienste testen (Browser)

Ersetze `<PI-IP>` durch die IP deines Raspberry Pi:

Dienst	URL	Port
Gogs	`http://<PI-IP>:3000`	3000
DokuWiki	`http://<PI-IP>:8080`	8080
File Browser	`http://<PI-IP>:8081`	8081
Cockpit	`https://<PI-IP>:9090`	9090

---

10) RAM-Überwachung

```
sudo apt install htop
htop
```

Ziel: < 1.8GB RAM-Verbrauch (sonst Swapping → Performance-Einbruch).
Optimierungstipps:

- **Gogs:** In `app.ini` `MAX_LOAD = 5` setzen (begrenzt gleichzeitige Anfragen)
- **DokuWiki:** PHP-Cache aktivieren (in `php.ini`)

---

11) Optional: RAM-Upgrade

Falls du später **mehr Dienste** (z. B. n8n, code-server) nutzen willst:

- **Raspberry Pi 4 mit 4GB oder 8GB RAM** nachrüsten (~50–100€)
- **Dann** die **ursprüngliche CoreStack-Doku** verwenden

Fertig! Dein **Raspberry Pi 4 (2GB RAM)** läuft jetzt stabil mit:

- **DokuWiki** (Dokumentation)
- **File Browser** (Dateimanagement)
- **Gogs** (Git-Repository)
- **Cockpit** (Systemverwaltung)

Viel Erfolg mit deinem neuen Home-Server!
