Serie: Digitalcraft WikiCore – CoreStack **Ziel:** Modernes Self-Hosting auf dem Raspberry Pi 4 mit Docker/Compose – inkl. n8n, Gitea, VS Code Server, Filebrowser und DokuWiki als **Digitalcraft WikiCore**. **Hinweis:** Die SSD ist bereits vorbereitet (Raspberry Pi OS läuft direkt auf der 1TB-SSD). Das SSD-Setup wurde in einem früheren Video gezeigt.

---

1) Titelvorschläge (SEO + Klickmotiv)

1. **Raspberry Pi 4 CoreStack: Docker + n8n + DokuWiki (WikiCore) in einem System**
2. **Modern Self-Hosting auf dem Raspberry Pi 4: Mein Digitalcraft CoreStack**
3. **Pi 4 (2GB) als Home-Server: Docker-Stack mit n8n, Gitea, VS Code & DokuWiki**
4. **Von Null zum CoreStack: Docker-Plattform auf SSD (Pi 4) – sauber & wartbar**
5. **Digitalcraft WikiCore: Die moderne Kurs-Zentrale auf Raspberry Pi 4 (Docker)**

---

2) Thumbnail-Hook (kurz, hart, klickbar)

1. **CORESTACK in 30 MIN**
2. **Pi4 + Docker = MODERN**
3. **n8n + WikiCore + Git**
4. **SD? NEIN. SSD!**

---

3) Kapitelmarken (YouTube Chapters)

1. **0:00** Intro: Digitalcraft WikiCore & CoreStack-Ziel
2. **0:45** Voraussetzung: Raspberry Pi OS läuft direkt auf SSD (Link zum SSD-Video)
3. **1:15** Digitalcraft-Standard: Ordnerstruktur `/srv/docker/corestack`
4. **2:30** System-Update + Architektur-Check (64-bit?)
5. **4:00** Docker + Compose installieren
6. **7:00** CoreStack Deploy: Postgres + n8n + DokuWiki + Gitea + VS Code + Filebrowser
7. **11:30** Proof: Alle Dienste laufen (URLs/Ports)
8. **13:30** Ausblick: **Video 02 = Migration Pi3 → Pi4 (Cutover)** + Checkliste

---

4) Sprechertext (fertig zum Ablesen)

Intro (0:00 – 0:45)

„Willkommen im **Digitalcraft WikiCore-Universum**. Heute bauen wir den **CoreStack** auf einem Raspberry Pi 4 mit 2GB RAM – modern, wartbar und reproduzierbar: Docker + Compose, dazu n8n, Gitea, VS Code Server, Filebrowser und unser Kurs-Wiki **Digitalcraft WikiCore**."

On-Screen: Overlay „Digitalcraft WikiCore – CoreStack v1".

Voraussetzung (0:45 – 1:15)

„Wichtig: Das Betriebssystem läuft bei mir **direkt auf der 1TB-SSD**. Das SSD-Setup habe ich bereits in einem eigenen Video gezeigt – heute starten wir genau dort: System ist da, jetzt kommt die Plattform."

On-Screen: `df -h` + kurzer Blick in `lsblk`.

Digitalcraft-Standard (1:15 – 2:30)

„Bevor wir irgendwas installieren, legen wir unseren **Digitalcraft-Standard** fest: Alles, was Docker betrifft, liegt sauber unter `/srv/docker/corestack`. So bleibt das System nachvollziehbar – und später ist Migration oder Backup kein Ratespiel."

Docker Installation (2:30 – 7:00)

„Jetzt kommt das Fundament: Docker Engine und das Compose-Plugin. Profi-Hinweis: 64-bit ist langfristig die bessere Basis. Ich zeige euch kurz den Check – danach installieren wir Docker sauber."

CoreStack Deploy (7:00 – 11:30)

„Jetzt wird's spannend: Wir starten unseren ersten Digitalcraft CoreStack. Ziel ist nicht Overkill, sondern: **alle Dienste laufen stabil**, Daten sind persistent, und wir haben eine Plattform, die wir später migrieren und erweitern können."

Proof & Ausblick (11:30 – Ende)

„Wenn ihr das hier seht, habt ihr eine moderne Self-Hosting-Plattform auf dem Pi – komplett auf SSD, sauber als Stack. Im nächsten Video kommt der kritische Part: **Migration Pi 3 → Pi 4**, inklusive Cutover und Checks. Link zur Checkliste und zum Compose-Pack findet ihr in der Beschreibung."

---

5) Terminal-Cheatsheet (Copy & Paste)

5.1 Architektur-Check (64-bit?)

```
getconf LONG_BIT
```

`getconf LONG_BIT` Zeigt dir kurz gesagt, ob dein laufendes Linux-System 32-Bit (Ausgabe: 32) oder 64-Bit (Ausgabe: 64) ist. In meinem fall hier ist es ein 64 bit Linux System also die ausgabe ist 64.

```
dpkg --print-architecture
# erwarted arm64
```

`dpkg –print-architecture` gibt die Paket-Architektur deines Debian/Ubuntu-Systems aus – also für welche CPU-Plattform dein System Software installiert (z. B. welche .deb-Pakete passen). Typische Ausgaben:

- amd64 = 64-Bit PC/Server (Intel/AMD)
- arm64 = 64-Bit ARM (z. B. Raspberry Pi OS 64-Bit)
- armhf = 32-Bit ARM (z. B. Raspberry Pi OS 32-Bit)
- i386 = 32-Bit PC (selten heute)

```
df -h
```

`df -h` zeigt dir übersichtlich den freien und belegten Speicherplatz deiner Laufwerke/Partitionen an.

Auf meinem System ist sie volgendermaßen:

Filesystem	Size	Used	Avail	Use%	Mounted on		
udev	658M	0		658M		0%	/dev
tmpfs	369M	9.0M		360M		3%	/run
/dev/sda2	939G	3.7G		898G		1%	/
tmpfs	923M	0		923M		0%	/dev/shm
tmpfs	5.0M	16K		5.0M		1%	/run/lock
tmpfs	1.0M	0		1.0M		0%	/run/credentials/systemd-journald.service
tmpfs	923M	0		923M		0%	/tmp
/dev/sda1	510M	74M		437M		5%	/boot/firmware
tmpfs	1.0M	0		1.0M		0%	/run/credentials/getty@tty1.service
tmpfs	1.0M	0		1.0M		0%	/run/credentials/serial-getty@ttyS0.service
tmpfs	185M	8.0K		185M		1%	/run/user/1000

```
lsblk
```

NAME	MAJ:MIN	RM	SIZE	RO	TYPE	MOUNTPOINTS
loop0	7:0	0	1.8G	0	loop	
sda	8:0	0	953.9G	0	disk	
├─sda1	8:1	0	512M	0	part	/boot/firmware
└─sda2	8:2	0	953.4G	0	part	/
zram0	254:0	0	1.8G	0	disk	[SWAP]

`lsblk` zeigt dir eine übersichtliche Liste aller Laufwerke und Partitionen (z. B. SSD, HDD, USB-Stick) und wie sie eingebunden sind.

Du siehst typischerweise:

- Gerätenamen wie sda, nvme0n1, mmcblk0
- Partitionen wie sda1, sda2
- Größe, Typ (Disk/Part), Dateisystem (je nach Option)
- Mountpoints (z. B. /, /mnt/data)

Praktisch, um schnell zu prüfen: Welche Platte ist welche und wo hängt sie im System.

`Profi-Hinweis: Docker weist darauf hin, dass Raspberry Pi OS 32-bit (armhf) in zukünftigen Major-Versionen nicht mehr voll unterstützt wird – 64-bit ist langfristig die bessere Basis.`

Hinweis: Offizielle Docker-Doku (Raspberry Pi OS / Debian):

- Docker auf Raspberry Pi OS
- Docker auf Debian

5.2 System-Update

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

`sudo apt update && sudo apt full-upgrade -y` Aktualisiert zuerst die Paketlisten (apt update) und führt danach ein vollständiges System-Upgrade durch (apt full-upgrade). Dabei dürfen – falls nötig – auch Pakete neu installiert oder entfernt werden, um Abhängigkeiten sauber aufzulösen. -y bestätigt alles automatisch. Das && sorgt dafür, dass das Upgrade nur startet, wenn

das Update erfolgreich war.

`sudo reboot` Startet den Rechner neu (oft sinnvoll nach Kernel-/System-Updates).

5.3 Digitalcraft-Ordnerstruktur

```
sudo apt install mc
```

`sudo apt install mc` installiert das Programm Midnight Commander (mc).
`sudo` → Installation mit Admin-Rechten
`apt install` → Paket installieren
`mc` → Midnight Commander, ein textbasierter Dateimanager (zweispaltig), mit dem du Dateien/Ordner im Terminal bequem kopieren, verschieben, bearbeiten und durchsuchen kannst.

```
sudo mkdir -p /srv/docker/corestack/{postgres,n8n,dokuwiki,gitea,codeserver,filebrowser,shared}
```

`sudo mkdir -p /srv/docker/corestack/{postgres,n8n,dokuwiki,gitea,codeserver,filebrowser,shared}` erstellt eine Ordnerstruktur für deine Docker-Services.
`sudo` → mit Admin-Rechten ausführen
`mkdir` → Ordner anlegen
`-p` → legt auch fehlende Zwischenordner an und meckert nicht, wenn Ordner schon existieren
`/srv/docker/corestack/` → Basisordner
`{…}` → Kurzschreibweise (Brace Expansion): daraus werden mehrere Ordnernamen erzeugt

Ergebnis sind diese Ordner:

- /srv/docker/corestack/postgres
- /srv/docker/corestack/n8n
- /srv/docker/corestack/dokuwiki
- /srv/docker/corestack/gitea
- /srv/docker/corestack/codeserver
- /srv/docker/corestack/filebrowser
- /srv/docker/corestack/shared

Kurz: Ein Befehl, der dir sauber die Verzeichnisse für alle Container-Daten vorbereitet.

```
sudo chown -R $USER:$USER /srv/docker/corestack
```

`sudo chown -R $USER:$USER /srv/docker/corestack` setzt den Besitzer des Ordners /srv/docker/corestack (und aller Unterordner) auf deinen aktuellen Benutzer.
`sudo` → mit Admin-Rechten (sonst darfst du Besitzer nicht ändern)
`chown` → “change owner” (Besitzer ändern)
`-R` → rekursiv: betrifft alle Dateien und Unterordner
`$USER:$USER` → Benutzer:Gruppe = dein aktuell eingeloggter Nutzer (hier master:master)
`/srv/docker/corestack` → Zielordner

Zweck: Du kannst danach die Docker-Stack-Dateien und Configs in diesem Ordner ohne sudo bearbeiten.

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

`cd /srv/docker/corestack` wechselt im Terminal in das Verzeichnis `/srv/docker/corestack`.
`cd` = “change directory” (Verzeichnis wechseln)

Danach arbeitest du direkt in diesem Ordner (z. B. um dort docker-compose.yml zu bearbeiten oder Befehle auszuführen).

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

Zugriff im Browser (Self-signed Zertifikat ist normal):

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

Status prüfen:

```
systemctl status cockpit.socket --no-pager
ss -tulpn | grep 9090 || true
```

5.4 Docker + Compose (offizieller Weg, Debian-Style)

```
sudo apt-get install -y ca-certificates curl gnupg
sudo install -m 0755 -d /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/debian/gpg | sudo gpg --
dearmor -o /etc/apt/keyrings/docker.gpg
```

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

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

Docker-Gruppe (danach neu anmelden oder `newgrp`):

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

---

6) CoreStack Dateien (.env + docker-compose.yml)

6.1 .env (Pfad: /srv/docker/corestack/.env)

```
TZ=Europe/Berlin

# n8n: unbedingt setzen, damit Credentials stabil bleiben (auch bei
Migration)
N8N_ENCRYPTION_KEY=CHANGE_ME_TO_A_LONG_RANDOM_KEY

# Postgres
POSTGRES_DB=n8n
POSTGRES_USER=n8n
POSTGRES_PASSWORD=CHANGE_ME_STRONG

# Ports (Starter: bewusst einfach – Reverse Proxy kommt später als Pro-
Modul)
PORT_DOKUWIKI=8080
PORT_N8N=5678
PORT_GITEA=3000
PORT_GITEA_SSH=2222
PORT_CODESERVER=8443
PORT_FILEBROWSER=8081
```

n8n Hinweis: Environment Variables / Deployment: n8n – Deployment Env Vars

6.2 docker-compose.yml (Pfad: /srv/docker/corestack/docker-compose.yml)

```
services:
  postgres:
    image: postgres:16
    container_name: dc-postgres
    environment:
      - TZ=${TZ}
      - POSTGRES_DB=${POSTGRES_DB}
      - POSTGRES_USER=${POSTGRES_USER}
      - POSTGRES_PASSWORD=${POSTGRES_PASSWORD}
    volumes:
      - ./postgres/data:/var/lib/postgresql/data
    restart: unless-stopped

  n8n:
    image: n8nio/n8n:latest
    container_name: dc-n8n
    environment:
      - TZ=${TZ}
      - DB_TYPE=postgresdb
      - DB_POSTGRESDB_HOST=postgres
      - DB_POSTGRESDB_DATABASE=${POSTGRES_DB}
      - DB_POSTGRESDB_USER=${POSTGRES_USER}
      - DB_POSTGRESDB_PASSWORD=${POSTGRES_PASSWORD}
      - N8N_ENCRYPTION_KEY=${N8N_ENCRYPTION_KEY}
    ports:
      - "${PORT_N8N}:5678"
    volumes:
      - ./n8n/data:/home/node/.n8n
    depends_on:
      - postgres
    restart: unless-stopped

  dokuwiki:
    image: lscr.io/linuxserver/dokuwiki:latest
    container_name: dc-wikicore
    environment:
      - TZ=${TZ}
      - PUID=1000
      - PGID=1000
    volumes:
      - ./dokuwiki/config:/config
    ports:
      - "${PORT_DOKUWIKI}:80"
```

```
    restart: unless-stopped

  gitea:
    image: gitea/gitea:latest
    container_name: dc-gitea
    environment:
      - TZ=${TZ}
    volumes:
      - ./gitea/data:/data
    ports:
      - "${PORT_GITEA}:3000"
      - "${PORT_GITEA_SSH}:22"
    restart: unless-stopped

  codeserver:
    image: lscr.io/linuxserver/code-server:latest
    container_name: dc-codeserver
    environment:
      - TZ=${TZ}
      - PUID=1000
      - PGID=1000
    volumes:
      - ./codeserver/config:/config
      - ./shared:/workspace
    ports:
      - "${PORT_CODESERVER}:8443"
    restart: unless-stopped

  filebrowser:
    image: filebrowser/filebrowser:latest
    container_name: dc-filebrowser
    volumes:
      - ./shared:/srv
      - ./filebrowser/database:/database
      - ./filebrowser/config:/config
    ports:
      - "${PORT_FILEBROWSER}:80"
    restart: unless-stopped
```

Image-Dokus (optional, fürs Quellen-Kapitel):

- DokuWiki (LinuxServer): Docker DokuWiki – LinuxServer
- code-server (LinuxServer): Docker code-server – LinuxServer
- Gitea (Docker): Gitea – Install with Docker

---

7) Stack starten + prüfen

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

Logs (bei Bedarf):

```
docker logs -f dc-wikicore
docker logs -f dc-n8n
docker logs -f dc-gitea
```

---

8) Dienste testen (Browser)

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

- **WikiCore:**

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

- **n8n:**

  ```
  http://<PI-IP>:5678
  ```

- **Gitea:**

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

- **VS Code (code-server):**

  ```
  https://<PI-IP>:8443
  ```

- **Filebrowser:**

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

---

9) YouTube-Beschreibung (fertig zum Einfügen)

Digitalcraft CoreStack (Pi 4 / 2GB / 1TB SSD):
Heute bauen wir eine moderne Self-Hosting-Plattform mit Docker & Compose – inklusive **n8n**, **Gitea**, **VS Code Server**, **Filebrowser** und **DokuWiki als Digitalcraft WikiCore**.

Ergebnis: Alles läuft als Stack, Daten sind persistent, Struktur ist sauber.
Nächstes Video: **Migration Pi 3 → Pi 4 (Cutover + Checks)**

Downloads (kostenlos): CoreStack-Compose + Checkliste + WikiCore-Startseite → Link in der Beschreibung.

Chapters:
0:00 Intro … (Kapitel einfügen)

---

10) Nächster Schritt (Video 02 Teaser)

Nächstes Video: Migration Pi 3 → Pi 4

1. Freeze (Dienste stoppen)
2. Backup (Volumes/Configs)
3. Transfer (rsync)
4. Restore (Pi 4)
5. Funktionstest + Cutover

Ziel: Umstieg ohne Chaos – nachvollziehbar, wiederholbar, Digitalcraft-Style.
