GRO-MATE

Dokumentation

Inhaltsverzeichnis

6. Sicherheit
    6.1 Firewall-Regeln (UFW / nftables)
    6.2 Secrets Management (ENV, Docker, Mailcow)
    6.3 Zertifikate & TLS (Let's Encrypt / Reverse Proxy)
    6.4 Benutzer- und API-Authentifizierung
    6.5 Monitoring & Alerts (aktueller Stand / Planung)

7. Reverse Proxy / SSL
    7.1 Eingesetzter Reverse Proxy (Nginx Proxy Manager)
    7.2 Domain- und DNS-Konfiguration
    7.3 SSL/TLS-Zertifikate (Auto-Refresh, Let's Encrypt)
    7.4 Routing-Regeln & Proxy-Hosts

8. Deployment & Betrieb
    8.1 Starten und Stoppen aller Services
    8.2 Logs prüfen und auswerten
    8.3 Updates einspielen (System, Docker, Images)
    8.4 Backup & Restore
    8.5 Fehlerbehebung (Troubleshooting Guide)

9. Monitoring
    9.1 System-Metriken (CPU, RAM, Disk, Load)
    9.2 Container-Healthchecks
    9.3 Optional Setup: Prometheus / Grafana

10. Anhänge
    A. Vollständige `docker-compose.yml` (bereinigte Version)
    B. `.env`-Dateien mit Dummywerten
    C. Systemd-Service-Dateien (falls eingeführt)
    D. Backup-Skripte
    E. Netzwerkdiagramm (PDF/PNG)

1. Einleitung

1.1 Zweck der Dokumentation

Diese Dokumentation beschreibt die aktuelle Infrastruktur des Projekts GroceryMate inklusive des angebundenen Mailcow-Mailservers auf einem Hetzner vServer.
Ziele der Dokumentation sind:

- Sicherstellung eines nachvollziehbaren Betriebs (Runbooks für Start/Stop/Updates)
- Unterstützung bei Fehleranalyse und Troubleshooting
- Onboarding neuer Administratoren oder Entwickler
- Grundlage für spätere Erweiterungen (Monitoring, Backup-Automatisierung, Keycloak-Integration, etc.)

1.2 Übersicht der Infrastruktur

Die produktive Umgebung besteht aus folgenden Hauptkomponenten:

- Hetzner vServer (KVM-VM, Ubuntu 24.04.3 LTS, 2 vCPU, 4 GB RAM, ~38 GB SSD)
- Docker Engine + Docker Compose als Containerplattform
- GroceryMate-Stack im Verzeichnis `/home/jeff/grocery-mate`
    - Frontend: Vue 3 + Vite + Tailwind, ausgeliefert über Nginx im Container `grocery_frontend`
    - Backend: FastAPI + Gunicorn/Uvicorn im Container `grocery_backend`
    - Datenbank: PostgreSQL 16 im Container `grocery_postgres`

    - Datenbank-Admin: pgAdmin im Container `pgadmin`
    - Reverse Proxy: Nginx Proxy Manager im Container `nginx_proxy_manager`
- Mail-Infrastruktur mit Mailcow im Verzeichnis `/opt/mailcow-dockerized`
    - Mehrere Container (Postfix, Dovecot, Rspamd, SOGo usw.)
    - Eigene Daten-Volumes und eigenes Docker-Netzwerk
- Öffentlich erreichbare Domains (über Nginx Proxy Manager):
    - `gro-mate.tech` / `www.gro-mate.tech` – GroceryMate Frontend
    - `api.gro-mate.tech` – GroceryMate Backend (REST API)
    - `pg.gro-mate.tech` – pgAdmin (admin-only)
    - `mail.gro-mate.tech` – Mailcow Weboberfläche
    - `id.gro-mate.tech` – vorbereitete Keycloak-Instanz

1.3 Zielgruppe / Verantwortlichkeiten

Zielgruppe

- Systemadministratoren, die den Server betreiben und warten
- Entwickler*innen, die GroceryMate weiterentwickeln
- Prüfer / Dozenten, die die Architektur nachvollziehen möchten

Verantwortlichkeiten (Stand jetzt)

- Systemadministration & Deployment:
  Jeff (Benutzer `jeff`, Mitglied der Gruppen `sudo` und `docker`)

- Anwendungsentwicklung (Frontend/Backend):
  Ebenfalls Jeff (lokale Entwicklung auf Windows 11 Pro, Deployment via SSH)

- Mail-Infrastruktur / Mailcow:
  Technisch ebenfalls Jeff, mit Updates über das Mailcow-`update.sh`-Skript

2. Server Umgebung

2.1 Server-Spezifikationen (CPU, RAM, Storage)

- Provider: Hetzner
- Hardware-Modell: `vServer`
- Virtualisierung: KVM
- vCPU: 2 vCPUs
    - Typ: Intel Xeon (Skylake, IBRS, ohne TSX)
- RAM: 4 GB (ca. 3,7 GiB nutzbar)
- Storage:
    - Gerät: `/dev/sda` (≈ 38,1 GB SSD)
    - Partitionen:
        - `/dev/sda1` – 37,9 GB, eingebunden als `/`
        - `/dev/sda15` – 256 MB, eingebunden als `/boot/efi`
- Plattennutzung (Auszug):
    - `/dev/sda1` – 38G, davon 29G genutzt, ca. 6,9G frei (~81 % Belegung)

2.2 Betriebssystem (Linux Distribution, Version)

- Distribution: Ubuntu 24.04.3 LTS (Noble Numbat)
- Kernel: `Linux 6.8.0-87-generic`

- Architektur: x86-64
- Hostname: `jeff-4gb-nbg1-15`
- Firmware/Hypervisor: KVM, Hetzner

2.3 Netzwerk (IP, DNS, Firewall, Ports)

Netzwerkinterfaces

- `lo` – Loopback (127.0.0.1, ::1)
- `eth0` – Hauptinterface mit öffentlicher IP
    - IPv4: `91.99.21.21/32` (dynamisch vom Provider)
    - IPv6 (global): `2a01:4f8:1c1a:ca2e::1/64`
    - IPv6 (link-local): `fe80::9000:6ff:fe9f:f487/64`
- Docker/Bridge-Interfaces:
    - `docker0` – Standard-Docker-Bridge (`172.17.0.1/16`)
    - `br-6d7f2bca4e18` – `grocery-mate_devnet` (`172.18.0.1/16`)
    - `br-mailcow` – Mailcow-Bridge (`172.22.1.1/24`)

DNS

- DNS-Server werden vom Hetzner-Hosting gestellt (Standard-Konfiguration).
- Öffentliche Domains (`gro-mate.tech`, `mail.gro-mate.tech` etc.) zeigen via DNS auf die öffentliche IPv4 des Servers (91.99.21.21).

Firewall (UFW + nftables)

- `ufw` ist aktiv, Default-Regeln:
    - Eingehend: `deny`
    - Ausgehend: `allow`
- Wichtige freigegebene Ports (IPv4 + IPv6):
    - 22/tcp – SSH (OpenSSH)
    - 80/tcp – HTTP (Nginx Proxy Manager, Mailcow)
    - 443/tcp – HTTPS (Nginx Proxy Manager, Mailcow)
    - 81/tcp – NPM Admin-GUI (über Host-Port erreichbar)
    - 8082/tcp – pgAdmin
    - 8000/tcp – Backend/Reservierung (wird über NPM nach außen geroutet)
    - 8080/tcp – (Keycloak / interne Dienste)
    - 8181/tcp – (reserviert / historisch)
    - 5174/tcp – Vite-Dev-Server (für frühere lokale Tests, in Prod nicht aktiv)
    - Mailcow-Ports:
        - 25/tcp – SMTP (eingehende Mails)
        - 465/tcp – SMTPS
        - 587/tcp – SMTP Submission
        - 110/tcp – POP3
        - 995/tcp – POP3S
        - 143/tcp – IMAP

        - 993/tcp – IMAPS

2.4 Zugriffsrechte & SSH-Konfiguration

- Hauptbenutzer: `jeff`
    - UID/GID: 1000
    - Gruppen: `jeff`, `sudo`, `users`, `docker`
    - `jeff` kann Docker-Kommandos ohne `sudo` ausführen.
- SSH-Zugriff:
    - Dienst: OpenSSH
    - Port: 22/tcp (von überall erlaubt, durch UFW)
    - Authentifizierung: Passwort (Stand) – es wird empfohlen, auf SSH-Keys umzustellen.
- Projektpfade:
    - GroceryMate: `/home/jeff/grocery-mate`
    - Mailcow: `/opt/mailcow-dockerized`
    - NPM-Daten & Zertifikate: `/srv/dockerdata/nginx/{data,letsencrypt}`

3. Docker Umgebung

3.1 Installierte Software (Docker Engine, Docker Compose)

- Docker Engine:
  - Version: `Docker version 29.0.1, build eedd969`
- Docker Compose (plugin):
  - Version: `Docker Compose version v2.40.3`
- Sonstige relevante Software:
  - Python: `Python 3.12.3` (für lokale Skripte)
  - NodeJS: über `node:18-slim` im Frontend-Dockerfile für den Build-Prozess

3.2 Verzeichnisstruktur

Wesentliche Verzeichnisse auf dem Server:

```
/home/jeff/grocery-mate                 # Projekt-Root
GroceryMate

├── backend/                            # FastAPI-Backend
inkl. Dockerfile und .env

├── frontend/                           # Vue 3 Frontend inkl.
Dockerfile, nginx.conf und .env

├── data/                               # PostgreSQL-Daten
(ältere lokale Variante)

├── docker-compose.yml                  # Compose-Stack für
GroceryMate + Nginx Proxy Manager
```

```
├── pgadmin_servers.json              # Serverdefinition für
pgAdmin
└── .vscode/, .git/, .gitignore       # Editor- und
Versionskontroll-Konfiguration


/opt/mailcow-dockerized               # Mailcow-Stack
(separates Repository)
├── docker-compose.yml                # Compose-Definition
für Mailcow
├── mailcow.conf                      # Konfigurationsdatei
(Domains etc.)
└── data/, helper-scripts/, update.sh, ...


/srv/dockerdata/nginx
├── data/                             # NPM-Konfiguration,
Proxy-Hosts, Logs, DB etc.
└── letsencrypt/                      # Zertifikate /
ACME-Daten für NPM
```

3.3 Nutzer & Berechtigungen (docker group)

- Der Benutzer `jeff` ist Mitglied der Gruppe `docker`:
    - `id` → `groups=1000(jeff),27(sudo),100(users),110(docker)`
- Damit kann `jeff` folgende Befehle ohne sudo ausführen:
    - `docker ps`, `docker images`, `docker compose up -d`, `docker compose logs` usw.
- Sicherheitsimplikation: Mitgliedschaft in der Gruppe `docker` entspricht faktisch Root-Rechten, da Container Zugriff auf den Host erlangen können.
  → Zugriff auf den Benutzer `jeff` muss entsprechend geschützt werden (starkes Passwort, SSH-Hardening).

3.4 Backup- und Update-Konzept

Ist-Stand (Stand jetzt)

- Es existieren noch keine dedizierten Cronjobs für:
    - Datenbank-Backups von PostgreSQL
    - Sicherung der Docker-Volumes (GroceryMate, Mailcow, NPM)
- Systemweite Cronjobs (Standard):
    - `/etc/cron.daily`: `apt-compat`, `logrotate`, `man-db`, `sysstat` usw.
    - `/etc/cron.d/certbot`: Zertifikats-Erneuerung für Mailcow (ACME)
- Mailcow:
    - Aktualisierung über `/opt/mailcow-dockerized/update.sh` (manuell auszuführen, am besten nach Backup).

Empfohlenes (geplantes) Konzept

- PostgreSQL (GroceryMate):
    - Regelmäßiger `pg_dump` (z.B. täglich) des `grocery_db` in ein Backup-Verzeichnis (`/backup/postgres/`).
    - Optional: zusätzliches Backup der Volume-Daten `grocery-mate_postgres_data`.
- Mailcow:
    - Verwendung der offiziellen Mailcow-Backup-Skripte (z.B. `helper-scripts/backup_and_restore.sh`).
- Konfigurations-Backups:
    - Regelmäßige Sicherung von:
        - `docker-compose.yml` (GroceryMate + Mailcow)
        - `.env`-Dateien (mit Dummywerten für externe Doku)
        - `mailcow.conf`
        - NPM-Daten (`/srv/dockerdata/nginx/data`).
- Update-Prozess:
    - GroceryMate:
        - `docker compose pull` (bzw. `build`) und `docker compose up -d`
    - Mailcow:
        - `./update.sh` gemäß Mailcow-Dokumentation, nur nach aktuellem Backup.

4. Container Infrastruktur

4.1 Übersicht aller Services/Container

Aktuell laufende Container (`docker ps`):

- `grocery_frontend` – Image `grocery-mate-frontend` – Vue/Nginx-Frontend, Port 80 (intern)
- `grocery_backend` – Image `grocery-mate-backend` – FastAPI-Backend, Port 8000 (intern)
- `pgadmin` – Image `dpage/pgadmin4:latest` – PostgreSQL-Admin-UI, Host-Port 8082 → Container 80
- `grocery_postgres` – Image `postgres:16` – relationale Datenbank, Port 5432 (intern)
- `nginx_proxy_manager` – Image `jc21/nginx-proxy-manager:latest` – Reverse Proxy, Ports 80/81/443

Zusätzlich (Mailcow, aus separatem Compose-Stack):

- Mehrere Container mit Images aus `ghcr.io/mailcow/...`:
    - `nginx-mailcow`, `postfix`, `dovecot`, `rspamd`, `sogo`, `acme` usw.
- Keycloak:
    - Image `quay.io/keycloak/keycloak:latest` ist vorhanden, aber aktuell offenbar nicht als dauerhafter Container gestartet (nur vorbereitet).

4.2 Netzwerk-Topologie (Docker Networks)

Vorhandene Docker-Netzwerke:

- `grocery-mate_devnet` (Bridge)
    - Wird automatisch von `docker-compose.yml` im Projekt `grocery-mate` erstellt.
    - Enthält:
        - `grocery_frontend`
        - `grocery_backend`
        - `grocery_postgres`
        - `pgadmin`
        - `nginx_proxy_manager`
- `mailcowdockerized_mailcow-network` (Bridge)
    - Wird von Mailcow-Compose definiert.
    - Enthält alle Mailcow-Container (MTA, IMAP, Web, ACME usw.).
- `bridge`, `host`, `none`
    - Standard-Docker-Netzwerke.

Logische Verbindung

- Externer Traffic → Host-Port 80/443 → Container `nginx_proxy_manager` → interne Weiterleitung:

    - `gro-mate.tech` → `grocery_frontend:80` im `grocery-mate_devnet`
    - `api.gro-mate.tech` → `grocery_backend:8000`
    - `pg.gro-mate.tech` → `pgadmin:80`
    - `mail.gro-mate.tech` → `nginx-mailcow:8443` im `mailcowdockerized_mailcow-network` (über Peer-Routing/Ports)
- Backend ↔ Datenbank:
    - `grocery_backend` verbindet sich via `DB_HOST=postgres` (Service-Name) mit `grocery_postgres:5432` im selben Netzwerk.

4.3 Volumes & Persistente Daten

GroceryMate-Stack:

- Named Volumes:
    - `grocery-mate_postgres_data` – Daten von PostgreSQL 16
    - `grocery-mate_pgadmin_data` – Konfiguration und interne Daten von pgAdmin
- Bind-Mounts:
    - `./pgadmin_servers.json:/pgadmin4/servers.json:ro` – vordefinierte Serverliste
- Historische Daten:
    - `./data/` im Projekt-Root enthält eine PostgreSQL-Datadir-Struktur (älterer Stand / lokale Datenmigration).

Mailcow-Stack:

- Diverse Volumes mit Präfix `mailcowdockerized_...`, u.a.:
    - `mailcowdockerized_mysql-vol-1` – MySQL/MariaDB-Datenbank
    - `mailcowdockerized_redis-vol-1` – Redis
    - `mailcowdockerized_vmail-vol-1`, `vmail-index-vol-1` – Maildaten
    - `mailcowdockerized_rspamd-vol-1`, `clamd-db-vol-1` – Antispam/Antivirus
- ACME/Zertifikatsdaten:
    - Mailcow nutzt eigene ACME-Container (`acme`) mit Volumes.

Nginx Proxy Manager:

- Volumes:
    - `npm_data` – Konfigurationsdaten, UI-Einstellungen, DB
    - `npm_letsencrypt` – Let's-Encrypt-Zertifikate

Alle Volumes liegen physisch unter `/var/lib/docker/volumes/` auf `/dev/sda1`.

4.4 Logging (Docker Logs, zentrale Logfiles)

- Container-Logs:
    - Standardmäßig über `docker logs <container>` abrufbar:
        - `docker logs -f grocery_backend`
        - `docker logs -f grocery_frontend`
        - `docker logs -f nginx_proxy_manager`
        - `docker logs -f pgadmin`
        - `docker logs -f grocery_postgres`
- Nginx Proxy Manager:
    - Zusätzliche Logs im Volume `npm_data` (intern in SQLite/Dateien).
- Mailcow:
    - Eigene Logstruktur in `/opt/mailcow-dockerized/data/` und in den jeweiligen Container-Logs.
- System-Logs:
    - Standardmäßig unter `/var/log/` (z.B. `syslog`, `auth.log`), abrufbar mit `journalctl`.

5. Anwendungen (Container)

Im Folgenden werden die wichtigsten Container des GroceryMate-Stacks beschrieben. Mailcow wird als separater Anwendungs-Block betrachtet.

5.1 Service: PostgreSQL (grocery_postgres)

- Zweck:
  Relationale Datenbank für alle GroceryMate-Daten (Benutzer, Inventar, Rezepte, Einkaufslisten, Session-Daten etc.).

- Image / Version:
  `postgres:16`

- docker-compose.yaml Abschnitt (vereinfacht):

```
postgres:
  image: postgres:16
  container_name: grocery_postgres
  environment:
    POSTGRES_USER: grocery_user
    POSTGRES_PASSWORD: grocery_pass
    POSTGRES_DB: grocery_db
  volumes:
    - postgres_data:/var/lib/postgresql/data
  healthcheck:
```

```
    test: ["CMD-SHELL", "pg_isready -U grocery_user -d
grocery_db"]
    interval: 10s
    timeout: 5s
    retries: 5
  networks:
    - devnet
  restart: unless-stopped
```

- Umgebungsvariablen (.env / Compose):
  - `POSTGRES_USER=grocery_user`
  - `POSTGRES_PASSWORD=grocery_pass`
  - `POSTGRES_DB=grocery_db`
- Persistente Daten (Volumes):
  - Named Volume `postgres_data` → `/var/lib/postgresql/data` im Container
- Exponierte Ports:
  - Container-Port: 5432 (nur im `devnet` sichtbar, nicht direkt am Host gemappt)
- Start/Stop Befehle:

```
cd /home/jeff/grocery-mate
```

```
docker compose up -d postgres      # Start

docker compose stop postgres       # Stop

docker compose logs -f postgres    # Logs verfolgen
```

- Update-Prozess:

```
cd /home/jeff/grocery-mate

docker compose pull postgres       # Neues Image ziehen
(falls verwendet)

docker compose up -d postgres      # Container neu starten
```

5.2 Service: pgAdmin (pgadmin)

- Zweck:
  Weboberfläche zur Verwaltung und Analyse der PostgreSQL-Datenbank.

- Image / Version:
  `dpage/pgadmin4:latest`

- docker-compose.yaml Abschnitt (vereinfacht):

```
pgadmin:

  image: dpage/pgadmin4:latest

  container_name: pgadmin

  environment:

    PGADMIN_DEFAULT_EMAIL: admin@grocerymate.com
```

```
    PGADMIN_DEFAULT_PASSWORD: admin_password
  ports:
    - "8082:80"
  depends_on:
    postgres:
      condition: service_healthy
  volumes:
    - pgadmin_data:/var/lib/pgadmin
    - ./pgadmin_servers.json:/pgadmin4/servers.json:ro
  networks:
    - devnet
  restart: unless-stopped
```

- Umgebungsvariablen:
    - `PGADMIN_DEFAULT_EMAIL=admin@grocerymate.com`
    - `PGADMIN_DEFAULT_PASSWORD=admin_password`
- Persistente Daten:
    - Volume `pgadmin_data` → `/var/lib/pgadmin`
    - Read-only Bind-Mount `pgadmin_servers.json` → `/pgadmin4/servers.json`
- Exponierte Ports:

    - Host-Port `8082` → Container-Port `80`
    - Erreichbar über `http(s)://pg.gro-mate.tech` via Nginx Proxy Manager

- Start/Stop Befehle:

```
cd /home/jeff/grocery-mate

docker compose up -d pgadmin

docker compose stop pgadmin

docker compose logs -f pgadmin
```

- Update-Prozess:

```
cd /home/jeff/grocery-mate

docker compose pull pgadmin

docker compose up -d pgadmin
```

5.3 Service: Backend (grocery_backend)

- Zweck:
  REST-API auf Basis von FastAPI, inklusive Authentifizierung, Geschäftslogik, AI-Funktionen (Google Gemini) und Datenzugriff auf PostgreSQL.

- Image / Version:
  `grocery-mate-backend:latest` (lokal gebaut über Dockerfile)

- Dockerfile (wichtigste Punkte):

    - Multi-Stage Build:
        - Stage 1: `python:3.11-slim`, Installation von Build-Abhängigkeiten + `requirements.txt` in ein virtuelles Environment `/venv`
        - Stage 2: `python:3.11-slim`, Kopieren von `/venv` und des Ordners `app/`

    Startkommando:

```
CMD ["gunicorn", "app.main:app",
     "-k", "uvicorn.workers.UvicornWorker",
     "--bind", "0.0.0.0:8000",
     "--workers", "4"]
```

    -
- docker-compose.yaml Abschnitt (vereinfacht):

```
backend:
  build:
    context: ./backend
    dockerfile: Dockerfile
  container_name: grocery_backend
  command: >
    gunicorn app.main:app
    -k uvicorn.workers.UvicornWorker
    --bind 0.0.0.0:8000
```

```
    --workers 4
    --timeout 120
  env_file:
    - ./backend/.env
  environment:
    DB_HOST: postgres
    DB_USER: grocery_user
    DB_PASSWORD: grocery_pass
    DB_NAME: grocery_db
  expose:
    - "8000"
  depends_on:
    postgres:
      condition: service_healthy
  networks:
    - devnet
  restart: unless-stopped
```

- Umgebungsvariablen (`backend/.env` + Compose):
    - `DATABASE_URL=postgresql+psycopg2://grocery_user:grocery_pass@grocery_postgres:5432/grocery_db`

    - `GEMINI_API_KEY=***REDACTED***`
    - `DB_HOST=postgres`, `DB_USER=grocery_user`, `DB_PASSWORD=grocery_pass`, `DB_NAME=grocery_db`
- Persistente Daten:
    - Der Backend-Container selbst speichert keine persistente Nutzdaten, alle Daten liegen in PostgreSQL.
- Exponierte Ports:
    - Container-Port `8000` (REST-API)
    - Nicht direkt als Host-Port gemappt, sondern über Nginx Proxy Manager via `api.gro-mate.tech` erreichbar.
- Start/Stop Befehle:

```
cd /home/jeff/grocery-mate

docker compose up -d backend

docker compose stop backend

docker compose logs -f backend
```

- Update-Prozess:

```
cd /home/jeff/grocery-mate

docker compose build backend        # Backend neu bauen
(z.B. nach Code-Änderungen)

docker compose up -d backend        # Container mit neuem
Image starten
```

5.4 Service: Frontend (grocery_frontend)

- Zweck:
  Auslieferung des GroceryMate-Webfrontends (Vue 3 + Vite + Tailwind) über Nginx.

- Image / Version:
  `grocery-mate-frontend:latest` (lokal gebaut)

- Dockerfile (wichtigste Punkte):

  - Stage 1: `node:18-slim` – npm-Install (`npm ci`) + `npm run build`

  - Stage 2: `nginx:alpine` – Kopieren von `/app/dist` nach `/usr/share/nginx/html`, Verwendung eigener `nginx.conf`

- docker-compose.yaml Abschnitt (vereinfacht):

```
frontend:
  build:
    context: ./frontend
    dockerfile: Dockerfile
  container_name: grocery_frontend
  expose:
    - "80"
  networks:
    - devnet
  restart: unless-stopped
```

- Umgebungsvariablen (`frontend/.env`):
    - `VITE_BACKEND_URL=https://api.gro-mate.tech`
- Persistente Daten:
    - Keine Anwendungsspeicherung; nur statischer Build im Image.
- Exponierte Ports:
    - Container-Port `80`
    - Über Nginx Proxy Manager auf `gro-mate.tech` / `www.gro-mate.tech` gemappt.
- Start/Stop Befehle:

```
cd /home/jeff/grocery-mate

docker compose up -d frontend

docker compose stop frontend

docker compose logs -f frontend
```

- Update-Prozess:

```
cd /home/jeff/grocery-mate

docker compose build frontend      # neuen Frontend-Build
erstellen

docker compose up -d frontend      # Container mit neuem
Image starten
```

5.5 Service: Nginx Proxy Manager (nginx_proxy_manager)

- Zweck:
  Zentraler Reverse Proxy für alle HTTP/HTTPS-Dienste inklusive automatischer TLS-Zertifikate (Let's Encrypt) und komfortabler Web-GUI.

- Image / Version:
  `jc21/nginx-proxy-manager:latest`

- docker-compose.yaml Abschnitt (vereinfacht):

```
nginx-proxy-manager:
  image: jc21/nginx-proxy-manager:latest
  container_name: nginx_proxy_manager
  ports:
    - "80:80"
    - "81:81"
    - "443:443"
  volumes:
    - /srv/dockerdata/nginx/data:/data
    - /srv/dockerdata/nginx/letsencrypt:/etc/letsencrypt
  networks:
    - devnet
  restart: unless-stopped
```

- Umgebungsvariablen:
    - Standardkonfiguration des Images (DB in `/data/database.sqlite` o.ä.).
- Persistente Daten:
    - `/srv/dockerdata/nginx/data` – Proxy-Host-Konfigurationen, Benutzer, Logs
    - `/srv/dockerdata/nginx/letsencrypt` – TLS-Zertifikate / ACME-Daten
- Exponierte Ports:
    - `80/tcp` – HTTP-Eingang für alle Domains
    - `443/tcp` – HTTPS-Eingang
    - `81/tcp` – Admin-Weboberfläche von NPM
- Start/Stop Befehle:

```
cd /home/jeff/grocery-mate

docker compose up -d nginx-proxy-manager

docker compose stop nginx-proxy-manager

docker compose logs -f nginx-proxy-manager
```

- Update-Prozess:

```
cd /home/jeff/grocery-mate

docker compose pull nginx-proxy-manager

docker compose up -d nginx-proxy-manager
```

5.6 Service: Mailcow-Stack

- Zweck:
  Vollwertige Mailserverlösung für die Domain `gro-mate.tech` (SMTP, IMAP, Webmail, Spamfilter, Groupware).

- Basis:
  - Separates Git-Repository unter `/opt/mailcow-dockerized`
  - Docker Compose-Stack mit zahlreichen Containern (nginx-mailcow, postfix, dovecot, rspamd, sogo, acme, watchdog, redis, mariadb usw.)

- Konfiguration:
  - `mailcow.conf` – zentrale Konfigurationsdatei (Hostname, Domains, Optionen)
  - `.env` – liegt im Projektverzeichnis (nicht lesbar ohne Root-Rechte)

- Start/Stop/Update:

```
cd /opt/mailcow-dockerized

docker compose up -d              # Start des gesamten
Mailcow-Stacks

docker compose stop               # Stop

docker compose ps                 # Statusübersicht


# Update (nur nach aktuellem Backup!

sudo ./update.sh
```

6. Sicherheit

6.1 Firewall-Regeln (UFW / nftables)

Auf dem Server ist **UFW** als Firewall aktiviert und nutzt intern **nftables** als Backend.

- Standard-Policy:
    - Eingehend: `deny`
    - Ausgehend: `allow`
    - Weitergeleitet: `deny`
- Erlaubte eingehende Ports (IPv4 & IPv6):

Management & Infrastruktur

- `22/tcp` – SSH (Remote-Verwaltung)
- `81/tcp` – Nginx Proxy Manager Admin-GUI

Web / Anwendungen

- `80/tcp` – HTTP (Nginx Proxy Manager, Mailcow-HTTP, Let's Encrypt HTTP-01)
- `443/tcp` – HTTPS (Nginx Proxy Manager, Mailcow-HTTPS)
- `8082/tcp` – pgAdmin Weboberfläche
- `8000/tcp`, `8001/tcp`, `8080/tcp`, `8081/tcp`, `8181/tcp`, `5174/tcp` – Ports für Entwicklungs- und Anwendungscontainer (Backend, Keycloak, frühere Vite-Dev-Umgebungen)

Mailcow (Maildienste)

- `25/tcp` – SMTP (eingehende Mails)
- `465/tcp` – SMTPS
- `587/tcp` – SMTP Submission (Client-Authentifizierung)
- `110/tcp` – POP3
- `995/tcp` – POP3S
- `143/tcp` – IMAP
- `993/tcp` – IMAPS

Die eigentliche Paketfilterlogik wird über die UFW-nftables-Chains gesteuert (`ufw-before-input`, `ufw-user-input` usw.). Es existieren aktuell keine zusätzlichen, manuell gepflegten nftables-Regeln außerhalb von UFW.

6.2 Secrets Management (ENV, Docker, Mailcow)

Aktueller Stand

- Anwendungsspezifische Secrets (z.B. DB-Passwörter, API-Keys) werden in **`.env`-Dateien** und Docker-Compose-Umgebungsvariablen verwaltet:
  - `/home/jeff/grocery-mate/backend/.env`
  - `/home/jeff/grocery-mate/frontend/.env`
  - `/opt/mailcow-dockerized/.env` (Root-Rechte erforderlich)
- Diese Dateien sind über `.gitignore` vom Git-Repository ausgeschlossen und werden **nicht versioniert**.

- Nginx Proxy Manager speichert Zugangsdaten und Zertifikate verschlüsselt bzw. in internen Datenbanken im Volume `/srv/dockerdata/nginx/data`.

Risiken und Empfehlungen

- `.env`-Dateien liegen im Klartext auf dem Server und sind bei einem Account-Kompromiss auslesbar.
- Empfohlen:
    - Dateirechte einschränken (z.B. `chmod 600 backend/.env`).
    - SSH-Hardening (Schlüssel-Authentifizierung, kein Password-Login).
    - Mittel- bis langfristig: Einsatz von **Docker Secrets** oder einem externen Secret-Store (z.B. HashiCorp Vault, passwortgeschützte Konfigurationsdateien).

6.3 Zertifikate & TLS (Let's Encrypt / Reverse Proxy)

GroceryMate / Nginx Proxy Manager

- TLS-Termination erfolgt zentral im Container **`nginx_proxy_manager`**.
- Für jede Domain (z.B. `gro-mate.tech`, `api.gro-mate.tech`, `pg.gro-mate.tech`, `id.gro-mate.tech`) wird ein **Let's-Encrypt-Zertifikat** eingerichtet.
- Zertifikate werden im Volume `/srv/dockerdata/nginx/letsencrypt` gespeichert und automatisch erneuert (ACME-Client des Images).
- Die Kommunikation zwischen NPM und den Anwendungscontainern (Frontend, Backend, pgAdmin) erfolgt im internen Docker-Netzwerk **unverschlüsselt (HTTP)**, da TLS bereits am Reverse Proxy beendet wird.

Mailcow

- Mailcow betreibt eine eigene ACME-Integration über den Container `acme` (Image `ghcr.io/mailcow/acme`).
- Zertifikate werden für `mail.gro-mate.tech` und ggf. weitere Mail-Domains erzeugt.
- Diese Zertifikate werden von den Maildiensten (Postfix, Dovecot, SOGo) für SMTPS/IMAPS/HTTPS verwendet.
- Daten werden in Volumes mit Präfix `mailcowdockerized_...` abgelegt.

6.4 Benutzer- und API-Authentifizierung

System / Server

- Administration erfolgt über den Benutzer **`jeff`**:
    - Mitglied der Gruppen `sudo` und `docker`.
    - Volle Administrationsrechte auf Host- und Docker-Ebene.
- SSH-Authentifizierung erfolgt derzeit per Passwort.
  Empfehlung:
    - Umstellung auf **SSH-Key-Authentifizierung**.
    - Deaktivierung von Passwort-Login (`PasswordAuthentication no`).
    - Fail2ban / Rate-Limiting für SSH-Zugriffe.

Anwendung (GroceryMate)

- Das Backend (FastAPI) implementiert die Anwendungs-Authentifizierung (Login/Registrierung, Sessions/Tokens).

- relevanter Code: im Backend-Verzeichnis `app/` (z.B. `auth.py`, `routers`, `schemas`).
- Kommunikation:
    - Frontend ↔ Backend ausschließlich über **HTTPS** via `api.gro-mate.tech`.
    - Frontend-URL im Build: `VITE_BACKEND_URL=https://api.gro-mate.tech`.

Mail

- Mail-Benutzer (IMAP/SMTP/Webmail) werden in Mailcow verwaltet.
- Die Weboberfläche ist über `https://mail.gro-mate.tech` erreichbar (TLS, Login mit Mail-Accounts).

6.5 Monitoring & Alerts (aktueller Stand / geplant)

Aktueller Stand

- Systemweite Tools:
    - `sysstat` (Cronjobs vorhanden, z.B. `/etc/cron.d/sysstat`, `/etc/cron.daily/sysstat`).
    - Standard-Befehle wie `top`, `free -h`, `df -h` für Ad-hoc-Checks.
- Docker:
    - Healthcheck für PostgreSQL-Container (`pg_isready`).
    - Statusprüfung über `docker ps`, `docker logs`.
- Mailcow:

- Eigene Monitoring-/Watchdog-Mechanismen im Container `watchdog`.

Geplante/empfohlene Erweiterungen

- Einfache Uptime-Überwachung (z.B. UptimeRobot oder selbst gehosteter Uptime-Kontrolldienst).
- Einführung eines **zentralen Monitorings**:
  - Node Exporter (Host-Metriken)
  - cAdvisor (Container-Metriken)
  - Prometheus + Grafana (siehe Punkt 9.3)
- E-Mail-Alerts (z.B. bei niedrigem Plattenplatz, nicht laufenden Containern, TLS-Fehlern).

7. Reverse Proxy / SSL

7.1 Genutzter Reverse Proxy

- Eingesetzte Lösung: **Nginx Proxy Manager** (Image `jc21/nginx-proxy-manager:latest`).
- Läuft im Container `nginx_proxy_manager` im Netzwerk `grocery-mate_devnet`.
- Admin-Weboberfläche:
  - Host-Port: `81/tcp`
  - URL: `http(s)://<Server-IP>:81` bzw. via internem Admin-Zugriff über Domain (optional).

7.2 Domain- und DNS-Konfiguration

Alle relevanten Domains zeigen per DNS (A/AAAA-Records) auf die öffentliche IP des Hetzner-Servers:

- `gro-mate.tech` → GroceryMate Frontend (NPM → `grocery_frontend:80`)
- `www.gro-mate.tech` → wie oben (Redirect / Alias)
- `api.gro-mate.tech` → Backend FastAPI (NPM → `grocery_backend:8000`)
- `pg.gro-mate.tech` → pgAdmin (NPM → `pgadmin:80`)
- `mail.gro-mate.tech` → Mailcow-Weboberfläche / SOGo (NPM oder Mailcow-nginx; je nach finaler Konfiguration)
- `id.gro-mate.tech` → Keycloak (geplant, Image bereits vorhanden)

DNS-Änderungen werden beim Domain-Registrar/Provider gepflegt und sind nicht direkt Teil dieser Dokumentation.

7.3 SSL/TLS Zertifikate (Auto-Refresh)

- Für GroceryMate-Webdienste werden **Let's-Encrypt-Zertifikate** zentral über NPM verwaltet:
    - Erstellung direkt aus der NPM-GUI (HTTP/HTTPS-Challenge).
    - Automatische Erneuerung (ACME-Client im Container).
- Zertifikate liegen im Volume `/srv/dockerdata/nginx/letsencrypt`.
- Für Mailcow erfolgt die Zertifikatsverwaltung über den ACME-Container `acme`:
    - Konfiguration in `mailcow.conf` und in der Mailcow-Umgebung.

    - Zertifikate werden in `mailcowdockerized_*`-Volumes abgelegt.

- Voraussetzung für automatische Erneuerung:
    - Ports `80` und `443` müssen dauerhaft aus dem Internet erreichbar sein.
    - Keine konkurrierenden Webserver auf den gleichen Ports.

7.4 Routing-Regeln / Proxy-Hosts

In Nginx Proxy Manager sind für jede Domain sogenannte **Proxy Hosts** definiert, z.B.:

- `gro-mate.tech` → Weiterleitung auf `http://grocery_frontend:80`
- `api.gro-mate.tech` → Weiterleitung auf `http://grocery_backend:8000`
- `pg.gro-mate.tech` → Weiterleitung auf `http://pgadmin:80`
- `mail.gro-mate.tech` → Weiterleitung auf `https://nginx-mailcow:8443` (Mailcow-Stack)

Zusätzliche Einstellungen:

- **HTTP → HTTPS-Redirect** in NPM aktiviert (erzwingt verschlüsselte Verbindungen).
- Optionale **CORS- und Security-Header**:
    - Für `api.gro-mate.tech` sind CORS-Regeln notwendig, damit das Frontend auf die API zugreifen kann.
    - CORS kann entweder im Backend (FastAPI-Middleware) oder in NPM per Custom Header/Advanced Config konfiguriert werden.
- Zugriff auf Admin-Oberflächen (NPM, pgAdmin, Mailcow) sollte in der Praxis auf bestimmte IP-Bereiche oder Benutzer eingeschränkt werden.

8. Deployment & Betrieb

8.1 Starten / Stoppen aller Services

GroceryMate-Stack

Arbeitsverzeichnis: `/home/jeff/grocery-mate`

```
cd /home/jeff/grocery-mate

# Kompletten Stack starten
docker compose up -d

# Einzelne Services starten
docker compose up -d postgres backend frontend pgadmin
nginx-proxy-manager

# Stack stoppen
docker compose stop

# Einzelnen Service stoppen
docker compose stop backend
```

Mailcow-Stack

Arbeitsverzeichnis: `/opt/mailcow-dockerized`

```
cd /opt/mailcow-dockerized

# Kompletten Mailcow-Stack starten
docker compose up -d

# Stack stoppen
docker compose stop
```

Empfohlene Start-Reihenfolge nach einem Neustart:

1. Mailcow (falls Maildienste kritisch sind)
2. GroceryMate-Stack (Postgres → Backend → Frontend → NPM)

8.2 Logs prüfen

Docker-Container

```
cd /home/jeff/grocery-mate
docker compose logs -f backend
docker compose logs -f frontend
docker compose logs -f postgres
docker compose logs -f pgadmin
docker compose logs -f nginx-proxy-manager
```

Für Mailcow:

```
cd /opt/mailcow-dockerized
docker compose logs -f nginx-mailcow
docker compose logs -f postfix
docker compose logs -f dovecot
docker compose logs -f rspamd
```

System-Logs

- `journalctl -u docker.service` – Docker-Dienst
- `journalctl -u ssh` – SSH-Login-Versuche
- `journalctl -xe` – generelle Systemfehler
- Klassische Logs unter `/var/log/` (z.B. `syslog`, `auth.log`)

8.3 Updates einspielen (Images, Docker, OS)

Betriebssystem

```
sudo apt update
sudo apt upgrade
sudo reboot  # wenn Kernel/wichtige Pakete aktualisiert wurden
```

Docker Engine & Compose

- Werden in der Regel über das Paketmanagement bzw. die Docker-eigenen Repositories aktualisiert.

Nach Updates ggf. Docker neu starten:

```
sudo systemctl restart docker
```

GroceryMate-Stack

```
cd /home/jeff/grocery-mate

# Source-Code aktualisieren (Git)
git pull

# Neue Images bauen
docker compose build

# Stack mit neuen Images starten
docker compose up -d
```

Mailcow

```
cd /opt/mailcow-dockerized

# WICHTIG: vorher Backup!
sudo ./update.sh
```

8.4 Backup & Restore

PostgreSQL (GroceryMate)

Backup (Beispiel, manuell)

```
mkdir -p /home/jeff/backups/postgres
docker exec -t grocery_postgres \
  pg_dump -U grocery_user grocery_db \
  > /home/jeff/backups/postgres/grocery_db_$(date +%F).sql
```

Restore (Beispiel)

```
docker exec -i grocery_postgres \
  psql -U grocery_user -d grocery_db \
  < /home/jeff/backups/postgres/grocery_db_YYYY-MM-DD.sql
```

Docker-Volumes

- Backup per `tar`:

```
# Beispiel für NPM-Volumepfade:
sudo tar czf /home/jeff/backups/npm_data_$(date +%F).tar.gz \
  /srv/dockerdata/nginx/data /srv/dockerdata/nginx/letsencrypt
```

- Restore: Entpacken der Archive an die ursprünglichen Pfade (bei gestoppten Containern).

Mailcow

- Verwendung der offiziellen Backup-Skripte (z.B. `helper-scripts/backup_and_restore.sh`).

- Backups sollten auf ein externes Storage (z.B. Hetzner Storage Box, S3) synchronisiert werden.

8.5 Fehlerbehebung (Troubleshooting Guide)

Typischer Minimal-Workflow bei Störungen:

1. **Erreichbarkeit prüfen**
    - `ping gro-mate.tech`
    - `curl -I https://gro-mate.tech`
2. **Docker-Status prüfen**
    - `docker ps`
    - Sind alle wichtigen Container `Up`? Gibt es `Restarting`-Container?
3. **Logs checken**
    - `docker compose logs -f backend`
    - `docker compose logs -f nginx-proxy-manager`
    - Mailprobleme: `docker compose logs -f postfix dovecot rspamd`
4. **Ressourcen prüfen**
    - `free -h` – RAM
    - `df -h` – Plattenplatz (→ besonders, da `/dev/sda1` bereits ~80 % belegt ist)
    - `top` – CPU-Last

5. **Netzwerk & Firewall**
    - `sudo ufw status verbose`
    - Stimmt die Portfreigabe mit den NPM-Einstellungen überein?
6. **Let's-Encrypt-Probleme**
    - Sind Ports 80/443 erreichbar?
    - NPM-/Mailcow-ACME-Logs prüfen (`nginx_proxy_manager`, `acme`).

9. Monitoring

9.1 System-Metriken (CPU, RAM, Disk, Load)

Aktuell verfügbare Werkzeuge:

- `top` / `htop` (falls installiert) – Live-Ansicht CPU/RAM/Prozesse
- `free -h` – RAM-Nutzung
- `df -h` – Plattenbelegung
- `uptime` – Load Average
- `sysstat` (`sar`-Befehle) – historische Performance-Daten

Diese Tools werden direkt auf dem Host-System eingesetzt (SSH-Login).

9.2 Container-Healthchecks

- Für den PostgreSQL-Container ist ein Docker-Healthcheck definiert:

```
healthcheck:
  test: ["CMD-SHELL", "pg_isready -U grocery_user -d
grocery_db"]
  interval: 10s
  timeout: 5s
  retries: 5
```

- Über `docker ps` kann der Health-Status eingesehen werden (`healthy` / `unhealthy`).
- Für andere Container (Backend, Frontend, NPM) existieren derzeit keine expliziten Healthchecks; der Status ergibt sich aus Exit-Codes und Logs.
- Empfehlung:
    - Healthchecks für Backend (HTTP 200 auf `/health` o.ä.) und evtl. Frontend hinzufügen.
    - Nutzung eines Watchdogs, der bei `unhealthy`-Status automatisiert benachrichtigt oder restarts auslöst.

9.3 Optional: Prometheus / Grafana Setup

Aktuell **nicht implementiert**, aber empfohlenes Ziel-Setup:

- **Prometheus** – Sammeln von Metriken
- **Grafana** – Visualisierung und Dashboards
- **node_exporter** – Host-Metriken (CPU, RAM, Disk, Netzwerk)
- **cAdvisor** – Container-Metriken (CPU/RAM pro Container)
- Optional:
    - Mailcow-spezifische Metriken (Queue-Länge, Spam-Rate)

    - Backend-spezifische Metriken (Anzahl Requests, Antwortzeiten)

Die Komponenten könnten selbst in einem separaten Docker-Compose-Stack betrieben werden (z.B. `/home/jeff/monitoring/docker-compose.yml`).

10. Anhänge

A. Vollständige `docker-compose.yml`

- Datei: `/home/jeff/grocery-mate/docker-compose.yml`
- Inhalt:
    - Definition der Services `postgres`, `pgadmin`, `backend`, `frontend`, `nginx-proxy-manager`
    - Netzwerkdefinition `devnet`
    - Volume-Definitionen `postgres_data`, `pgadmin_data`
- Für die Dokumentation wird eine **bereinigte Version** verwendet:
    - Passwörter und sensible Werte anonymisiert (Dummywerte).

B. `.env` Dateien (mit Dummywerten)

- Backend: `/home/jeff/grocery-mate/backend/.env`
    - `DATABASE_URL=postgresql+psycopg2://grocery_user:***@grocery_postgres:5432/grocery_db`

    - `GEMINI_API_KEY=***`

- Frontend: `/home/jeff/grocery-mate/frontend/.env`
    - `VITE_BACKEND_URL=https://api.gro-mate.tech`

- Mailcow: `/opt/mailcow-dockerized/.env`
    - Diverse interne Variablen (Datenbank, Hostname, ACME-Config, etc.)

Für die Dokumentation sollten **Kopien mit Dummywerten** erstellt werden (z.B. `env.backend.example`, `env.frontend.example`).

C. Systemd Service Files (falls genutzt)

- Derzeit werden Docker-Stacks manuell über `docker compose` gestartet.
- Es existieren aktuell **keine** eigenen Systemd-Units wie `grocerymate.service`.
- Perspektivisch können Einheiten unter `/etc/systemd/system/` angelegt werden, z.B.:
    - `grocerymate.service` – führt `docker compose up -d` im Projektverzeichnis aus.
    - `mailcow.service` – analog für `/opt/mailcow-dockerized`.

D. Backup-Skripte

- Stand jetzt: Backups werden (falls vorhanden) manuell ausgeführt.
- Geplante Struktur:
    - `/usr/local/sbin/backup-grocerymate.sh`
    - `/usr/local/sbin/backup-mailcow.sh`
- Diese Skripte sollten:
    - Datenbanken (PostgreSQL, Mailcow-DB) sichern
    - Wichtige Volumes packen (`tar.gz`)
    - Backups auf externes Storage synchronisieren.

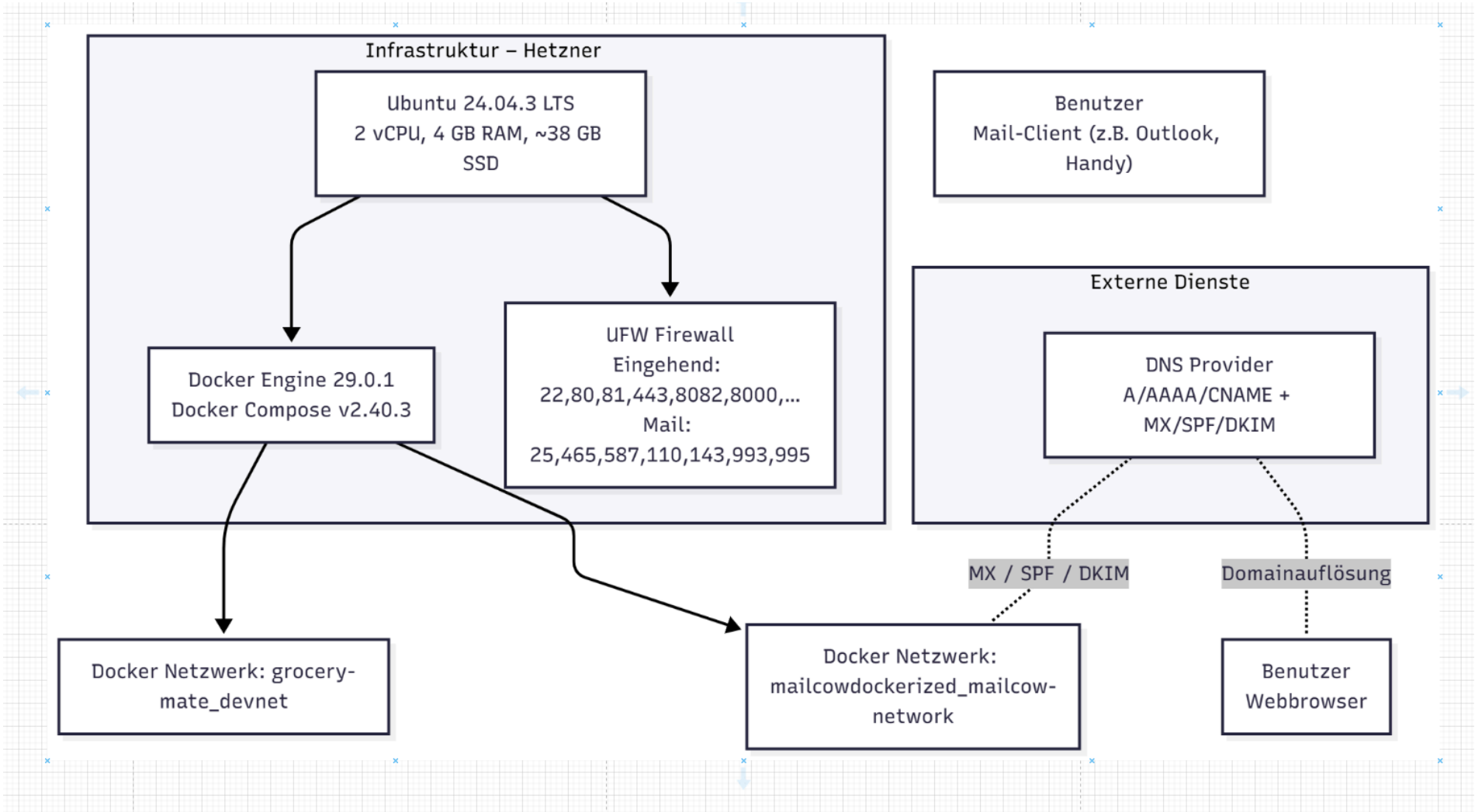
Die Skripte können in einem späteren Projektschritt ergänzt und als Anhang der Dokumentation hinzugefügt werden.

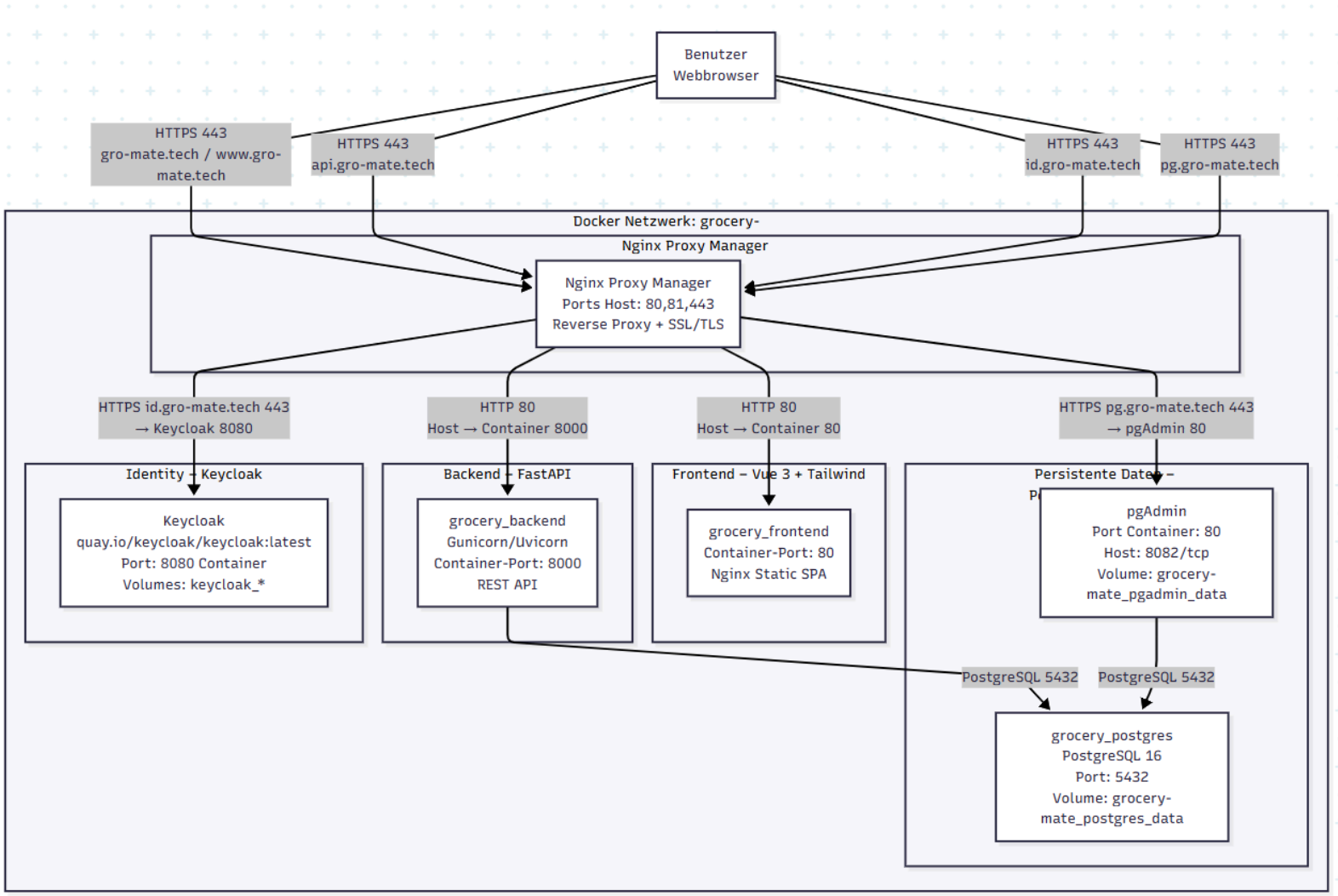
E. Netzwerkdiagramm (PDF/PNG)

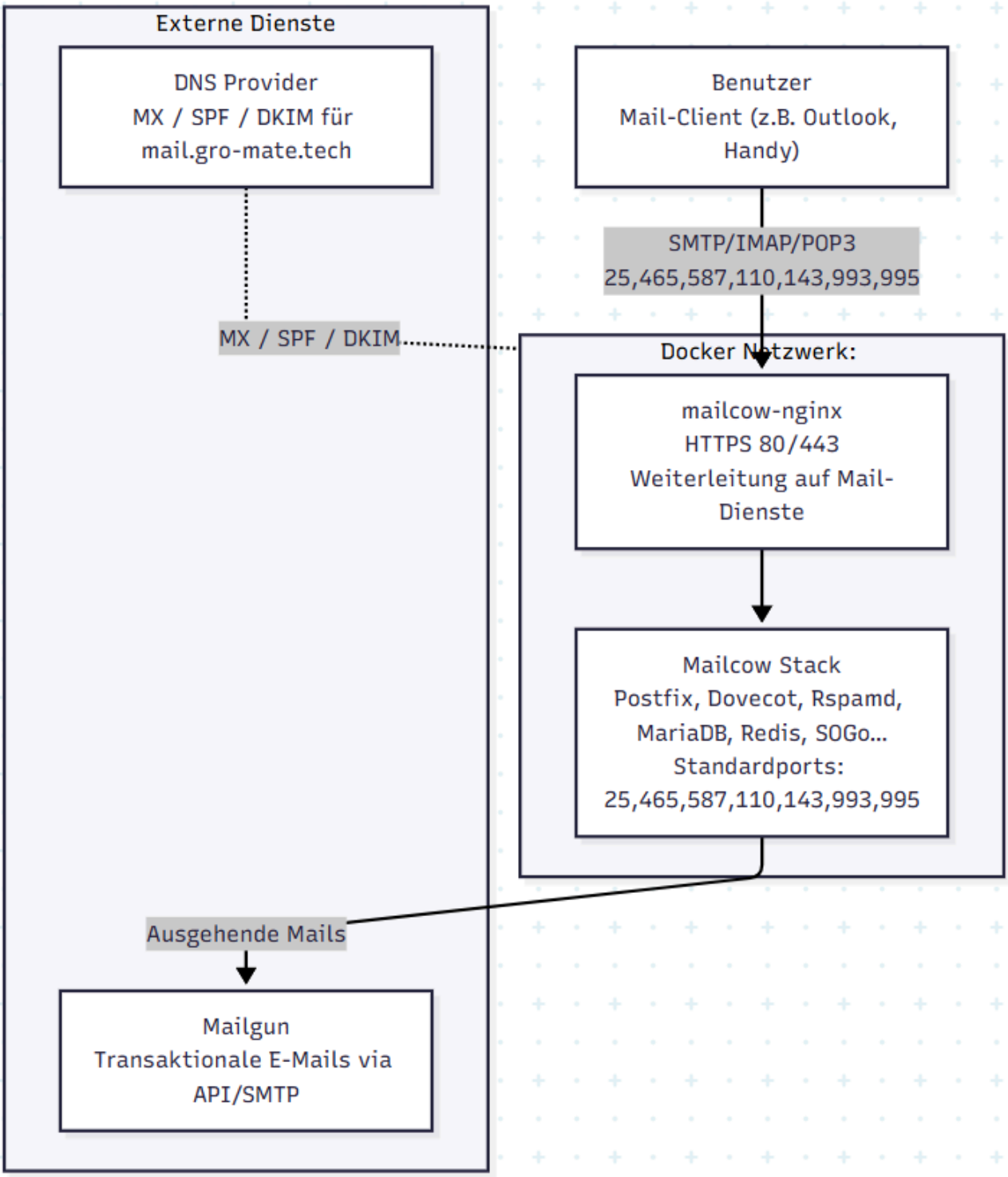
- Das Netzwerkdiagramm beschreibt:
    - Benutzerzugriff über HTTP/HTTPS
    - Nginx Proxy Manager als zentrale Eingangsschicht
    - Interne Docker-Netzwerke (`grocery-mate_devnet`, `mailcowdockerized_mailcow-network`)
    - Verbindung der Application-Layer (Frontend/Backend) mit der Datenbank-Layer (PostgreSQL, Mailc ow-DB)
- Export:

- Format: **PNG** und **PDF**

- Ablage z.B. unter
- ## Diagramme Infrastructure – GroceryMate
-
- **Image PNG :**
- https://github.com/jflpro/grocery-mate/blob/GM/docs/infra-grocerymate.png
-
- **Code Mermaid :**
- https://github.com/jflpro/grocery-mate/blob/GM/docs/infra-grocerymate.mmd


Infrastruktur – Hetzner
Ubuntu 24.04.3 LTS
2 vCPU, 4 GB RAM, ~38 GB SSD
Docker Engine 29.0.1
Docker Compose v2.40.3
UFW Firewall
Eingehend:
22,80,81,443,8082,8000,...
Mail:
25,465,587,110,143,993,995
Benutzer
Mail-Client (z.B. Outlook, Handy)
Externe Dienste
DNS Provider
A/AAAA/CNAME + MX/SPF/DKIM
MX / SPF / DKIM
Domainauflösung
Docker Netzwerk: grocery-mate_devnet
Docker Netzwerk: mailcowdockerized_mailcow-network
Benutzer
Webbrowser


Benutzer
Webbrowser
HTTPS 443
gro-mate.tech / www.gro-mate.tech
HTTPS 443
api.gro-mate.tech
HTTPS 443
id.gro-mate.tech
HTTPS 443
pg.gro-mate.tech
Docker Netzwerk: grocery-
Nginx Proxy Manager
Nginx Proxy Manager
Ports Host: 80,81,443
Reverse Proxy + SSL/TLS
HTTPS id.gro-mate.tech 443
→ Keycloak 8080
HTTP 80
Host → Container 8000
HTTP 80
Host → Container 80
HTTPS pg.gro-mate.tech 443
→ pgAdmin 80
Identity – Keycloak
Keycloak
quay.io/keycloak/keycloak:latest
Port: 8080 Container
Volumes: keycloak_*
Backend – FastAPI
grocery_backend
Gunicorn/Uvicorn
Container-Port: 8000
REST API
Frontend – Vue 3 + Tailwind
grocery_frontend
Container-Port: 80
Nginx Static SPA
pgAdmin
Port Container: 80
Host: 8082/tcp
Volume: grocery-mate_pgadmin_data
PostgreSQL 5432
PostgreSQL 5432
grocery_postgres
PostgreSQL 16
Port: 5432
Volume: grocery-mate_postgres_data


Externe Dienste
DNS Provider
MX / SPF / DKIM für
mail.gro-mate.tech
Benutzer
Mail-Client (z.B. Outlook,
Handy)
SMTP/IMAP/POP3
25,465,587,110,143,993,995
MX / SPF / DKIM
Docker Netzwerk:
mailcow-nginx
HTTPS 80/443
Weiterleitung auf Mail-
Dienste
Mailcow Stack
Postfix, Dovecot, Rspamd,
MariaDB, Redis, SOGo...
Standardports:
25,465,587,110,143,993,995
Ausgehende Mails
Mailgun
Transaktionale E-Mails via
API/SMTP