

# BAS Prüfungsgenerator - Installation & Deployment auf VServer

Diese Anleitung beschreibt die Installation und Konfiguration des BAS Prüfungsgenerators auf einem Ubuntu LTS VServer mit eigener Domain und SSL-Zertifikat.

## Inhaltsverzeichnis

1. [Hardware-Anforderungen](#)
2. [Systemvoraussetzungen](#)
3. [Server-Setup](#)
4. [SSL-Zertifikat Integration](#)
5. [Deployment Konfiguration](#)
6. [Installation](#)
7. [Backup-Strategie](#)
8. [Monitoring & Wartung](#)
9. [Update-Prozedur](#)
10. [Troubleshooting](#)

---

## Hardware-Anforderungen

### Nutzungsanalyse

Das System ist für folgende Szenarien ausgelegt:

- **Erwartete aktive Nutzer:** 150
- **Gleichzeitige Nutzer (Normal):** 15-30 (10-20%)
- **Gleichzeitige Nutzer (Peak):** 40-80

### Empfohlene Konfiguration

Ressource	Spezifikation
vCPUs	4
RAM	8 GB
Storage	40 GB SSD

---

## Systemvoraussetzungen

- **Betriebssystem:** Ubuntu 24.04 LTS
- **Root-Zugriff** oder sudo-Berechtigungen
- **Offene Ports:** 22 (SSH), 80 (HTTP), 443 (HTTPS)
- **Domain:** Eigene Domain mit DNS-Konfiguration
- **SSL-Zertifikat:** PEM-Format (cert.pem + key.pem)
- **LDAP-Zugang:** LDAP-Server für Benutzerauthentifizierung

---

## Server-Setup

## 1. Basis-Installation

```
# System aktualisieren  
sudo apt update && sudo apt upgrade -y  
  
# Essenzielle Pakete installieren  
sudo apt install -y curl wget git nano ufw logrotate
```

- Tested

## 2. Docker Installation

```
# Docker aus Ubuntu Repository installieren  
sudo apt install -y docker.io docker-compose-v2  
  
# Docker-Dienst starten und aktivieren  
sudo systemctl start docker  
sudo systemctl enable docker  
  
# Benutzer zur Docker-Gruppe hinzufügen  
sudo usermod -aG docker $USER  
  
# Abmelden und neu anmelden, damit Gruppenzugehörigkeit wirksam wird
```

- Tested

### Docker Installation verifizieren:

```
# Nach erneutem Login:  
docker --version  
docker compose version
```

- Tested

## 3. Firewall konfigurieren

```
# Firewall-Regeln einrichten  
sudo ufw allow 22/tcp      # SSH  
sudo ufw allow 80/tcp      # HTTP  
sudo ufw allow 443/tcp     # HTTPS  
  
# Firewall aktivieren  
sudo ufw enable  
  
# Status prüfen  
sudo ufw status verbose
```

- Tested

## 4. Verzeichnisstruktur erstellen

```
# Arbeitsverzeichnis erstellen
sudo mkdir -p /opt/bas-pruefungsgenerator
cd /opt/bas-pruefungsgenerator

# Datenverzeichnisse anlegen
sudo mkdir -p data/{app,postgres,ssl,backups,logs/caddy}

# Berechtigungen setzen
sudo chown -R $USER:$USER /opt/bas-pruefungsgenerator
chmod 755 data
```

- Tested

## SSL-Zertifikat Integration

### 1. Zertifikate bereitstellen

Der Kunde stellt folgende Dateien bereit:

- **cert.pem**: SSL-Zertifikat (Full Chain)
- **key.pem**: Private Key

```
# Zertifikate in SSL-Verzeichnis kopieren
sudo cp /pfad/zum/cert.pem /opt/bas-pruefungsgenerator/data/ssl/cert.pem
sudo cp /pfad/zum/key.pem /opt/bas-pruefungsgenerator/data/ssl/key.pem
```

### ⚠️ NUR ZUM TESTEN Selfsigned Zertifikat erstellen ⚠️

```
cd /opt/bas-pruefungsgenerator/data/ssl

openssl req -x509 -nodes -days 365 -newkey rsa:2048 \
-keyout key.pem \
-out cert.pem \
-subj "/C=DE/ST=SN/L=Leipzig/O=Meetle GmbH/OU=Nachhall Ops/CN=bas-pg-test.meetle.dev" \
-addext "subjectAltName=DNS:bas-pg-test.meetle.dev"
```

### Self-Signed Zertifikat im Browser trotz HSTS etc. akzeptieren:

- **Chrome**: Auf der Fehlerseite `thisisunsafe` tippen (ohne Eingabefeld)

### !! WICHTIG: Zugriffsrechte

```
# Berechtigungen einschränken (wichtig für Sicherheit!)
sudo chmod 600 /opt/bas-pruefungsgenerator/data/ssl/*.pem
sudo chown root:root /opt/bas-pruefungsgenerator/data/ssl/*.pem
```

## Cert Prüfen

```
# Gültigkeit prüfen
openssl x509 -in /opt/bas-pruefungsgenerator/data/ssl/cert.pem -noout -dates

# Subject und Issuer anzeigen
openssl x509 -in /opt/bas-pruefungsgenerator/data/ssl/cert.pem -noout -subject -issuer

# Alle Details anzeigen
openssl x509 -in /opt/bas-pruefungsgenerator/data/ssl/cert.pem -text -noout
```

- Tested

## 2. Caddyfile für Production erstellen

```
# Caddyfile anlegen
nano /opt/bas-pruefungsgenerator/Caddyfile
```

### Inhalt:

```
{
    auto_https off
}

:443 {
    tls /ssl/cert.pem /ssl/key.pem

    # Backend API
    handle /api/* {
        reverse_proxy localhost:3000
    }

    # Swagger Dokumentation
    handle /doc* {
        reverse_proxy localhost:3000
    }

    # Frontend (SPA)
    handle /* {
        root * /app/web
        try_files {path} /index.html
        file_server
    }

    # Logging
    log {
        output file /var/log/caddy/access.log
        format json
    }
}
```

```

# Security Headers
header {
    # HSTS (1 Jahr)
    Strict-Transport-Security "max-age=31536000; includeSubDomains; preload"
    # Clickjacking-Schutz
    X-Frame-Options "SAMEORIGIN"
    # XSS-Schutz
    X-Content-Type-Options "nosniff"
    # Referrer Policy
    Referrer-Policy "strict-origin-when-cross-origin"
}

# HTTP -> HTTPS Redirect
:80 {
    redirect https://:{host}{uri} permanent
}

```

- Tested

## Deployment Konfiguration

### 1. docker-compose.yml erstellen

```
nano /opt/bas-pruefungsgenerator/docker-compose.yml
```

#### Inhalt:

```

services:
  app:
    image: ghcr.io/campus-12/bas-pruefungsgenerator-container:latest
    container_name: bas-app
    restart: unless-stopped
    ports:
      - "80:80"
      - "443:443"
    volumes:
      # Persistente Daten
      - ./data/app:/data
      # Custom Caddyfile mit SSL
      - ./Caddyfile:/etc/caddy/Caddyfile:ro
      # SSL-Zertifikate
      - ./data/ssl:/ssl:ro
      # Logs
      - ./data/logs/caddy:/var/log/caddy
    depends_on:
      db:
        condition: service_healthy
    environment:
      # Node.js Performance (für 8GB RAM Server)

```

```

- NODE_ENV=production
- NODE_OPTIONS=--max-old-space-size=2048

# Backend Configuration
- CONSOLE_LOG_LEVEL=info
- ENABLE_SWAGGER=true
# Swagger verwendet Standard-Credentials aus Container (User: swagger, Password: swagger)
# - SWAGGER_USER=${SWAGGER_USER}
# - SWAGGER_USER_PASSWORD=${SWAGGER_PASSWORD}

# Database
-
DB_CONNECTION=postgres://${POSTGRES_USER}:${POSTGRES_PASSWORD}@db:5432/${POSTGRES_DB}?
sslmode=disable

# LDAP Configuration
- LDAP_SERVER_URL=${LDAP_SERVER_URL}
- LDAP_BIND_DN=${LDAP_BIND_DN}
- LDAP_BIND_CREDENTIALS=${LDAP_BIND_CREDENTIALS}
- LDAP_SEARCH_BASE=${LDAP_SEARCH_BASE}
- 'LDAP_SEARCH_FILTER=(sAMAccountName={{username}})'
- 'LDAP_USER_OBJECT_CLASSES=${LDAP_USER_OBJECT_CLASSES:-user,person}'
- LDAP_GROUP_FILTER=${LDAP_GROUP_FILTER}
- LDAP_GROUP_ADMIN=${LDAP_GROUP_ADMIN:-SG_PFG-Admin}
- LDAP_GROUP_EDUCATOR=${LDAP_GROUP_EDUCATOR:-SG_PFG-Educator}
- LDAP_GROUP_TRAINEE=${LDAP_GROUP_TRAINEE:-SG_PFG-Trainee}

# LDAP Sync (optional)
- LDAP_SYNC_ENABLED=${LDAP_SYNC_ENABLED:-true}
- LDAP_SYNC_CRON=${LDAP_SYNC_CRON:-0 */1 * * *}

# JWT Secret
- JWT_SECRET=${JWT_SECRET}
healthcheck:
  test: ["CMD", "wget", "--no-check-certificate", "--spider", "-q",
"https://localhost/api/core/version"]
  interval: 30s
  timeout: 10s
  retries: 3
  start_period: 40s
networks:
- bas-network

db:
  image: postgres:17-alpine
  container_name: bas-db
  restart: unless-stopped
  environment:
    - POSTGRES_USER=${POSTGRES_USER}
    - POSTGRES_PASSWORD=${POSTGRES_PASSWORD}
    - POSTGRES_DB=${POSTGRES_DB}

```

```

- POSTGRES_INITDB_ARGS="--encoding=UTF8 --locale=en_US.UTF-8
volumes:
- ./data/postgres:/var/lib/postgresql/data
ports:
- "127.0.0.1:5432:5432" # Nur localhost-Zugriff
healthcheck:
  test: ["CMD-SHELL", "pg_isready -U ${POSTGRES_USER} -d ${POSTGRES_DB}"]
  interval: 10s
  timeout: 5s
  retries: 5
networks:
- bas-network
command:
- "postgres"
# Performance Tuning für 8GB RAM Server
- "-c"
- "shared_buffers=1GB"          # 25% von 4GB (für DB reserviert)
- "-c"
- "effective_cache_size=3GB"    # 75% von 4GB
- "-c"
- "work_mem=16MB"
- "-c"
- "maintenance_work_mem=256MB"
- "-c"
- "random_page_cost=1.1"        # Optimiert für SSD
- "-c"
- "effective_io_concurrency=200" # SSD-optimiert
- "-c"
- "log_timezone=UTC"
- "-c"
- "timezone=UTC"
# Audit Logging (optional, auskommentiert für Production)
# - "-c"
# - "log_statement=mod"          # Log INSERT/UPDATE/DELETE
# - "-c"
# - "log_duration=on"           # Log Query-Ausführungszeit

networks:
bas-network:
  driver: bridge

```

- Tested

## 2. Environment-Variablen konfigurieren

```
nano /opt/bas-pruefungsgenerator/.env
```

### Inhalt (.env):

```
# PostgreSQL Configuration
POSTGRES_USER=bas_user
```

```

POSTGRES_PASSWORD=<HIER_SICHERES_PASSWORD_EINTRAGEN>
POSTGRES_DB=bas_pruefungsgenerator

# Swagger Authentication
# Verwendet Standard-Credentials aus Container: User=swagger, Password=swagger
# Falls eigene Credentials gewünscht: Variablen aktivieren (auch in docker-compose.yml) und
# bcrypt-Hash generieren
# SWAGGER_USER=admin
# SWAGGER_PASSWORD=<BCRYPT_HASH_HIER>

# JWT Secret (64+ Zeichen)
JWT_SECRET=<HIER_JWT_SECRET_EINTRAGEN>

# =====
# LDAP Configuration
# =====

# LDAP Server
LDAP_SERVER_URL=ldap://188.245.245.81:1389
# oder für LDAPS:
# LDAP_SERVER_URL=ldaps://kunde-ldap.domain.de:636

# Service Account für LDAP-Bind
LDAP_BIND_DN=cn=admin,dc=miracode,dc=io
LDAP_BIND_CREDENTIALS=<LDAP_SERVICE_PASSWORD>

# Benutzersuche
LDAP_SEARCH_BASE=ou=users,dc=miracode,dc=io

# Gruppenfilter
LDAP_GROUP_FILTER=(memberOf=cn={{group}},ou=Groups,dc=miracode,dc=io)

# LDAP Synchronisation (optional)
LDAP_SYNC_ENABLED=true
LDAP_SYNC_CRON=0 */1 * * *

```

- Tested

### 3. Secrets generieren

#### Sichere Passwörter und Secrets erstellen:

```

# PostgreSQL Passwort (32 Zeichen)
echo "POSTGRES_PASSWORD=$(openssl rand -hex 32)"

# JWT Secret (64 Zeichen)
echo "JWT_SECRET=$(openssl rand -base64 64)"

```

- Tested

**Wichtig:** Kopieren Sie die generierten Werte in die `.env` Datei.

## 4. Berechtigungen setzen

```
# .env-Datei vor unbefugtem Zugriff schützen  
chmod 600 /opt/bas-pruefungsgenerator/.env
```

- Tested

# Installation

## 1. GitHub Container Registry Login

```
# GitHub Token für campus-12 image repo  
export GITHUB_TOKEN="ghpxxxxxxxxxxxxxxxxxxxxxx"  
  
# Login durchführen (username für ghcr egal für docker cli aber benötigt)  
echo $GITHUB_TOKEN | docker login ghcr.io -u - --password-stdin
```

- Tested

## 2. Docker Image pullen

```
cd /opt/bas-pruefungsgenerator  
  
# Image herunterladen  
docker pull ghcr.io/campus-12/bas-pruefungsgenerator-container:latest
```

- Tested

## 3. Container starten

```
# Container im Hintergrund starten  
docker compose -f docker-compose.yml --env-file .env up -d  
  
# Logs überwachen  
docker compose -f docker-compose.yml logs -f
```

- Tested

## 4. Installation verifizieren

### Prüfen Sie folgende Punkte:

```
# 1. Container-Status  
docker compose -f docker-compose.yml ps  
  
# Erwartete Ausgabe:  
# NAME          IMAGE                      STATUS  
# bas-app      ghcr.io/campus-12/bas-pruefungsgenerator-container   Up (healthy)
```

```

# bas-db      postgres:17-alpine          Up (healthy)

# 2. Backend-API testen
curl -k https://ihre-domain.de/api/core/version

# Erwartete Ausgabe: {"version": "x.x.x"}

# 3. Frontend testen (im Browser)
# https://ihre-domain.de

# 4. Swagger-Dokumentation (im Browser mit Basic Auth)
# https://ihre-domain.de/doc

```

- Tested

## 5. Erste Schritte nach Installation

### Test-Login durchführen:

1. Browser öffnen: <https://ihre-domain.de>
2. Mit LDAP-Credentials einloggen
3. Prüfen, ob Benutzerrolle korrekt zugewiesen wurde

- Tested

## 6. Systemd Service einrichten (Autostart)

Um sicherzustellen, dass der Docker Compose Stack nach einem Server-Neustart automatisch startet:

```

# Systemd Service-Datei erstellen
sudo nano /etc/systemd/system/bas-pruefungsgenerator.service

```

### Inhalt der Service-Datei:

```

[Unit]
Description=BAS Prüfungsgenerator Docker Compose Stack
Requires=docker.service
After=docker.service network-online.target
Wants=network-online.target

[Service]
Type=oneshot
RemainAfterExit=yes
WorkingDirectory=/opt/bas-pruefungsgenerator
EnvironmentFile=/opt/bas-pruefungsgenerator/.env
ExecStart=/usr/bin/docker compose -f docker-compose.yml --env-file .env up -d
ExecStop=/usr/bin/docker compose -f docker-compose.yml down
TimeoutStartSec=300
Restart=on-failure
RestartSec=10s

```

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

#### Service aktivieren und starten:

```
# Systemd-Konfiguration neu laden
sudo systemctl daemon-reload

# Service aktivieren (Autostart bei Neustart)
sudo systemctl enable bas-pruefungsgenerator.service

# Service starten
sudo systemctl start bas-pruefungsgenerator.service

# Status prüfen
sudo systemctl status bas-pruefungsgenerator.service
```

#### Service-Management-Befehle:

```
# Service stoppen
sudo systemctl stop bas-pruefungsgenerator.service

# Service neu starten
sudo systemctl restart bas-pruefungsgenerator.service

# Logs anzeigen
sudo journalctl -u bas-pruefungsgenerator.service -f

# Autostart deaktivieren
sudo systemctl disable bas-pruefungsgenerator.service
```

#### Neustart testen:

```
# Server neu starten
sudo reboot

# Nach Neustart: Service-Status prüfen
sudo systemctl status bas-pruefungsgenerator.service

# Container-Status prüfen
docker compose -f /opt/bas-pruefungsgenerator/docker-compose.yml ps
```

- Tested

## Backup-Strategie

### 1. Automatisches Backup-Script erstellen

```
nano /opt/bas-pruefungsgenerator/backup.sh
```

#### Inhalt:

```
#!/bin/bash
#
# BAS Prüfungsgenerator - Backup Script
# Führt tägliche Backups von Datenbank und App-Daten durch
#
set -e # Bei Fehler abbrechen

BACKUP_DIR="/opt/bas-pruefungsgenerator/data/backups"
DATE=$(date +%Y%m%d_%H%M%S)
RETENTION_DAYS=30
COMPOSE_FILE="/opt/bas-pruefungsgenerator/docker-compose.yml"

echo "=====
echo "BAS Backup gestartet: $(date)"
echo "=====

# PostgreSQL Dump
echo "Erstelle PostgreSQL Backup..."
docker compose -f "$COMPOSE_FILE" exec -T db \
  pg_dump -U bas_user -d bas_pruefungsgenerator \
  --format=custom \
  --compress=9 \
  > "$BACKUP_DIR/db_backup_${DATE}.dump"

# Alternativ: SQL-Format
#docker compose -f "$COMPOSE_FILE" exec -T db \
#  pg_dump -U bas_user -d bas_pruefungsgenerator | \
#  gzip > "$BACKUP_DIR/db_backup_${DATE}.sql.gz"

echo "PostgreSQL Backup erstellt: db_backup_${DATE}.dump"

# App-Daten Backup (Templates, Uploads)
echo "Erstelle App-Daten Backup..."
tar -czf "$BACKUP_DIR/app_data_${DATE}.tar.gz" \
  -C /opt/bas-pruefungsgenerator/data app

echo "App-Daten Backup erstellt: app_data_${DATE}.tar.gz"

# Alte Backups löschen (älter als RETENTION_DAYS)
echo "Lösche alte Backups (älter als ${RETENTION_DAYS} Tage)..."
find "$BACKUP_DIR" -type f \(
  -name "*.dump" -o -name "*.sql.gz" -o -name "*.tar.gz" \
) \
-mtime +${RETENTION_DAYS} -delete

# Backup-Größe ausgeben
BACKUP_SIZE=$(du -sh "$BACKUP_DIR" | cut -f1)
echo "Gesamte Backup-Größe: $BACKUP_SIZE"
```

```
echo "====="
echo "BAS Backup abgeschlossen: $(date)"
echo "=====
```

- Tested

#### Script ausführbar machen:

```
chmod +x /opt/bas-pruefungsgenerator/backup.sh
```

- Tested

## 2. Cron-Job für automatische Backups

```
# Crontab bearbeiten
crontab -e

# Folgenden Eintrag hinzufügen:
# Backup täglich um 3:00 Uhr
0 3 * * * /opt/bas-pruefungsgenerator/backup.sh >> /var/log/bas-backup.log 2>&1
```

- Tested

## 3. Backup wiederherstellen

#### Datenbank wiederherstellen:

```
# Container stoppen
docker compose -f docker-compose.yml stop app

# Datenbank aus Custom-Format wiederherstellen
docker compose -f docker-compose.yml exec -T db \
  pg_restore -U bas_user -d bas_pruefungsgenerator --clean --if-exists \
  < /opt/bas-pruefungsgenerator/data/backups/db_backup_YYYYMMDD_HHMMSS.dump

# Alternativ: Aus SQL-Format
gunzip < /opt/bas-pruefungsgenerator/data/backups/db_backup_YYYYMMDD_HHMMSS.sql.gz | \
  docker compose -f docker-compose.yml exec -T db \
  psql -U bas_user -d bas_pruefungsgenerator

# Container neu starten
docker compose -f docker-compose.yml start app
```

- Tested

#### App-Daten wiederherstellen:

```
docker compose -f docker-compose.yml stop app

# Backup entpacken
```

```
tar -xzf /opt/bas-pruefungsgenerator/data/backups/app_data_YYYYMMDD_HHMMSS.tar.gz \
-C /opt/bas-pruefungsgenerator/data

# Container neu starten
docker compose -f docker-compose.yml restart app
```

- Tested

## Monitoring & Wartung

### 1. Log-Rotation konfigurieren

```
sudo nano /etc/logrotate.d/bas-pruefungsgenerator
```

#### Inhalt:

```
/opt/bas-pruefungsgenerator/data/logs/caddy/*.log {
    daily
    rotate 14
    compress
    delaycompress
    notifempty
    missingok
    create 0640 root root
    sharedscripts
    postrotate
        docker compose -f /opt/bas-pruefungsgenerator/docker-compose.yml exec app killall -HUP
caddy 2>/dev/null || true
    endscript
}

/var/log/bas-backup.log {
    weekly
    rotate 8
    compress
    delaycompress
    notifempty
    missingok
    create 0640 root root
}
```

- Tested

### 2. Healthcheck-Script

```
nano /opt/bas-pruefungsgenerator/healthcheck.sh
```

#### Inhalt:

```

#!/bin/bash
#
# Healthcheck-Script für externe Monitoring-Tools
#
DOMAIN="https://ihre-domain.de"
TIMEOUT=10

# Backend API prüfen
if curl -k -f -m $TIMEOUT "$DOMAIN/api/core/version" > /dev/null 2>&1; then
    echo "OK: Application is healthy"
    exit 0
else
    echo "CRITICAL: Application is down"
    exit 2
fi

```

```

chmod +x /opt/bas-pruefungsgenerator/healthcheck.sh
/opt/bas-pruefungsgenerator/healthcheck.sh

```

Output:

```
OK: Application is healthy
```

- Tested

### 3. Wichtige Monitoring-Befehle

```

# Container-Status prüfen
docker compose -f docker-compose.yml ps

# Live-Logs anzeigen
docker compose -f docker-compose.yml logs -f app
docker compose -f docker-compose.yml logs -f db

# Ressourcenverbrauch
docker stats bas-app bas-db

# Festplattenplatz prüfen
df -h /opt/bas-pruefungsgenerator
du -sh /opt/bas-pruefungsgenerator/data/*

# PostgreSQL-Statistiken
docker compose -f docker-compose.yml exec db \
    psql -U bas_user -d bas_pruefungsgenerator -c "
        SELECT schemaname, tablename,
            pg_size.pretty(pg_total_relation_size(schemaname||'.'||tablename)) AS size
        FROM pg_tables
        WHERE schemaname NOT IN ('pg_catalog', 'information_schema')"

```

```
ORDER BY pg_total_relation_size(schemaname||'.'||tablename) DESC  
LIMIT 10;  
"
```

#### 4. Wartungsfenster-Skript

```
nano /opt/bas-pruefungsgenerator/maintenance.sh
```

##### Inhalt:

```
#!/bin/bash  
#  
# Wartungsarbeiten durchführen  
#  
  
set -e  
  
echo "==== Wartung gestartet: $(date) ==="  
  
# 1. Backup erstellen  
/opt/bas-pruefungsgenerator/backup.sh  
  
# 2. Docker Images aufräumen  
echo "Räume alte Docker Images auf..."  
docker image prune -af --filter "until=720h" # 30 Tage  
  
# 3. Docker Volumes prüfen  
echo "Prüfe Docker Volumes..."  
docker volume ls -qf dangling=true | xargs -r docker volume rm  
  
# 4. Logs komprimieren  
echo "Komprimiere alte Logs..."  
find /opt/bas-pruefungsgenerator/data/logs -name "*.log" -mtime +7 -exec gzip {} \;  
  
# 5. PostgreSQL VACUUM  
echo "Führe PostgreSQL VACUUM durch..."  
docker compose -f /opt/bas-pruefungsgenerator/docker-compose.yml exec db \  
    psql -U bas_user -d bas_pruefungsgenerator -c "VACUUM ANALYZE;"  
  
echo "==== Wartung abgeschlossen: $(date) ==="
```

```
chmod +x /opt/bas-pruefungsgenerator/maintenance.sh  
  
# Wöchentliche Ausführung (Sonntag 4 Uhr)  
crontab -e  
# 0 4 * * 0 /opt/bas-pruefungsgenerator/maintenance.sh >> /var/log/bas-maintenance.log 2>&1
```

- Tested

## Update-Prozedur

### Reguläres Update

```
cd /opt/bas-pruefungsgenerator

# 1. Backup erstellen (wichtig!)
./backup.sh

# 2. Aktuelles Image für Rollback sichern
docker tag ghcr.io/campus-12/bas-pruefungsgenerator-container:latest \
    ghcr.io/campus-12/bas-pruefungsgenerator-container:backup-$(date +%Y%m%d)

# 3. Neues Image pullen
docker pull ghcr.io/campus-12/bas-pruefungsgenerator-container:latest

# 4. Container mit neuem Image starten (Rolling Update)
docker compose -f docker-compose.yml --env-file .env up -d --no-deps app

# 5. Logs überwachen
docker compose -f docker-compose.yml logs -f app

# 6. Healthcheck durchführen
./healthcheck.sh
```

- Tested

### Update mit Downtime (bei größeren Änderungen)

```
# 1. Backup erstellen
./backup.sh

# 2. Container stoppen
docker compose -f docker-compose.yml down

# 3. Neues Image pullen
docker pull ghcr.io/campus-12/bas-pruefungsgenerator-container:latest

# 4. Container neu starten
docker compose -f docker-compose.yml --env-file .env up -d

# 5. Logs prüfen
docker compose -f docker-compose.yml logs -f

# 6. Funktionstest durchführen
curl -k https://ihre-domain.de/api/core/version
```

- Tested

## Rollback durchführen

```
# Falls Update fehlschlägt: Rollback auf vorherige Version

# 1. Verfügbare Images anzeigen
docker images | grep bas-pruefungsgenerator-container

# Erwartete Ausgabe:
# ghcr.io/.../container    latest          abc123456789   1 hour ago   512MB
# ghcr.io/.../container    backup-20251023  2b62809e356e  7 days ago  508MB

# 2. Container stoppen
docker compose -f docker-compose.yml down

# 3. Alte Version in docker-compose.yml aktivieren
# Option A: Backup-Tag verwenden (wenn vorhanden)
#   image: ghcr.io/campus-12/bas-pruefungsgenerator-container:backup-20251023
#
# Option B: Image-ID verwenden (immer verfügbar)
#   image: 2b62809e356e
nano docker-compose.yml

# 4. Container mit alter Version starten
docker compose -f docker-compose.yml --env-file .env up -d

# 5. Alte Datenbank wiederherstellen (falls nötig, siehe Backup-Sektion)
```

- Tested

## Troubleshooting

### Container startet nicht

```
# Logs prüfen
docker compose -f docker-compose.yml logs app db

# Häufige Probleme:
# - Falsche Environment-Variablen in .env
# - Datenbank nicht erreichbar
# - Ports bereits belegt

# Ports prüfen
sudo netstat -tulpn | grep -E ':(80|443|5432)'

# Container-Konfiguration testen
docker compose -f docker-compose.yml config
```

### Datenbank-Verbindungsfehler

```
# PostgreSQL-Logs prüfen
docker compose -f docker-compose.yml logs db

# In Container einloggen und Verbindung testen
docker compose -f docker-compose.yml exec app sh
wget -O- http://db:5432 # Sollte PostgreSQL-Response zeigen

# Direkt zur Datenbank verbinden
docker compose -f docker-compose.yml exec db \
psql -U bas_user -d bas_pruefungsgenerator
```

## LDAP-Authentifizierung funktioniert nicht

```
# LDAP-Konfiguration prüfen
docker compose -f docker-compose.yml exec app env | grep LDAP

# LDAP-Verbindung testen (ldapsearch Tool im Container)
docker compose -f docker-compose.yml exec app sh
# apk add openldap-clients
# ldapsearch -x -H "$LDAP_SERVER_URL" -D "$LDAP_BIND_DN" -w "$LDAP_BIND_CREDENTIALS" -b "$LDAP_SEARCH_BASE"

# Backend-Logs nach LDAP-Fehlern durchsuchen
docker compose -f docker-compose.yml logs app | grep -i ldap
```

## SSL-Zertifikat-Fehler

```
# Zertifikat-Dateien prüfen
ls -la /opt/bas-pruefungsgenerator/data/ssl/

# Zertifikat-Gültigkeit prüfen
openssl x509 -in /opt/bas-pruefungsgenerator/data/ssl/cert.pem -noout -dates -subject

# Caddyfile-Syntax prüfen
docker compose -f docker-compose.yml exec app caddy validate --config /etc/caddy/Caddyfile

# Caddy-Logs prüfen
docker compose -f docker-compose.yml logs app | grep -i caddy
```

## Hohe Ressourcennutzung

```
# Ressourcenverbrauch überwachen
docker stats bas-app bas-db

# PostgreSQL Query-Performance analysieren
docker compose -f docker-compose.yml exec db \
psql -U bas_user -d bas_pruefungsgenerator -c "
SELECT pid, username, application_name, state, query, query_start
```

```

    FROM pg_stat_activity
    WHERE state != 'idle'
    ORDER BY query_start;
  ""

# Langsame Queries identifizieren (pg_stat_statements Extension erforderlich)
docker compose -f docker-compose.yml exec db \
  psql -U bas_user -d bas_pruefungsgenerator -c "
    SELECT query, calls, total_exec_time, mean_exec_time
    FROM pg_stat_statements
    ORDER BY mean_exec_time DESC
    LIMIT 10;
"

```

## Speicherplatz voll

```

# Speicherverbrauch analysieren
du -sh /opt/bas-pruefungsgenerator/data/*

# Alte Logs löschen
find /opt/bas-pruefungsgenerator/data/logs -name "*.log" -mtime +30 -delete

# Alte Backups löschen
find /opt/bas-pruefungsgenerator/data/backups -mtime +30 -delete

# Docker aufräumen
docker system prune -af
docker volume prune -f

```

## In Container einloggen für Debugging

```

# In App-Container
docker compose -f docker-compose.yml exec app sh

# In DB-Container
docker compose -f docker-compose.yml exec db sh

# Als root (falls nötig)
docker compose -f docker-compose.yml exec -u root app sh

```

## DNS-Konfiguration

### Für den Kunden: DNS A-Record einrichten

Hostname: pruefungsgenerator.ihre-domain.de  
Type: A  
Value: <IP-Adresse des VServers>  
TTL: 3600

## DNS-Propagation prüfen:

```
# Lokal testen  
nslookup pruefungsgenerator.ihre-domain.de  
  
# Weltweit testen  
# https://www.whatismydns.net
```

- Tested
- 

## Sicherheits-Checkliste

Nach der Installation folgende Punkte prüfen:

- **Firewall (ufw)** ist aktiv mit nur Port 22, 80, 443 offen
  - **SSH** nutzt Key-basierte Authentifizierung (Passwort deaktiviert)
  - **PostgreSQL** ist nur über localhost erreichbar (127.0.0.1:5432)
  - **Starke Passwörter** (min. 32 Zeichen) für PostgreSQL und ggf. Swagger
  - **JWT Secret** hat mindestens 64 Zeichen
  - **SSL-Zertifikate** sind mit chmod 600 geschützt
  - **.env-Datei** ist mit chmod 600 geschützt
  - **Swagger Basic Auth** ist in Production aktiviert
  - **SSL/TLS** läuft mit gültigem Zertifikat
  - **Automatische Backups** sind konfiguriert und getestet
  - **Log-Rotation** ist aktiv
  - **Docker-Container** haben restart-Policy "unless-stopped"
  - **Security Headers** sind in Caddyfile konfiguriert
  - **Monitoring** ist eingerichtet (Healthchecks, Logs)
  - **Systemd Service** ist eingerichtet; Prüfungsgenerator startet nach reboot automatisch
- 

**Letzte Aktualisierung:** Oktober 2025 **Version:** 1.0