ADMIN:DOCS

LEMP-Stack auf Ubuntu 24.04 LTS einrichten: Nginx, MariaDB, PHP 8.3 & Let’s Encrypt

Produktionsreife Installation des LEMP-Stacks auf Ubuntu 24.04 LTS: Nginx Webserver, MariaDB 10.11, PHP 8.3-FPM Pool-Tuning, UFW-Absicherung und automatisches Let’s Encrypt TLS.

Kategorie: Serverumgebungen **Veröffentlicht:** 13.10.2024

> **Aktuellere LTS-Generation verfügbar:** Planst du eine Neuinstallation auf der aktuellen Langzeitversion? Im Leitfaden LEMP-Stack auf Ubuntu 26.04 LTS einrichten findest du die Dokumentation für Nginx mit nativem HTTP/3 & QUIC, MariaDB 11.4 und PHP 8.5.

Der LEMP-Stack repräsentiert eine der bewährtesten und ressourceneffizientesten Architekturen für den Betrieb dynamischer Webanwendungen unter Linux. Linux bildet das stabile Betriebssystemfundament, Nginx fungiert als hochperformanter, asynchroner Webserver und Reverse Proxy, MariaDB stellt die relationale Datenbankverwaltung bereit und PHP führt die serverseitige Anwendungslogik aus.

Im Vergleich zu klassischen LAMP-Architekturen (Apache mit `mod_php`) zeichnet sich der LEMP-Stack durch strikte Prozessentkopplung aus. Nginx verarbeitet statische Anfragen über eine ereignisgesteuerte Non-Blocking-Architektur und delegiert dynamische Skriptaufrufe über Unix Domain Sockets an den FastCGI Process Manager (`php-fpm`). Das senkt den Arbeitsspeicherverbrauch bei parallelen Verbindungen drastisch und schützt das Gesamtsystem vor Lastspitzen.

Quelle: https://admindocs.de/de/serverumgebungen/ubuntu-24-04-lemp-stack-installieren

Dieser Leitfaden führt dich durch den Aufbau eines gehärteten, produktionsbereiten LEMP-Stacks auf Basis von **Ubuntu 24.04 LTS (Noble Numbat)**. Wir behandeln die Systemhärtung mit UFW und Fail2ban, das Leistungs-Tuning von MariaDB und PHP 8.3-FPM sowie die verschlüsselte Auslieferung über Let’s Encrypt mit modernem TLS 1.3.

> **Langzeitsupport (LTS):** Ubuntu 24.04 LTS garantiert offizielle Sicherheitsaktualisierungen und gepflegte Paketstände bis April 2029. In Produktionsumgebungen sorgt dieser planbare Fünfjahreszyklus für hohe Betriebsstabilität bei minimalem Wartungsaufwand.

> ⚠ **Voraussetzungen:** Erforderlich sind administrativer Zugriff (`sudo`) auf eine saubere Installation von Ubuntu 24.04 LTS, grundlegende Kenntnisse der Shell-Administration sowie ein öffentlich auflösbarer DNS-Record (A/AAAA) für deine Domain, falls ein Let's Encrypt TLS-Zertifikat ausgestellt werden soll.

Systemvoraussetzungen und Komponentenmatrix

Bevor Pakete installiert werden, sollte die Serverhardware auf den geplanten Workload abgestimmt sein. Nginx ist extrem genügsam, während relationale Datenbankabfragen und parallele PHP-Worker direkten Einfluss auf RAM und I/O-Performance haben.

Komponente	Paketversion in Ubuntu 24.04	Standard-Port / Socket	Mindestanforderung	Empfohlen für Produktion
Betriebssystem	Ubuntu 24.04 LTS	-	1 vCPU, 1 GB RAM	2–4 vCPUs, 4–8 GB RAM
Nginx	1.24+	TCP 80, 443	128 MB RAM	512 MB RAM
MariaDB	10.11 LTS	TCP 3306 (127.0.0.1)	512 MB RAM	2 bis 4 GB RAM (InnoDB Buffer)

Komponente	Paketversion in Ubuntu 24.04	Standard-Port / Socket	Mindestanforderung	Empfohlen für Produktion
PHP-FPM	8.3	`/run/php/php8.3-fpm.sock`	256 MB RAM	1 bis 2 GB RAM (für Worker)
Storage	NVMe / SSD	-	10 GB frei	25+ GB mit dediziertem Backup

1. Systemvorbereitung und Basisabsicherung

Jede saubere Serverbereitstellung beginnt mit einem aktualisierten Paketindex und einer restriktiven Firewall-Konfiguration. Dadurch wird verhindert, dass während der Einrichtung ungeschützte Standard-Dienste direkt aus dem Internet erreichbar sind.

Paketquellen aktualisieren und Bereinigung durchführen

Zunächst bringen wir den lokalen Paketcache auf den aktuellen Stand und installieren anstehende Kernel- und Sicherheitsupdates:

```
# Paketlisten aktualisieren und installierte Pakete upgraden
sudo apt update && sudo apt upgrade -y

# Nicht mehr benötigte Alt-Abhängigkeiten restlos entfernen
sudo apt autoremove --purge -y && sudo apt autoclean
```

Diagnose und Ressourcen-Check:

Vor der Installation größerer Serverdienste verifizieren wir verfügbaren Speicherplatz, Arbeitsspeicher und Systemzeit:

```
# Festplattenplatz auf der Root-Partition prüfen
df -h /

# RAM-Verfügbarkeit und Swap-Status kontrollieren
free -h

# Zeitsynchronisation via NTP sicherstellen
timedatectl status
```

Firewall (UFW) und Brute-Force-Schutz (Fail2ban)

Die Uncomplicated Firewall (UFW) verwaltet das Kernel-Paketfiltersystem (nftables) mit klaren Standardrichtlinien: Sämtlicher eingehender Traffic wird blockiert, ausgehende Verbindungen werden erlaubt.

```
# UFW und Fail2ban installieren
sudo apt install ufw fail2ban -y

# Standardrichtlinien definieren
sudo ufw default deny incoming
sudo ufw default allow outgoing

# SSH zwingend vor Aktivierung freischalten (Port 22)
sudo ufw allow OpenSSH

# Firewall aktivieren und Status prüfen
sudo ufw --force enable
sudo ufw status verbose
```

> ⚠ **Kritische Stolperfalle:** Aktiviere UFW niemals, ohne zuvor den SSH-Dienst (`sudo ufw allow OpenSSH` oder `sudo ufw allow 22/tcp`) explizit freigegeben zu haben. Andernfalls verlierst du mit dem Aktivierungsbefehl sofort den Remote-Zugriff und benötigst die Notfallkonsole deines Hosters.

```
# Bestehende Webserver-Dienste prüfen (z. B. Apache-Altlasten)
sudo ss -tuln | grep -E ':(80|443)' || echo "Ports 80 und 443 sind frei"
```

2. Nginx als Webserver installieren

Nginx fungiert als Frontend unseres Stacks. Er nimmt eingehende TCP-Verbindungen entgegen, terminiert TLS-Verschlüsselung, liefert statische Assets (CSS, JavaScript, Bilder) direkt aus und reicht dynamische PHP-Anfragen an den FPM-Socket weiter.

Nginx installieren und Firewall-Regel erweitern

```
# Nginx aus den offiziellen Ubuntu-Repositories installieren
sudo apt install nginx -y

# Dienst für automatischen Systemstart registrieren und starten
sudo systemctl enable nginx
sudo systemctl start nginx

# HTTP (Port 80) und HTTPS (Port 443) in der Firewall freigeben
sudo ufw allow 'Nginx Full'
sudo ufw status
```

Betriebszustand verifizieren:

```
# Laufenden Status der Systemd-Unit prüfen
systemctl status nginx --no-pager

# Lokale HTTP-Antwort testen
curl -I http://127.0.0.1
```

Die Antwort muss den Statuscode `HTTP/1.1 200 OK` zusammen mit dem Header `Server: nginx/...` zurückgeben.

> ⚠ **Portkonflikte vermeiden:** Startet Nginx nach der Installation nicht, blockiert häufig ein vorinstallierter Apache-Dienst (`apache2`) den Port 80. Prüfe mit `sudo ss -tuln | grep :80` . Falls Apache aktiv ist, deaktiviere ihn mit `sudo systemctl stop apache2 && sudo systemctl disable apache2` .

Verzeichnisstruktur von Nginx

Unter Debian und Ubuntu folgt Nginx einem modularen Aufbau, der Konfigurationsvorlagen (`sites-available`) von aktiven Instanzen (`sites-enabled`) über symbolische Links trennt:

```
┌──────────────────────────────────────────────────────┐
│                Nginx Verzeichnisstruktur             │
├──────────────────────────────────────────────────────┤
│                                                      │
│   /etc/nginx/                                        │
│   ├── nginx.conf          Globale Hauptkonfiguration │
│   ├── conf.d/             Modulare Server-Konfigurationen │
│   ├── sites-available/    VHost-Vorlagen (Inaktiv)   │
```

```
|   └── sites-enabled/      Aktive VHost-Symlinks            |
|                                                            |
└────────────────────────────────────────────────────────────┘
```

- **`nginx.conf`:** Globale Parameter wie Worker-Prozesse, Event-Modelle, Gzip-Komprimierung und Logging-Formate.
- **`sites-available/`:** Konfigurationsdateien für individuelle virtuelle Hosts (Domains).
- **`sites-enabled/`:** Enthält Symlinks auf Dateien in `sites-available/`. Nur VHosts in diesem Ordner werden aktiv von Nginx geladen.
- **`conf.d/`:** Globale modulare Konfigurations-Snippets für Upstream-Pools oder Sicherheitsrichtlinien.

3. MariaDB Datenbankserver bereitstellen

Als relationales Datenbanksystem setzen wir auf MariaDB 10.11 LTS. MariaDB bietet volle Kompatibilität zu MySQL, zeichnet sich jedoch durch die moderne InnoDB-Speicher-Engine, verbesserte Thread-Pools und hohe Performance bei Web-Workloads aus.

Installation und automatischer Start

```
# MariaDB Server und Client-Tools installieren
sudo apt install mariadb-server mariadb-client -y

# Dienst aktivieren und Status überprüfen
sudo systemctl enable mariadb
sudo systemctl status mariadb --no-pager
```

Absicherung mit `mariadb-secure-installation`

Direkt nach der Installation enthält die Datenbank unsichere Standardeinstellungen (anonyme Benutzerkonten, remote zugänglicher Root-Account und eine öffentliche Testdatenbank). Wir bereinigen diese Schwachstellen:

```
# Interaktiven Sicherheitsassistenten ausführen
sudo mariadb-secure-installation
```

Empfohlene Antworten für den Produktionsbetrieb:

1. **Enter current password for root:** `Enter` drücken (Standardmäßig ist kein Kennwort gesetzt).
2. **Switch to unix_socket authentication [Y/n]:** `Y` (erlaubt Root-Login via System-Root ohne Klartextpasswort).
3. **Change the root password? [Y/n]:** `Y` (Vergib ein komplexes, langes Kennwort und sichere es extern).
4. **Remove anonymous users? [Y/n]:** `Y` (Anonyme Accounts restlos entfernen).
5. **Disallow root login remotely? [Y/n]:** `Y` (Root-Zugriff strikt auf `localhost` beschränken).
6. **Remove test database and access to it? [Y/n]:** `Y` (Test-Tabellen löschen).
7. **Reload privilege tables now? [Y/n]:** `Y` (Rechte-Tabellen sofort neu einlesen).

> □□ **Socket-Authentifizierung unter Linux:** Ubuntu nutzt für den MariaDB-Root-Account standardmäßig das Plugin `unix_socket`. Das bedeutet: Der System-Benutzer `root` kann sich per `sudo mariadb` direkt ohne Passwort anmelden. Ein externes Root-Passwort wird primär benötigt, wenn Drittanbieter-Tools ohne Sudo-Rechte lokalen administrativen Zugriff anfordern.

MariaDB Verzeichnisstruktur und Tuning

Die Konfiguration ist modular über das Verzeichnis `/etc/mysql/` organisiert:



Zur Optimierung für produktive Web-Server passen wir in `/etc/mysql/mariadb.conf.d/50-server.cnf` die InnoDB-Ressourcenzuteilung an:

```
# Konfiguration zur Bearbeitung öffnen
sudo nano /etc/mysql/mariadb.conf.d/50-server.cnf
```

Kernparameter im Abschnitt `[mysqld]`:

```
[mysqld]
# Netzwerkbindung strikt auf localhost beschränken
bind-address            = 127.0.0.1
```

```
# Performance und InnoDB Tuning
innodb_buffer_pool_size = 512M          # Bei 2 GB RAM; bei 4 GB+ auf 50-70% des RAMs
setzen
innodb_log_file_size    = 128M          # Schreibt Transaktionen effizient ins Log
innodb_flush_log_at_trx_commit = 2     # Reduziert Disk-I/O bei Schreiblasten
innodb_file_per_table   = 1             # Erstellt eigene .ibd-Dateien pro Tabelle
max_connections         = 100           # Verhindert Überlastung durch zu viele Clients
```

Erläuterung der Tuning-Parameter:

Parameter	Empfohlener Wert	Wirkung im Serverbetrieb
`bind-address`	`127.0.0.1`	Schützt MariaDB vor direktem Netzwerkzugriff von außen
`innodb_buffer_pool_size`	50–70% des freien RAMs	Hält Tabellendaten und Indizes im schnellen Arbeitsspeicher
`innodb_flush_log_at_trx_commit`	`2`	Schreibt Logs im Sekundentakt auf Platte; drastischer I/O-Gewinn
`max_connections`	`50–150`	Begrenzt gleichzeitige Verbindungen und verhindert OOM-Abstürze

Nach der Anpassung starten wir den Dienst neu:

```
# MariaDB-Konfiguration neu einlesen
sudo systemctl restart mariadb
```

4. PHP 8.3-FPM installieren und abstimmen

PHP wird unter Ubuntu 24.04 in der Version 8.3 bereitgestellt. Der FastCGI Process Manager (`php-fpm`) läuft als eigenständiger Systemdienst und verwaltet Worker-Prozesse, die PHP-Dateien kompilieren und ausführen.

PHP-FPM und Kernmodule installieren

Für moderne Webapplikationen (wie WordPress, Nextcloud oder Laravel) wird neben dem FPM-Dienst eine Reihe spezialisierter PHP-Erweiterungen benötigt:

```
# PHP 8.3-FPM sowie Datenbank-, Bild- und Optimierungsmodule installieren
sudo apt install -y php8.3-fpm php8.3-common php8.3-mysql php8.3-xml
php8.3-curl php8.3-gd php8.3-mbstring php8.3-zip                    php8.3-opcache
php8.3-intl
```

Dienststatus und Module prüfen:

```
# Status von PHP-FPM kontrollieren
sudo systemctl status php8.3-fpm --no-pager

# Installierte PHP-Version und aktive Erweiterungen listen
php -v
php -m | grep -E '(mysqli|pdo_mysql|opcache|curl)'
```

> ⚠ **Komponenten-Abhängigkeit:** Achte zwingend darauf, dass das Paket `php8.3-fpm` installiert ist. Installierst du versehentlich nur das Metapaket `php` , bindet Ubuntu Apache

> als Abhängigkeit ein und startet einen kollidierenden Webserver.

PHP 8.3 Verzeichnisstruktur



Worker-Pool Tuning in `www.conf`

Die Datei `/etc/php/8.3/fpm/pool.d/www.conf` steuert, wie PHP-FPM Prozesse spawnt und Anfragen entgegennimmt. Standardmäßig nutzt Ubuntu den Modus `dynamic`:

```
; /etc/php/8.3/fpm/pool.d/www.conf
[www]
user = www-data
group = www-data

; Kommunikation über lokalen Unix-Domain-Socket
listen = /run/php/php8.3-fpm.sock
```

```
listen.owner = www-data
listen.group = www-data
listen.mode = 0660

; Prozessmanagement
pm = dynamic
pm.max_children = 25
pm.start_servers = 4
pm.min_spare_servers = 2
pm.max_spare_servers = 8
pm.max_requests = 500
```

Richtwerte zur Berechnung von `pm.max_children`:

```
# Durchschnittlichen RAM-Bedarf aktiver PHP-FPM-Worker ermitteln (in MB)
ps --no-headers -o rss -C php-fpm8.3 | awk '{total+=$1; count++} END {if (count>0) print
int(total/count/1024) " MB"; else print "60 MB (Standard-Richtwert)"}'

# Faustformel zur Berechnung von pm.max_children:
# max_children = (Verfügbarer RAM für PHP in MB) / (Durchschnittlicher RAM pro Worker in
MB)

# Rechenbeispiel: 2048 MB zugewiesener RAM bei ca. 60 MB pro Worker
echo $(( 2048 / 60 ))
# Ergebnis: 34
```

Direktive	Typischer Wert	Funktion
`pm.max_children`	`20–50`	Maximale Anzahl gleichzeitiger Worker-Prozesse
`pm.start_servers`	`4–8`	Anzahl der Worker, die beim Dienststart erzeugt werden
`pm.min_spare_servers`	`2–4`	Minimale Reserve an inaktiven Workern bei Leerlauf
`pm.max_spare_servers`	`6–12`	Maximale Reserve an inaktiven Workern

Quelle: https://admindocs.de/de/serverumgebungen/ubuntu-24-04-lemp-stack-installieren

Direktive	Typischer Wert	Funktion
`pm.max_requests`	`500`	Recycelt Worker nach 500 Requests, um Memory-Leaks zu verhindern

```
# Nach Konfigurationsänderung den FPM-Dienst neu laden
sudo systemctl reload php8.3-fpm
```

5. Integration von Nginx und PHP-FPM

Nachdem Nginx und PHP-FPM betriebsbereit sind, konfigurieren wir einen virtuellen Host (Server-Block), der PHP-Anfragen an den Unix-Socket `/run/php/php8.3-fpm.sock` weiterleitet.





Server-Block anlegen

Wir deaktivieren die Standard-Konfiguration und erstellen einen sauberen Virtual Host unter `/etc/nginx/sites-available/lemp.conf`:

```
# Neue Server-Block-Konfiguration anlegen
sudo nano /etc/nginx/sites-available/lemp.conf
```

Konfigurationsinhalt:

```
server {
    listen 80;
    listen [::]:80;
    server_name deine-domain.de www.deine-domain.de;

    root /var/www/html;
    index index.php index.html index.htm;

    # Security Header gegen Clickjacking und MIME-Sniffing
    add_header X-Frame-Options "SAMEORIGIN" always;
    add_header X-Content-Type-Options "nosniff" always;
    add_header X-XSS-Protection "1; mode=block" always;

    # Statische Dateien und Verzeichnisabfragen
    location / {
```

```
        try_files $uri $uri/ =404;
    }

    # Dynamische Skripte an den PHP 8.3 FastCGI Unix-Socket leiten
    location ~ \.php$ {
        include snippets/fastcgi-php.conf;
        fastcgi_pass unix:/run/php/php8.3-fpm.sock;
        fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
        include fastcgi_params;

        # Buffer-Tuning für dynamische Antworten
        fastcgi_buffer_size 128k;
        fastcgi_buffers 4 256k;
        fastcgi_busy_buffers_size 256k;
    }

    # Zugriff auf versteckte Dateien (.git, .env, .htaccess) strikt blockieren
    location ~ /\. {
        deny all;
        access_log off;
        log_not_found off;
    }
}
```

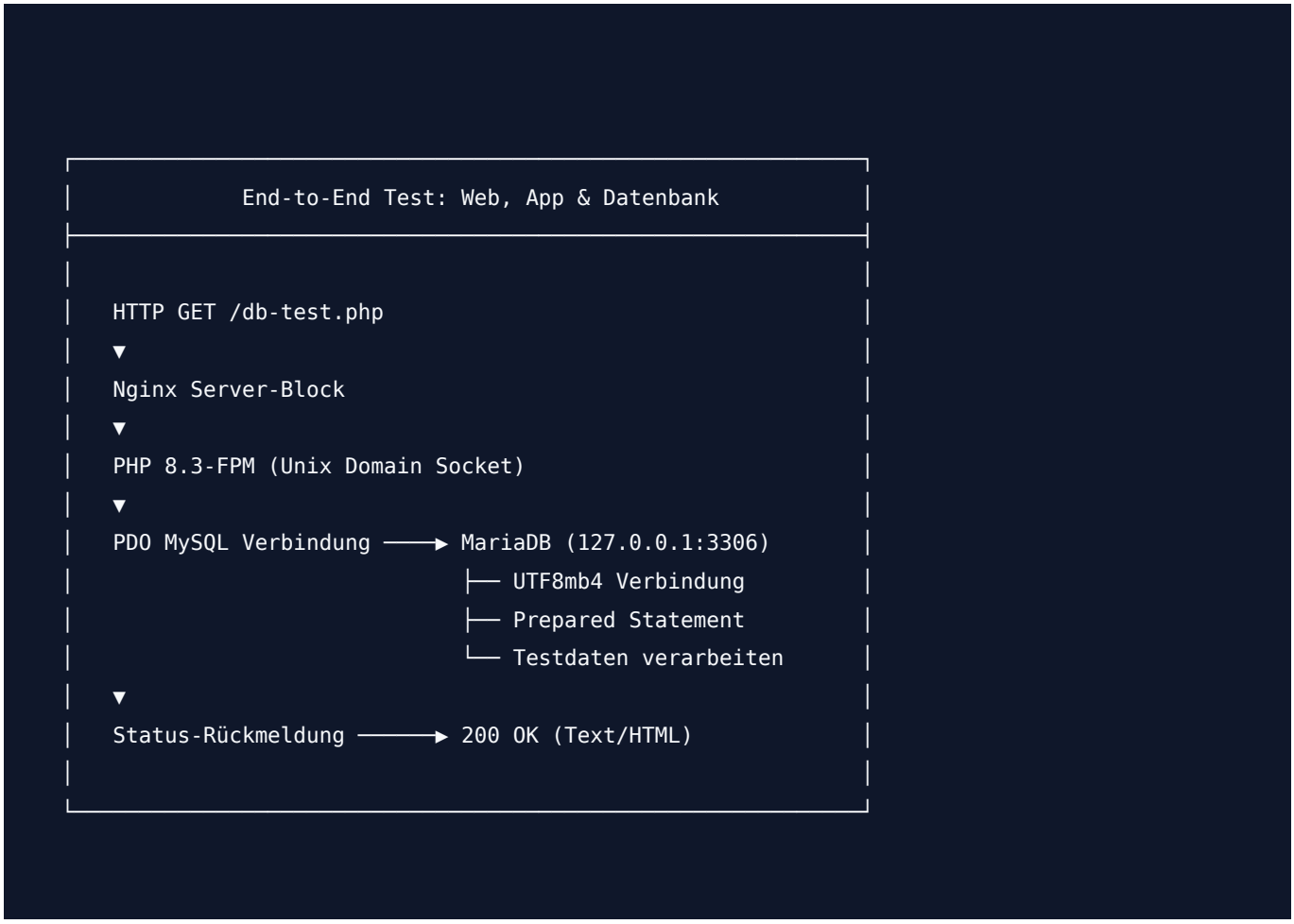
VHost aktivieren und Syntax validieren

```
# VHost über Symlink aktivieren
sudo ln -s /etc/nginx/sites-available/lemp.conf /etc/nginx/sites-enabled/

# Veralteten Standard-VHost deaktivieren
sudo rm -f /etc/nginx/sites-enabled/default

# Nginx-Syntax zwingend vor dem Reload prüfen
sudo nginx -t

# Nginx-Konfiguration ohne Downtime neu laden
sudo systemctl reload nginx
```

> ⚠ **FastCGI-Stolperfalle:** Erscheint im Browser der Fehler `502 Bad Gateway`, liegt das in 95 % aller Fälle an einem fehlerhaften Socket-Pfad oder Berechtigungsproblemen. Prüfe mit `ls -la /run/php/php8.3-fpm.sock`, ob der Socket existiert und dem Benutzer `www-data:www-data` gehört.

6. End-to-End-Validierung: Datenbank & Webstack

Wir verifizieren das Zusammenspiel aller Komponenten mit einer realen Testdatenbank und einem sicheren PHP-Skript unter Einsatz von Prepared Statements via PDO.



Dedizierten Testbenutzer in MariaDB anlegen

```
# In die MariaDB-Konsole einloggen
sudo mariadb -u root
```

SQL-Befehle ausführen:

```
CREATE DATABASE lemp_test CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;
CREATE USER 'lemp_user'@'localhost' IDENTIFIED BY 'EinStrengGeheimesPasswort123!';
GRANT ALL PRIVILEGES ON lemp_test.* TO 'lemp_user'@'localhost';
FLUSH PRIVILEGES;
EXIT;
```

Sicheres PDO-Testskript erstellen

Wir legen das Testskript unter `/var/www/html/db-test.php` an:

```
sudo nano /var/www/html/db-test.php
```

```
<?php
// /var/www/html/db-test.php - Nur für Verifikationszwecke!
declare(strict_types=1);
```

```
$dbConfig = [
    'host'    => '127.0.0.1',
    'dbname'  => 'lemp_test',
    'user'    => 'lemp_user',
    'pass'    => 'EinStrengGeheimesPasswort123!',
    'charset' => 'utf8mb4'
];

$options = [
    PDO::ATTR_ERRMODE            => PDO::ERRMODE_EXCEPTION,
    PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC,
    PDO::ATTR_EMULATE_PREPARES   => false,
    PDO::MYSQL_ATTR_INIT_COMMAND => 'SET NAMES utf8mb4'
];

try {
    $dsn =
"mysql:host={$dbConfig['host']};dbname={$dbConfig['dbname']};charset={$dbConfig['charset']
}";
    $pdo = new PDO($dsn, $dbConfig['user'], $dbConfig['pass'], $options);

    // Testtabelle erzeugen
    $pdo->exec("CREATE TABLE IF NOT EXISTS health_check (
        id INT AUTO_INCREMENT PRIMARY KEY,
        checked_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
    )");

    // Testdatensatz per Prepared Statement einfügen
    $stmt = $pdo->prepare("INSERT INTO health_check (checked_at) VALUES (NOW())");
    $stmt->execute();

    // Anzahl der Datensätze abrufen
    $count = (int) $pdo->query("SELECT COUNT(*) FROM health_check")->fetchColumn();

    header('Content-Type: text/plain; charset=utf-8');
    echo "LEMP-Stack Health Check: OK
";
    echo "PHP Version: " . PHP_VERSION . "
";
    echo "MariaDB Verbindung: Erfolgreich
";
    echo "Einträge in health_check: {$count}
";
```

```
} catch (PDOException $e) {
    http_response_code(500);
    header('Content-Type: text/plain; charset=utf-8');
    echo "LEMP-Stack Health Check: FEHLER
";
    echo "Grund: " . $e->getMessage() . "
";
}
```

```
# Dateirechte für den Webserver-Benutzer setzen
sudo chown www-data:www-data /var/www/html/db-test.php
sudo chmod 640 /var/www/html/db-test.php

# Test über die lokale CLI ausführen
curl -s http://localhost/db-test.php
```

Die Ausgabe bestätigt die korrekte Ausführung:

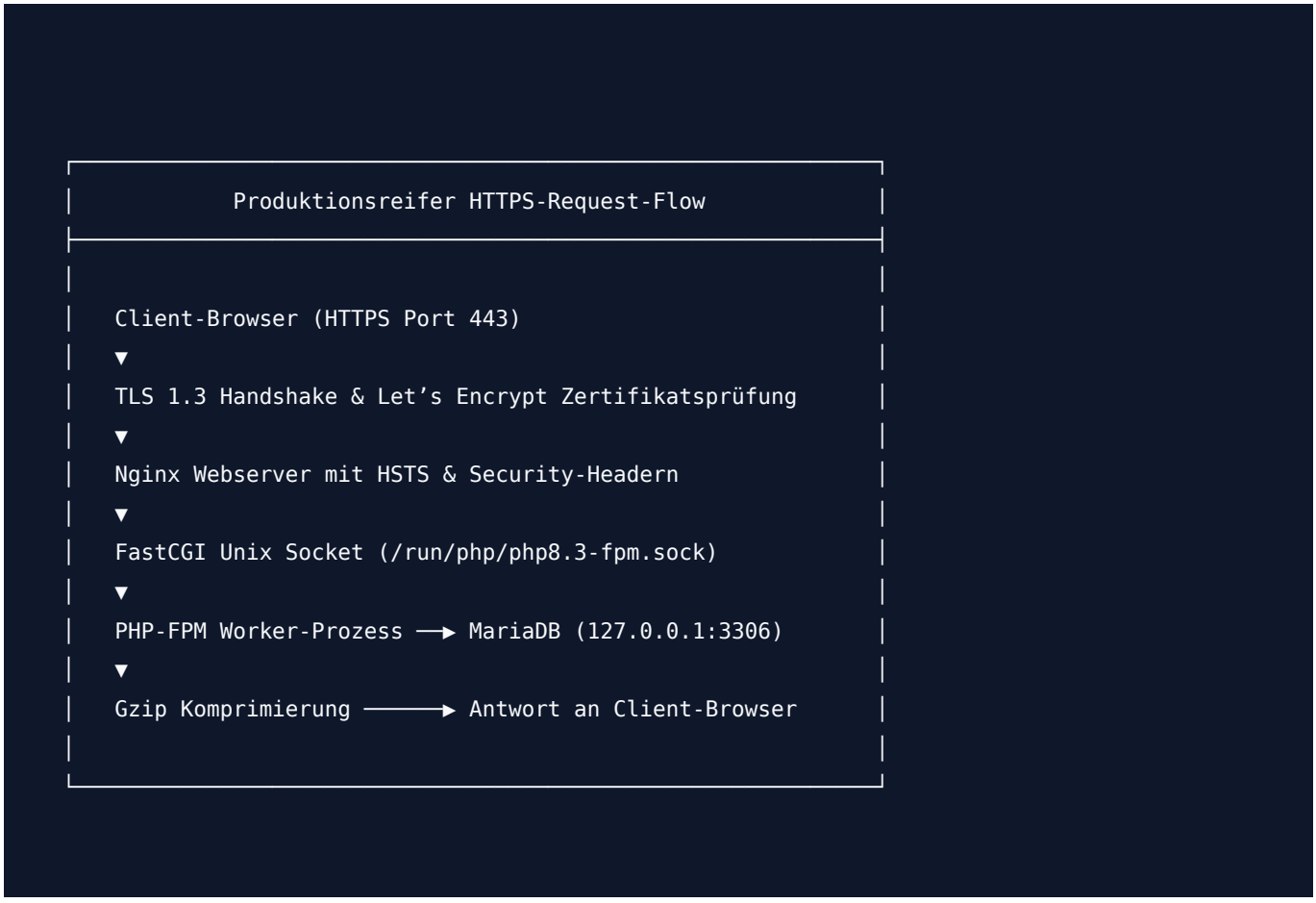
```
LEMP-Stack Health Check: OK
PHP Version: 8.3.x
MariaDB Verbindung: Erfolgreich
Einträge in health_check: 1
```

> ⚠ **Sicherheitsrisiko Testskripte:** Lösche die Datei `db-test.php` unmittelbar nach dem Funktionstest wieder (`sudo rm -f /var/www/html/db-test.php`). Öffentliche Skripte, die Datenbankstruktur oder Verbindungsdetails preisgeben, stellen ein vermeidbares

> Einfallstor für Angreifer dar.

7. TLS-Verschlüsselung mit Let’s Encrypt & Nginx-Härtung

Ein produktiver Webserver darf heutzutage ausschließlich über verschlüsselte HTTPS-Verbindungen betrieben werden. Wir automatisieren den Bezug und die Verlängerung von TLS-Zertifikaten mit dem EFF-Tool **Certbot**.



Certbot installieren und Zertifikat ausstellen

Die offizielle und am besten gewartete Version von Certbot wird über das Snap-Paketverwaltungssystem bezogen:

```
# Snapd sicherstellen und Certbot klassisch installieren
sudo snap install core && sudo snap refresh core
sudo snap install --classic certbot

# Symlink im Standard-Pfad erzeugen
sudo ln -sf /snap/bin/certbot /usr/bin/certbot

# Zertifikat anfordern und VHost automatisch für HTTPS konfigurieren
sudo certbot --nginx -d deine-domain.de -d www.deine-domain.de
```

Certbot fragt eine E-Mail-Adresse für Notfall-Benachrichtigungen ab, verifiziert den DNS-Eintrag über eine temporäre HTTP-01-Challenge, passt den Server-Block `/etc/nginx/sites-available/lemp.conf` automatisch auf Port 443 (SSL) an und richtet einen 301-Redirect von HTTP auf HTTPS ein.

Erweiterte TLS-Härtung und HSTS

Für ein A+-Rating bei SSL Labs ergänzen wir moderne TLS-Protokolle und HTTP Strict Transport Security (HSTS) im SSL-Serverblock:

```
# Ergänzung im Server-Block (Port 443)
ssl_protocols TLSv1.2 TLSv1.3;
ssl_prefer_server_ciphers off;
ssl_session_timeout 1d;
ssl_session_cache shared:MozSSL:10m;
ssl_session_tickets off;

# HSTS (erzwingt 2 Jahre HTTPS im Browser inklusive Subdomains)
add_header Strict-Transport-Security "max-age=63072000; includeSubDomains; preload"
always;
```

```
# Konfiguration validieren und Nginx neu laden
sudo nginx -t && sudo systemctl reload nginx

# Automatische Zertifikatserneuerung im Trockenlauf testen
sudo certbot renew --dry-run
```

> **Automatischer Renewal-Timer:** Certbot richtet bei der Snap-Installation automatisch einen systemd-Timer (`snap.certbot.renew.timer`) ein. Dieser prüft zweimal täglich, ob Zertifikate innerhalb der nächsten 30 Tage ablaufen, und verlängert sie vollautomatisch.

Befehlsreferenz (Cheatsheet)

Die wichtigsten operativen Befehle für Verwaltung, Überwachung und Fehlerdiagnose des LEMP-Stacks auf einen Blick:

Dienst / Aufgabe	Befehl	Erläuterung
Pakete & System	`sudo apt update && sudo apt upgrade -y`	Aktualisiert Paketquellen und installierte Pakete
Firewall (UFW)	`sudo ufw status verbose`	Zeigt aktive Regeln, Standardrichtlinien und Ports an
Firewall (UFW)	`sudo ufw allow 'Nginx Full'`	Öffnet HTTP (Port 80) und HTTPS (Port 443)
Nginx Syntax	`sudo nginx -t`	Prüft Konfigurationsdateien auf Syntax- und Pfadfehler

Dienst / Aufgabe	Befehl	Erläuterung
Nginx Reload	`sudo systemctl reload nginx`	Übernimmt Konfigurationsänderungen unterbrechungsfrei
MariaDB Shell	`sudo mariadb -u root -p`	Öffnet die administrative Datenbank-Konsole
MariaDB Status	`sudo systemctl status mariadb`	Überprüft Prozessstatus und Uptime des Datenbankservers
PHP-FPM Status	`sudo systemctl status php8.3-fpm`	Kontrolliert den Status des PHP-Prozessmanagers
PHP-FPM Socket	`ls -la /run/php/php8.3-fpm.sock`	Verifiziert Existenz und Berechtigung (`www-data`) des Sockets
TLS-Zertifikat	`sudo certbot --nginx -d deine-domain.de`	Bezieht und installiert Let's Encrypt TLS-Zertifikate
TLS-Erneuerung	`sudo certbot renew --dry-run`	Simuliert die automatische Verlängerung aller Zertifikate
Netzwerk-Ports	`sudo ss -tuln	grep -E ':(80|443|3306)'`	Kontrolliert gebundene Ports von Nginx und MariaDB

Weiterführende Ressourcen

Die folgenden internen Dokumentationen und Referenzen vertiefen den sicheren Betrieb, die Härtung und das Systemmanagement:

Ressource / Dokumentation	Beschreibung
Ubuntu-Upgrade: 22.04 auf 24.04 LTS	Grundlagen zum LTS-Releasezyklus, In-Place-Upgrades und Kernel-Features

Ressource / Dokumentation	Beschreibung
LEMP-Stack auf Ubuntu 26.04 LTS	Nachfolge-Leitfaden für Ubuntu 26.04 mit Nginx HTTP/3, MariaDB 11.4 und PHP 8.5
Linux Server Härtung: SSH & Firewall	Erweiterte Absicherung des Servers mit SSH-Key-Zwang, CrowdSec und Fail2ban
Virtuelle Maschinen für Tests	Isolierte Testumgebungen für Staging-Webserver unter KVM und Proxmox VE
Docker & Container-Workloads auf Linux	Container-basierte Bereitstellung von Webanwendungen und Microservices
Ubuntu Serverumgebungen	Übersicht aller Leitfäden für Administration, Wartung und Betrieb von Linux-Servern
Offizielle Nginx-Dokumentation	Referenzhandbuch zu allen Modulen, Upstream-Direktiven und Tuning-Parametern
Offizielle MariaDB-Dokumentation	Handbuch zu InnoDB-Engine, Replikation und Benutzer-Privilegien
Offizielle PHP-FPM-Dokumentation	Leitfaden zu Prozess-Managern, Pool-Direktiven und OPcache-Konfiguration

Fazit

Mit dem hier aufgebauten LEMP-Stack steht auf Ubuntu 24.04 LTS eine performante, ressourcenschonende und gehärtete Plattform für moderne Webanwendungen bereit. Durch das Zusammenspiel von Nginx als asynchronem Webserver, PHP 8.3-FPM über lokale Unix-Domain-Sockets und einem abgesicherten MariaDB 10.11 LTS-Server sind alle Komponenten sauber voneinander entkoppelt. Das Zusammenspiel aus strikten UFW-Firewallregeln, Fail2ban und automatisierter Let’s Encrypt-Verschlüsselung sorgt für ein stabiles Sicherheitsfundament im Produktivbetrieb.

> □□ **Praxistipp für den Produktivbetrieb:** Überwache die Auslastung deiner PHP-FPM-Worker in `/etc/php/8.3/fpm/pool.d/www.conf`. Bei steigenden Besucherzahlen sollten `pm.max_children`

> , `pm.start_servers` und `pm.max_spare_servers` an den verfügbaren Arbeitsspeicher angepasst werden, um Worker-Engpässe (`server reached max_children setting`) zu verhindern. Richte zudem in `php.ini` den OPcache mit mindestens 128 MB Speicher ein, um den Bytecode kompilierter PHP-Skripte im RAM zu halten und die Latenz drastisch zu senken.

Für den langfristigen Betrieb empfiehlt es sich, die Plattform schrittweise um ein zentrales Monitoring und automatisierte Log-Rotation zu ergänzen. Im nächsten Schritt kannst du deine Webserver-Infrastruktur mit fortgeschrittenen Sicherheitsmaßnahmen wie CrowdSec oder einer Multi-VHost-Architektur weiter ausbauen.

Quelle: https://admindocs.de/de/serverumgebungen/ubuntu-24-04-lemp-stack-installieren