KEULE SERVICES

Keule On-Prem: Technische Übersicht

Architektur, Installation, Updates, Betrieb und Sicherheit

Technisches Begleitdokument für Administratoren, IT-Leitungen und technische Entscheider

Ausgerichtet auf On-Prem-Betrieb, Release-Artefakte, Docker Compose, native Ubuntu-Installation, Preflight, Diagnose, Healthchecks und bewusste Updateprozesse.

Dokumentenstand: 24. Juni 2026

Inhaltsverzeichnis

1. Executive Summary

Keule On-Prem ist eine integrierte Kollaborations- und Organisationsplattform für Organisationen, die Datenhaltung und Betrieb in der eigenen Infrastruktur kontrollieren möchten. Die Anwendung bündelt Dateiablage, Aufgaben, Kalender, Chat, Notizen, News/Blog, Benachrichtigungen, Bereiche, Benutzerverwaltung und Systemmonitoring in einer Webanwendung.

Für Betreiber entscheidend ist der Wechsel vom Entwickler-Checkout zum Release-basierten Betrieb: Kundeninstallationen laufen über vorbereitete Release-Artefakte oder unveränderliche Container-Images. Auf Zielsystemen sind für fertige Releases kein Composer-Install und kein npm-/Node-Build erforderlich.

> **Admin-Hinweis**
> Docker Compose ist der empfohlene Standardweg. Die native Ubuntu-Installation bleibt eine unterstützte Alternative, wenn Containerbetrieb organisatorisch oder technisch nicht gewünscht ist.

Kernaussage	Bedeutung für Betreiber
Datenhoheit	MariaDB und Dateisystem liegen in der eigenen Betriebsumgebung. Persistente Daten werden nicht in Images oder Release-Artefakten transportiert.
Kontrollierte Installation	CLI, Preflight, Diagnose und Healthchecks machen Voraussetzungen und Blocker nachvollziehbar, bevor Dienste produktiv laufen.
Bewusste Updates	Updates müssen aktiv per CLI gestartet werden. Der Web-Systemmonitor zeigt Updateinformationen, startet aber keine Migrationen.
Verifizierte Distribution	Signatur, SHA-256, internes Release-Manifest und Paketvertrag werden vor Aktivierung geprüft.
Klare Grenzen	Kein automatischer Rollback nach Migrationen, keine Multi-Node-Freigabe und WebDAV nur für private Benutzerverzeichnisse.

2. Zielgruppe und Einsatzszenarien

Dieses Dokument richtet sich an Administratoren, IT-Leitungen, technische Ansprechpartner und Entscheider, die Keule als On-Prem-System bewerten, pilotieren oder betreiben möchten. Es ist bewusst technisch gehalten und konzentriert sich auf Architektur, Betrieb und Verantwortlichkeiten.

Szenario	Typischer Nutzen	Besondere Prüfpunkte
Verein / Verband	Gemeinsame Dateiablage, Aufgaben, Termine, News und Bereichsstruktur ohne große Cloud-Suite.	Rollenmodell, Selbstregistrierung, Mailversand, Backupverantwortung.
Schule / Bildungskontext	Bereiche für Klassen, Gruppen oder Projekte; zentrale Kommunikation und Organisation.	Datenschutz, externe Erreichbarkeit, Freigaberegeln, Administrationsprozess.
Kommunale Verwaltung / Organisationseinheit	Interne Kollaboration mit kontrollierter Datenhaltung und nachvollziehbarer Betriebsführung.	TLS/Reverse Proxy, Adminzugriff, Audit-Coverage, Update- und Backupfreigaben.
Kleines bis mittleres Unternehmen	Intranet-nahe Zusammenarbeit mit PWA, Dateiablage, Chat, Kalender und Aufgaben.	Integration in bestehende Infrastruktur, SMTP, WebDAV, Speicher- und Restoretests.

3. Funktionsumfang

Keule ist als modulare Webanwendung aufgebaut. Die folgenden Funktionen sind für die On-Prem-Bewertung relevant, ohne jedes Detail der Benutzeroberfläche auszubreiten.

Modul	Funktionaler Umfang	Technische Hinweise
Hub / Dashboard	Startseite mit Schnellzugriff, Statushinweisen, nächsten Terminen, Aufgabenstatus und Chat-Hinweisen, weitgehend Konfigurierbar.	PWA-fähige Navigation, mobile Pane-Umschalter, Command Palette.
Bereiche	Arbeitsräume für Teams, Gruppen oder Themen mit Join-Regeln, Mitgliedschaften und Bereichsdatei-Regeln.	Sichtbarkeit entsteht explizit über Bereichsfreigabe; Strict/Legacy-Regeln sind konfigurierbar.
Dateien	Private und gemeinsame Dateien, Ordner, Upload, Vorschau, Freigabelinks, Downloadcodes und Soft-Edit-Hinweise.	Userfiles bleiben Dateisystemdaten; Shared-/Bereichsregeln laufen über ACL-Logik.
Aufgaben	Status, Priorität, Fälligkeit, Erinnerungen, Kommentare, Checklisten, Anhänge, Assignees und	Reminder-State, Datei-/Event-Links und Hauptdaten liegen in MariaDB.

Modul	Funktionaler Umfang	Technische Hinweise
	Verknüpfungen.	
Kalender	Monats-, Wochen-, Tages- und Agendaansicht, Teilnehmer, Erinnerungen sowie ICS-/JSON-Import und -Export.	Kalenderdaten und Reminder-State liegen in MariaDB.
Notizen	Textnotizen, Checklisten, Farben, Pins, Archiv, Freigaben und Aufgabenverknüpfungen.	Notizen inkl. Freigaben und Task-Links sind DB-first.
Chat	Direkt- und Gruppenchats, Anhänge, Verlaufsladung, Read-State und Aufgabe-aus-Chat.	Long-Polling, Attachments mit Größen-/MIME-Grenzen, DB-first-Metadaten.
News / Blog	System-Feed, Bereichsfeeds, User-Feeds, Beiträge, Kommentare, Favoriten und Suche.	Rechte- und Moderationsmodell in MariaDB.
Benachrichtigungen / Push	In-App-Hinweise, Statusverlauf, Badge-Logik und optional Web-Push.	Push erfordert VAPID-Konfiguration und Browserberechtigung.
Administration / Monitoring	Benutzerverwaltung, Systemmonitor, Updatesicht, Diagnose, Zertifikats- und Logansichten.	Updateanzeige bleibt lesend; operative Updates bleiben CLI-only.

PWA-Grenze
Die PWA ist installierbar und unterstützt Service Worker sowie Push. Sie ist derzeit aber nicht als vollständige Offline-First-Datenanwendung zu verstehen; bei fehlender Verbindung sind Module eingeschränkt.

4. Technische Architektur

Keule ist eine serverseitige PHP-Webanwendung mit modularem Frontend-Bundling. Das Backend besteht aus PHP-FPM, gemeinsamer App-/DB-Logik, CLI-Kommandos, Migrationen und Hintergrundjobs. Das Frontend wird als gebaute Assets im Release ausgeliefert. Der produktive Fach-Storage ist DB-first, Binärdaten und Logs verbleiben im Dateisystem.

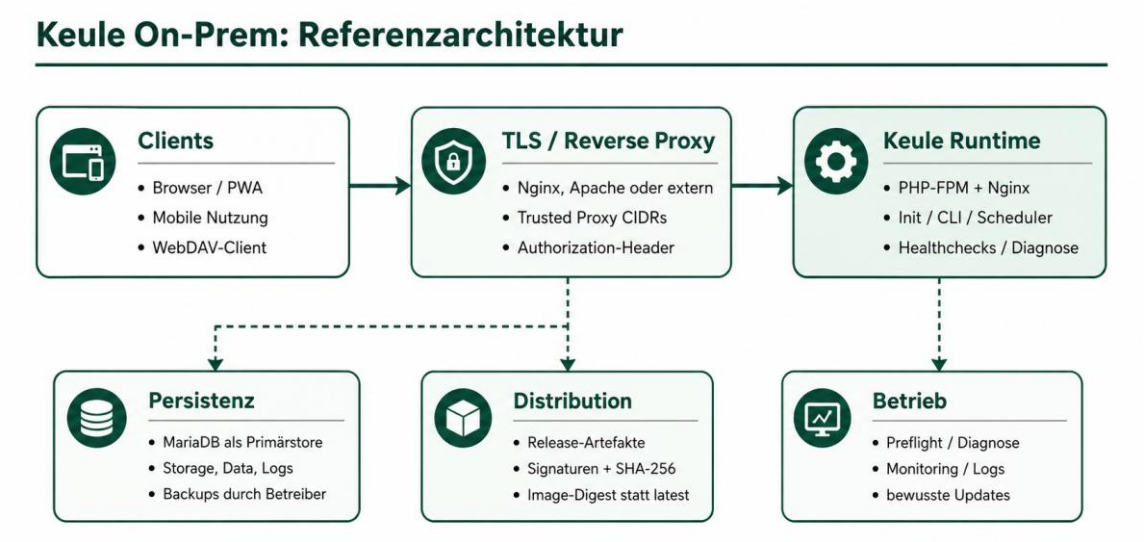

Abbildung 1: Schematische Keule-On-Prem-Referenzarchitektur

Schicht	Bestandteile	Betreiberrelevanz
Client	Browser, installierte PWA, WebDAV-Client.	TLS-Vertrauen, Browserkompatibilität und PWA-Origin liegen in der Betreiberumgebung.
Web/TLS	Nginx/Apache nativ oder Nginx im Compose-Stack; optional externer Reverse Proxy.	Document Root ist ausschließlich `public/`; Forwarded-Header werden nur bei Trust-CIDRs ausgewertet.
Anwendung	PHP-FPM, App-Code, APIs, CLI, Migrationen, Healthchecks.	Fertige Releases enthalten Produktionsabhängigkeiten und gebaute Assets.
Persistenz	MariaDB, Storage, Data, Logs, Attachments.	Backups müssen DB und Dateisystem konsistent gemeinsam erfassen.
Betrieb	Scheduler, Preflight, Diagnose, Updatecheck, Wartungsmodus.	Genau ein Scheduler; Updates und Migrationen laufen unter Lock.

5. Datenhaltung und Speicherorte

Die zentrale Architekturentscheidung lautet: Fachliche Kernobjekte liegen in MariaDB; Dateien, Anhänge, Logs und wenige Restdomänen bleiben Dateisystemdaten. Daraus folgt, dass ein vollständiges Backup immer Datenbank und Dateipfade gemeinsam betrachten muss.

Domäne	Primärspeicher	Hinweis
Benutzer, Profile, Einstellungen	MariaDB	Identity- und Profilzustände werden DB-seitig geführt.
Remember-, API-, Registrierung- und Reset-Tokens	MariaDB	Tokens werden nicht im Klartext gespeichert; Hash-/TTL-Logik beachten.
Bereiche, Mitgliedschaften, Requests	MariaDB	Grundlage für Organisationsmodell und Sichtbarkeit.
Aufgaben, Kalender, Notizen, Chat, News/Blog	MariaDB	Kernmodule inkl. Reminder-State und Verknüpfungen.
Notifications, Push-Subscriptions, VAPID	MariaDB	Push erfordert zusätzlich VAPID-Schlüssel.
Private und gemeinsame Dateien	Dateisystem	`storage/private`, `storage/shared` oder konfigurierte Betreiberpfade.
Chat-/Task-Anhänge	Dateisystem	Unter `DATA_PATH` beziehungsweise den Attachment-Pfaden.
Support-Tickets	JSON/Dateisystem	Aktive Restdomäne; bei Backup berücksichtigen.
Logs / Mail-Dumps	Dateisystem	Betriebsdiagnose; Aufbewahrung nach Betreiberpolicy.

Installationsart	Code	Konfiguration / Secrets	Persistenz / Logs
Docker Compose	Unveränderliches Image per Digest; App, Init und Scheduler nutzen dasselbe Image.	`.env`, `docker/compose.env`, Secret-Dateien unter `secrets/` und im Container unter `/run/secrets`.	Benannte Volumes oder Bind-Mounts für MariaDB, Storage, Data und Logs.
Native Ubuntu	`/opt/keule/releases/<version>` und aktiver Symlink `/opt/keule/current`.	`/etc/keule/keule.env` mit `0640 root:keule`; Token und Trust-Key unter `/etc/keule`.	`/var/lib/keule/storage`, `/var/lib/keule/data`, `/var/log/keule`.
Externe DB / NAS	Unverändert je Installationsweg.	DB-Zugangsdaten als Env/Secret; DB-TLS ist derzeit reserviert.	DB-Schema und gemounteter Storage müssen vorher bereitstehen; Keule mountet NFS/SMB nicht selbst.

6. Betriebsmodelle

Der Standardweg für neue Kundeninstallationen ist Docker Compose. Die native Ubuntu-Installation ist sinnvoll, wenn bestehende Betriebsrichtlinien Container ausschließen oder eine direkte Systemintegration mit Nginx/Apache, PHP-FPM und systemd bevorzugt wird.

Betriebsmodell	Eignung	Voraussetzungen	Grenzen
Docker Compose	Empfohlener Standard für die meisten On-Prem-Piloten und kleine bis mittlere Installationen.	Ubuntu 24.04/WSL2, Docker Engine 26+, Compose Plugin 2.24+, 2 vCPU, 4 GiB RAM, ca. 30 GiB frei.	Kein Multi-Node-Betrieb; produktives TLS typischerweise über externen Reverse Proxy.
Native Ubuntu	Direkte Integration in bestehende Linux-Betriebsprozesse und Webserverlandschaften.	Ubuntu 24.04/WSL2, PHP-FPM/CLI >= 8.2, MariaDB 10.11 oder externes Schema, Nginx/Apache.	Mehr OS-Verantwortung beim Betreiber; Paket-, Webserver- und Zertifikatszustand müssen gepflegt werden.
Externe DB	Betrieb mit vorhandener DB-Infrastruktur oder getrenntem DB-Host.	Vorab angelegtes Schema und unprivilegierter DB-Benutzer.	DB-TLS ist reserviert; Absicherung bis dahin über privates Netz oder Tunnel.
Host-/NAS-Bind-Mounts	Persistenz auf vorhandenen Storage-Systemen.	Gemounteter, verfügbarer Storage mit UID/GID-Rechten, Locking, Rename/Delete.	Keule mountet NAS nicht selbst; Backup und Mount-Verfügbarkeit sind Betreiberpflicht.

Empfohlener Startpunkt

Für einen ersten PoC sollte Docker Compose mit lokaler MariaDB, benannten Volumes und Reverse Proxy davor verwendet werden. Externe DB und Bind-Mounts sollten erst ergänzt werden, wenn der Basis-Smoke stabil ist.

7. Konfiguration und Preflight

Keule verwendet ein versioniertes Konfigurationsschema. Prozesswerte haben Vorrang vor Env-Dateien; native Installationen nutzen standardmäßig `/etc/keule/keule.env`, Docker Compose nutzt dieselbe Anwendungskonfiguration als `env_file` und transportiert sensitive Werte als Secret-Dateien.

Gruppe	Beispiele	Betreiberhinweis
App	`APP_ENV`, `APP_NAME`, `APP_URL`, `APP_TIMEZONE`, `APP_UPDATE_CHANNEL`	`APP_URL` muss eine Root-URL sein; URL-Unterpfade sind nicht unterstützt.
Organisation	`ORG_NAME`, `ORG_LEGAL_NAME`, `ORG_ADDRESS`, Kontakt- und Datenschutzadressen	Für Impressum, Mailtexte und rechtliche Anzeige relevant.
Datenbank	`DB_HOST`, `DB_PORT`, `DB_NAME`, `DB_USER`, `DB_PASS`	Bei Compose kann `DB_PASS` leer bleiben, wenn Secret-Datei genutzt wird.
Storage	`STORAGE_PATH`, `DATA_PATH`, `LOG_PATH`	Pfade müssen beschreibbar sein; Preflight legt fehlende produktive Storage-Pfade nicht blind an.
Auth / Registrierung	`AUTH_REGISTRATION_MODE`, Passwort- und Rate-Limit-Werte	Offene Registrierung benötigt Mailtransport. 2FA ist im Schema sichtbar, aber aktuell nicht produktiv umzusetzen.
Mail / Push / WebDAV	`MAIL_*`, `PUSH_ENABLED`, `WEBDAV_ENABLED`	Optionale Dienste werden nur geprüft, wenn sie aktiviert sind. WebDAV erfordert HTTPS.
Proxy / TLS	`KEULE_TRUSTED_PROXY_HEADERS`, `TRUSTED_PROXY_CIDRS`, Zertifikatspfade	Forwarded-Header nur aus explizit vertrauenswürdigen Quellnetzen.
Updates	`UPDATE_SERVER_URL`, `UPDATE_TOKEN_FILE`, `UPDATE_TRUST_KEY_FILE`	Updatecheck ist anonym; Downloads verwenden kundenspezifischen Token.

```
bin/keule config:assistant --output=/etc/keule/keule.env
bin/keule config:validate --env-file=/etc/keule/keule.env
bin/keule preflight --env-file=/etc/keule/keule.env
bin/keule diagnose --env-file=/etc/keule/keule.env --json
```

Preflight-Verhalten

`preflight` und `diagnose` sind read-only. Sie melden Blocker, Warnungen und OK-Zustände, geben aber keine Passwörter, Tokens oder privaten Schlüssel aus.

8. Installation

8.1 Docker Compose

Docker Compose startet MariaDB, Init-Service, PHP-FPM-App, Nginx und genau einen Scheduler. Nginx ist der einzige Dienst mit Host-Port; MariaDB und PHP-FPM bleiben intern. Der Init-Service validiert Konfiguration, wartet auf MariaDB, führt Migrationen aus und erstellt das initiale Administratorkonto idempotent.

```
cp .env.example .env
cp docker/compose.env.example docker/compose.env
install -d -m 0700 secrets
openssl rand -base64 36 > secrets/db_password
openssl rand -base64 48 > secrets/db_root_password
openssl rand -base64 36 > secrets/admin_password
chmod 0600 .env docker/compose.env
sudo chgrp 10001 secrets/*
chmod 0640 secrets/*

docker/bin/keule-compose config --quiet
docker/bin/keule-compose pull
docker/bin/keule-compose up -d
docker/bin/keule-compose ps
curl --fail http://127.0.0.1:8080/healthz
```

Compose-Datei	Zweck
compose.yaml	Standardstack mit App, Init, Scheduler, Nginx und lokaler MariaDB.
compose.bind-mounts.yaml	Ergänzung für Host- oder NAS-Pfade anstelle benannter Volumes.
compose.external-db.yaml	Variante für extern bereitgestellte DB ohne lokalen DB-Container.
compose.env.example	Compose-nahe Werte wie Image-Digest, Bind-Adresse, Port und Secret-Verzeichnis.
.env.example	Anwendungskonfiguration für App, Init und Scheduler.

8.2 Native Ubuntu-Installation

Die native Installation trennt unveränderlichen Release-Code und persistente Daten. Der erste Installerlauf ist ein read-only Plan. Änderungen werden erst mit `--apply` umgesetzt. Fehlende Pakete werden interaktiv bestätigt oder unbeaufsichtigt nur mit `--install-packages` installiert.

```
bin/keule config:assistant --output="$HOME/keule-native.env"

sudo ./bin/keule install \
  --env-file="$HOME/keule-native.env" \
  --webserver=nginx \
  --database=local \
  --scheduler=systemd

sudo ./bin/keule install \
  --apply \
  --env-file="$HOME/keule-native.env" \
  --webserver=nginx \
  --database=local \
  --scheduler=systemd
```

Native Komponente	Pfad / Verhalten
Releases	`/opt/keule/releases/<version>`; aktiv über `/opt/keule/current`.
Konfiguration	`/etc/keule/keule.env`, Eigentümer `root:keule`, Modus `0640`.
Storage / Data	`/var/lib/keule/storage`, `/var/lib/keule/data`, Sessions und temporäre Uploads.
Logs	`/var/log/keule`.
Webserver	Nginx oder Apache, Document Root `public/`, Konfiguration wird vor Reload getestet.
Scheduler	systemd-Timer als Standard oder Cron-Fallback; parallele verwaltete Scheduler sind Blocker.

8.3 TLS, Domains und Reverse Proxy

Modus	Zweck	Prüfpunkte
letsencrypt	Öffentliche Domain mit HTTP-01.	`APP_URL=https://DOMAIN`, DNS und Port 80 von außen erreichbar.
existing	Eigenes Zertifikat oder interne CA.	Fullchain/Key lesbar, Hostname/SAN, Ablaufdatum und Key-Match.
self-signed	Interne Namen oder IPs.	Clients müssen Zertifikat manuell vertrauen.
reverse-proxy	TLS terminiert vor Keule.	`KEULE_TRUSTED_PROXY_HEADERS=true` und enge `TRUSTED_PROXY_CIDRS`.
http	Bewusst bestätigte interne Ausnahme.	Nur mit `--allow-http`; kein stiller Fallback.

```
proxy_set_header Host $host;
proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
proxy_set_header X-Forwarded-Proto $scheme;
proxy_set_header Authorization $http_authorization;
```

9. Updates und Release-Verifikation

Sicherheit als Grundsatz: öffentlich abrufbare Metadaten, geschützte Artefakte, lokale Verifikation und bewusste Aktivierung. `update:check` bleibt anonym. `update` verwendet kundenspezifische Downloadtokens, prüft Signaturen und setzt Wartungsmodus, bevor produktive Änderungen erfolgen. Automatisiert über den Updater.

Schritt	Prüfung / Aktion	Ziel
1	Öffentliche Kanalmetadaten von `UPDATE_SERVER_URL` abrufen.	Anonyme Prüfung auf verfügbare Version.
2	Backup-Hinweis und bewusste Bestätigung.	Betreiber bestätigt, dass DB- und Dateibackups vorhanden sind.
3	Artefakt, `.sha256` und `.sha256.sig` mit Kundentoken laden.	Geschützter Download ohne Betreiber-Credentials.
4	Signatur der `.sha256` prüfen.	Update-Server allein reicht nicht als Vertrauensanker.
5	Archiv-SHA-256 und internes `RELEASE-MANIFEST.json` prüfen.	Manipulationen vor Aktivierung erkennen.
6	Paketvertrag per internem Verifier prüfen.	Nur erlaubte Release-Inhalte werden installiert.
7	Wartungsmodus aktivieren, Release oder Docker-Digest umschalten, Migrationen unter Lock ausführen.	Konsistenter, nachvollziehbarer Updatepfad.
8	Version, Health, DB, Storage und Scheduler prüfen.	Freigabe erst nach erfolgreichem Smoke.

```
bin/keule update:check --env-file=/etc/keule/keule.env

sudo bin/keule update \
  --env-file=/etc/keule/keule.env \
  --confirm-backup \
  --yes \
  --allow-production \
  --confirm-database=EXAKTES_SCHEMA
```

9.1 Docker-Update

Docker-Updates akzeptieren nur Metadaten mit unveränderlichem, signiertem Image-Digest. `latest` ist kein gültiger Updatevertrag. Ohne `--apply-compose` kann der geprüfte Digest vorbereitet werden; Pull und Recreate bleiben ein bewusster Betreiber-Schritt.

```
bin/keule update:check
bin/keule update --mode=docker --confirm-backup --yes

docker/bin/keule-compose pull
docker/bin/keule-compose up -d --force-recreate
docker/bin/keule-compose ps
curl --fail http://127.0.0.1:8080/healthz
```

Backup-Pflicht vor Updates
Vor jedem echten Update sind zur Zielversion passende Datenbank- und Dateibackups Betreiberpflicht.

10. Sicherheit und Zugriffskontrolle

Das Sicherheitsmodell kombiniert klassische Webschutzmaßnahmen, rollenbasierte Administration, bereichsbezogene Sichtbarkeit, Datei-ACLs, Rate Limits und harte Pfadprüfung. Betreiber müssen zusätzlich TLS, Reverse Proxy, Firewall, Backupschutz und Betriebssystemhärtung verantworten.

Bereich	Mechanismus	Betreiberrelevanz
Authentifizierung	Passwort-Hashes, Session-Cookies, Remember-Me-Rotation, Passwort-Reset-Tokens.	HTTPS und sichere Cookie-Kontexte produktiv erzwingen.
Autorisierung	Rollen `Administrator`, `Moderation`, `Benutzer`, Bereichsrechte und ACL-Prüfungen.	Berechtigungsmodell vor Pilot besprechen und testen.
CSRF	Session-Token und Header/Formularprüfung für mutierende Aktionen.	Reverse Proxy darf Header nicht verfälschen.

Bereich	Mechanismus	Betreiberrelevanz
Rate Limits	Login-, Registrierung- und Support-Limits.	Grenzwerte auf reale Nutzung abstimmen.
Dateisicherheit	Path-Containment, Uploadprüfung, ACL, Soft-Locks.	Storage-Rechte, NAS-Verhalten und Backupzugriff kontrollieren.
WebDAV	Nur private Verzeichnisse, Basic Auth mit API-Token, HTTPS-Pflicht, Write-Audit.	API-Token-Prozess und Clientvertrauen dokumentieren.
Audit	Strukturierte App- und Error-Logs, Audit-Ansichten.	Audit-Coverage ist nicht gleich vollständiges OS-/Webserver-Journal.

Sichtbarkeitsmodus
Im Strict-Betrieb werden Konfliktfälle fail-closed behandelt. Legacy-Kompatibilität ist als Übergangspfad zu verstehen und sollte nicht unbemerkt parallel zu Strict-Erwartungen betrieben werden.

11. Betrieb, Monitoring und Diagnose

Der Betrieb stützt sich auf Healthchecks, CLI-Diagnose, Scheduler-Heartbeat, Logs, Systemmonitor und optional Zertifikats-/Speichermonitoring. Entscheidend ist, dass Diagnosewerkzeuge Blocker und Warnungen anzeigen, ohne Secrets auszugeben oder produktive Pfade ungefragt zu verändern.

Werkzeug / Ansicht	Prüft / zeigt	Typische Betreiberaktion
`/healthz`	DB-Erreichbarkeit, Migrationen, Admin-Konto, Runtimepfade.	Nach Start, Update und Restore prüfen.
`bin/keule preflight`	Konfiguration, PHP, DB, Storage, TLS/Proxy, Scheduler und optionale Dienste.	Vor Installation, nach Domainwechsel, vor Go-live.
`bin/keule diagnose --json`	Detaillierte Diagnose mit OK/WARNING/BLOCKER.	Für Betriebsdokumentation und Supportfälle exportieren.
Scheduler-Heartbeat	Aktualisierung nach erfolgreichen periodischen Läufen.	Bei ausbleibendem Heartbeat Timer/Cron/Containerlogs prüfen.
Systemmonitor	Ressourcen, Storage, Updatesicht, Zertifikate und Logs je nach Konfiguration.	Kontrollansicht für Admins; keine Ersatzlösung für Infrastrukturmonitoring.
App-/Error-Logs	Strukturierte Betriebs- und Fehlerereignisse.	Logrotation, Zugriff und Aufbewahrung nach Betreiberpolicy.

```
docker/bin/keule-compose ps
docker/bin/keule-compose logs init
docker/bin/keule-compose logs app nginx scheduler
curl --fail http://127.0.0.1:8080/healthz

systemctl status keule-scheduler.timer
systemctl list-timers keule-scheduler.timer
journalctl -u keule-scheduler.service
```

12. Backup und Wiederherstellung

Ein belastbares Backup umfasst immer DB und Dateisystemdaten. Ein Restore muss vor produktivem Einsatz mindestens einmal in einer isolierten Umgebung getestet werden.

Backup-Bestandteil	Warum erforderlich	Hinweis
MariaDB/ DB	Kernobjekte, Benutzer, Bereiche, Aufgaben, Kalender, Chat, Notizen, News, Tokens und Metadaten.	Dump konsistent erstellen; Restore gegen passende Keule-Version testen.
Private/shared Dateien	Nutzdaten und gemeinsame Ablage.	Rechte, Eigentümer und Pfadstruktur erhalten.
Chat-/Task-Anhänge	Binäre Anhänge zu Modulen.	Zusammen mit DB sichern, damit Referenzen konsistent bleiben.
Support-Tickets	Noch nicht vollständig DB-migrierte Restdomäne.	Nicht vergessen, auch wenn Kernmodule DB-first sind.
Logs / Mail-Dumps	Diagnose und Nachvollziehbarkeit.	Je nach Compliance- und Speicherpolicy sichern oder rotieren.
Konfiguration / Secrets	Env-Datei, Update-Token, Trust-Key, Zertifikate.	Getrennt und besonders geschützt sichern; niemals in Release-Artefakte kopieren.

```
# Beispiel nativ: DB-Dump
mysqldump -u"$DB_USER" -p"$DB_PASS" "$DB_NAME" > keule.sql

# Beispiel Dateisystem: relevante Pfade sichern
rsync -aHAX /var/lib/keule/storage/ backup/keule-storage/
rsync -aHAX /var/lib/keule/data/    backup/keule-data/
rsync -aHAX /var/log/keule/         backup/keule-logs/
```

Restore-Regel
Nach fehlgeschlagener Migration darf nicht einfach ein älteres Image gegen ein bereits migriertes Schema gestartet werden. Restore bedeutet: zur Version passender DB-Stand plus passende Dateidaten.

13. Schnittstellen und Integrationen

Schnittstelle	Status / Funktion	Betriebsanforderungen
Web / PWA	Browserzugriff, installierbare PWA, Service Worker und Push-Handling.	Stabile Origin, TLS, ggf. Service-Worker-Neuladen bei Domainwechsel.
WebDAV	Private Benutzerverzeichnisse per `/dav` beziehungsweise `public/dav.php`.	HTTPS, API-Token, `Authorization`-Header im Proxy, PUT-Limit beachten.
E-Mail	SMTP, File-Mail oder deaktivierter Mailtransport.	Registrierung/Passwort-Reset benötigen aktiven Mailtransport. Test per `mail:test`.
Push	Web Push über VAPID und Browser-Subscription.	VAPID-Schlüssel generieren, Browserberechtigung, Diagnoseblocker beachten.
Update-Server	Anonymer `update:check`, geschützte Downloads mit Kundentoken.	Trust-Key, Token-Datei und Updatekanal pflegen.
Reverse Proxy	Externe TLS-Terminierung möglich.	Forwarded-Header nur aus konfigurierten CIDR-Netzen vertrauen.

13.1 WebDAV-Scope

WebDAV ist bewusst auf private Verzeichnisse begrenzt. `shared` und Bereichsverzeichnisse sind nicht Teil des aktuellen WebDAV-Scopes. Unterstützte Methoden sind `OPTIONS`, `PROPFIND`, `GET`, `HEAD`, `PUT`, `MKCOL`, `DELETE` und `MOVE`; Schreiboperationen werden auditiert.

WebDAV-Hinweis
Ein WebDAV-Client arbeitet mit Benutzername und API-Token. Das normale Benutzerpasswort sollte nicht als Integrationscredential verwendet werden.

14. Qualitätssicherung und Abnahme

Für Betreiber ist nicht nur die Erstinstallation relevant, sondern die reproduzierbare Abnahme: Basis-Smoke, Update-Smoke, Backup-/Restore-Test, optionale Dienste und Negativtests sollten vor produktiver Nutzung dokumentiert werden.

Abnahmebereich	Mindestprüfung
Installation	Release-/Imagebezug verifiziert, Konfiguration validiert, Preflight ohne Blocker.
Anwendung	Login mit `KeuleAdmin`, Session, Startseite, zentrale Module erreichbar.
Persistenz	DB-Verbindung, Migrationen, Storage-Read/Write/Rename/Delete.
Scheduler	Genau ein aktiver Scheduler; Reminder-/Heartbeat-Lauf erfolgreich.
TLS / Proxy	APP_URL, Zertifikat, Secure-Kontext, Forwarded-Header-Vertrauen und WebDAV-HTTPS-Gate.
Updates	`update:check`, Signatur-/Hashprüfung, Wartungsmodusverhalten, Healthcheck nach Update.
Backup / Restore	Wiederherstellung in isolierter Umgebung mit passender Version erfolgreich.
Optionale Dienste	Mailtest, Push/VAPID, WebDAV-Clienttest, Registrierung je gewähltem Modus.

Negativtest	Erwartetes Verhalten
Manipulierte Signatur oder Prüfsumme	Update bricht vor Installation ab.
Manipuliertes Archiv	SHA-256 oder interner Verifier blockiert.
Unvertrauenswürdige Forwarded-Header	HTTPS-Kontext wird nicht akzeptiert; WebDAV bleibt geschützt.
Parallele Scheduler	Preflight blockiert oder zweite Instanz bricht per Lock ab.
Source-Tree als Updateziel	Updater verweigert ohne explizit isoliertes Testziel.
Migrationsfehler	Wartungsmodus bleibt aktiv; kein automatischer Rollback.

15. Bekannte Grenzen und zu prüfende Punkte

Die folgenden Punkte sind keine Showstopper für einen Pilotbetrieb, sollten aber im Betreiberkonzept bewusst bewertet werden.

Grenze / Prüfpunkt	Einordnung
Keine URL-Unterpfade	`APP_URL` muss eine Root-URL sein; Betrieb unter z.B. `/keule` ist nicht vorgesehen.
Kein Multi-Node-Betrieb	Scheduler und Storage-Locks sind auf genau eine aktive Instanz ausgerichtet.
WebDAV nur private Verzeichnisse	Shared-/Bereichsordner bleiben außerhalb des aktuellen DAV-Scopes.
DB-TLS reserviert	Externe DB-Verbindungen bis zur vollständigen Umsetzung über privates Netz oder Tunnel absichern.
2FA noch nicht produktiv umgesetzt	In Entwicklung. Runtime soll fail-closed bleiben; `disabled` verwenden. Kommt zeitnah!
PWA nicht offline-first	Offline-Hinweise und Push ja, vollständige Offline-Datenbearbeitung nein.
Keine automatische DNS-/Firewall-/Routerkonfiguration	Insbesondere WSL2, Let's Encrypt HTTP-01 und öffentliche DAV-Domains erfordern Betreiberprüfung.
Kein automatischer DB-Rollback	Vor Updates und Migrationen müssen Backups vorliegen und Restorepfade getestet sein.
Audit-Coverage partial	App-/Error-Log-Auswertung ersetzt kein vollständiges SIEM oder OS-Journal.
Docker-Digest statt latest	Bewusst stabil; erfordert saubere Digest-/Signatur-Verifikation im Updateprozess.

16. Empfohlener Pilot-/PoC-Ablauf

1. Ziel und Scope festlegen: Nutzergruppen, Module, interne/externe Erreichbarkeit, Registrierung, Mail und WebDAV.
2. Betriebsmodell wählen: initial bevorzugt Docker Compose mit lokaler MariaDB und benannten Volumes.
3. DNS/TLS/Reverse Proxy vorbereiten und `APP_URL` endgültig festlegen.
4. Env-Datei mit `config:assistant` erstellen, Secrets getrennt ablegen und `config:validate` ausführen.
5. Preflight ohne Blocker erreichen; Warnungen dokumentieren und bewusst akzeptieren oder beheben.
6. Installation starten, Healthcheck und zentrale Module prüfen.
7. Rollen, Bereiche, ACLs und Beispielnutzer auf reale Organisationsstruktur abbilden.
8. Mail, Push und WebDAV nur aktivieren, wenn sie im PoC wirklich geprüft werden.
9. Backup- und Restorelauf in isolierter Umgebung durchführen.
10. Update-Smoke mit Wegwerf- oder Testinstanz durchführen.
11. Abnahmeprotokoll erstellen: Version, Konfiguration, Blocker/Warnungen, Smoke-Ergebnisse, Restpunkte.
12. Go-/No-Go anhand von Betriebsfähigkeit, Sicherheitsanforderungen und Restorefähigkeit entscheiden.

PoC-Empfehlung

Ein PoC sollte nicht nur zeigen, dass die Oberfläche funktioniert. Entscheidend sind Restore, Updatepfad, TLS/Proxy-Vertrauen, Berechtigungsmodell und ein realistisch getesteter Scheduler.

17. Anhang: wichtige Kommandos und Begriffe

17.1 Kommandos

Zweck	Kommando
Version anzeigen	`bin/keule version`
Konfigurationsassistent	`bin/keule config:assistant --output=/etc/keule/keule.env`
Konfiguration prüfen	`bin/keule config:validate --env-file=/etc/keule/keule.env`
Preflight	`bin/keule preflight --env-file=/etc/keule/keule.env`
Diagnose als JSON	`bin/keule diagnose --env-file=/etc/keule/keule.env --json`
Migration	`bin/keule migrate --allow-production --confirm-database=EXAKTES_SCHEMA`

Zweck	Kommando
Admin erstellen	`bin/keule admin:create --password-stdin --allow-production --confirm-database=EXAKTES_SCHEMA`
Mailtest	`bin/keule mail:test --to=admin@example.invalid`
VAPID erzeugen	`bin/keule vapid:generate --allow-production --confirm-database=EXAKTES_SCHEMA`
Update prüfen	`bin/keule update:check`
Update starten	`bin/keule update --confirm-backup --yes`
Wartungsmodus	`bin/keule maintenance:on` / `bin/keule maintenance:off`

17.2 Begriffe

Begriff	Bedeutung
Release-Artefakt	Fertiges `keule-<version>.tar.gz` mit PHP-Laufzeitdateien, Migrationen, Assets, Produktionsabhängigkeiten und Manifest.
RELEASE-MANIFEST.json	Interner Dateivertrag mit Version, Commit, Konfigurationsschema und SHA-256 pro Paketdatei.
Preflight	Read-only Prüfung der Installations- und Betriebsvoraussetzungen.
Healthcheck	Read-only Laufzeitprüfung über `/healthz`.
Digest	Unveränderlicher Container-Image-Bezug `image@sha256:<digest>` statt Tag `latest`.
Update-Token	Kundenspezifischer Downloadtoken für geschützte Release-Downloads; kein GitHub-Token.
Trust-Key	Öffentlicher Schlüssel zur Prüfung der Release-Signaturen.
Scheduler-Lock	Sperre gegen parallele Hintergrundläufe.
Strict Visibility	Betriebsmodus, der Sichtbarkeitskonflikte fail-closed behandelt.