LOGSHIP

ENTWURF V0.1 · 01.11.2026

# LogShip Cloud Betriebskonzept für den SaaS-Betrieb

Dedizierte 3PL-Instanzen und geteilte Händler-Instanzen auf Proxmox – zentral provisioniert, ausgerollt, überwacht und über LogShip abgerechnet.

BETRIEBSFORMEN

**Dedicated** – eine VM je Logistiker (3PL)
**Shared** – eine VM, viele Händler, je eine Organisation

STEUERUNGSEBENE

Neues Repo `logship-cloud`: Inventar + Ansible-Playbooks + eigene Nuxt-4 **Cloud Console** auf separater VM

ABRECHNUNG

Preis- und Feature-Matrix bleibt in LogShip (Prod-iDempiere). Console misst Nutzung, legt Rechnungsentwürfe auf Prod an; Lexoffice/SEPA wie bisher

GRUNDSATZ

Keine Flotten-/SaaS-Logik im ERP-Code. Nur ein später definierter, minimaler Berührungspunkt für die Feature-Buchung im Einstellungsbereich

LogYou GmbH · intern · Grundlage: Repo erp-nuxt-frontend (reference/template-cleanup, reference/edge-proxy, reference/ci), Golden-VM 115, Edge-Proxy VM 116, Jenkins deploy-instance

1 · ZUSAMMENFASSUNG

# Was gebaut werden soll – auf einer Seite

LogShip wird in zwei Formen als SaaS angeboten. Beide laufen als vollständige Instanz (iDempiere + PostgreSQL + Nuxt-Frontend + Strapi + Elasticsearch) in je einer VM, geklont von der Golden-VM. Neu ist die Steuerungsebene, die alle Instanzen kennt, erzeugt, aktualisiert und abrechnet.

### Dedizierte Instanz (3PL)

Ein Logistiker = eine VM, eigener Mandant 1000000, Händler als Organisationen – exakt das heutige Modell von app.logship.de. Vollständige Datentrennung je Kunde, individuelle Domain möglich.

### Geteilte Händler-Instanz

Eine VM, Betreiber LogYou, jeder Händler = eine Organisation mit eigenem Lager, eigener eingeschränkter Rolle (Ergebnis des bestehenden Onboarding-Assistenten). Für kleine Händler ohne eigenen 3PL.

## Drei Entscheidungen (01.11.2026)

| FRAGE | ENTSCHEIDUNG |
|---|---|
| Onboarding der Händler | Vertriebsgeführt über den vorhandenen Assistenten; Self-Service von der Landingpage später über dieselbe Schnittstelle |
| Abrechnung | Im LogYou-Prod-iDempiere: Software-Katalog (Pakete, Features, Add-ons) wird dort gepflegt; Console liest ihn nur, misst Nutzung und erzeugt Rechnungsentwürfe. Lexoffice, SEPA, OPOS unverändert |
| Steuerungsebene | Eigenes Repo `logship-cloud`: Inventar als Quelle der Wahrheit, Ansible-Playbooks für jeden Lebenszyklus-Schritt, eigene Nuxt-4 Cloud Console auf separater VM. Kein Flotten-Code im ERP |

### Vorhanden und wiederverwendet

- Golden-VM 115 + `clone-logship-instance.sh`, Skripte 03/04/05, `sync-from-prod.sh`
- Edge-Proxy VM 116 (Nginx Proxy Manager) + `add-instance.sh`, IONOS-DNS-Plugin
- Jenkins `erp-frontend-deploy-instance`: Build, rsync, pm2-Swap, /signin-Check, Rollback
- Händler-Onboarding-Orchestrator (9 Schritte), Vorlagenrolle 1000006
- Software-Katalog (`CUST_SoftwarePlan`, Katalogpositionen) mit Einstellungsseite; Gebührenrechnungen, Lexoffice, SEPA
- Check_MK, Proxmox Backup Server, Sentry

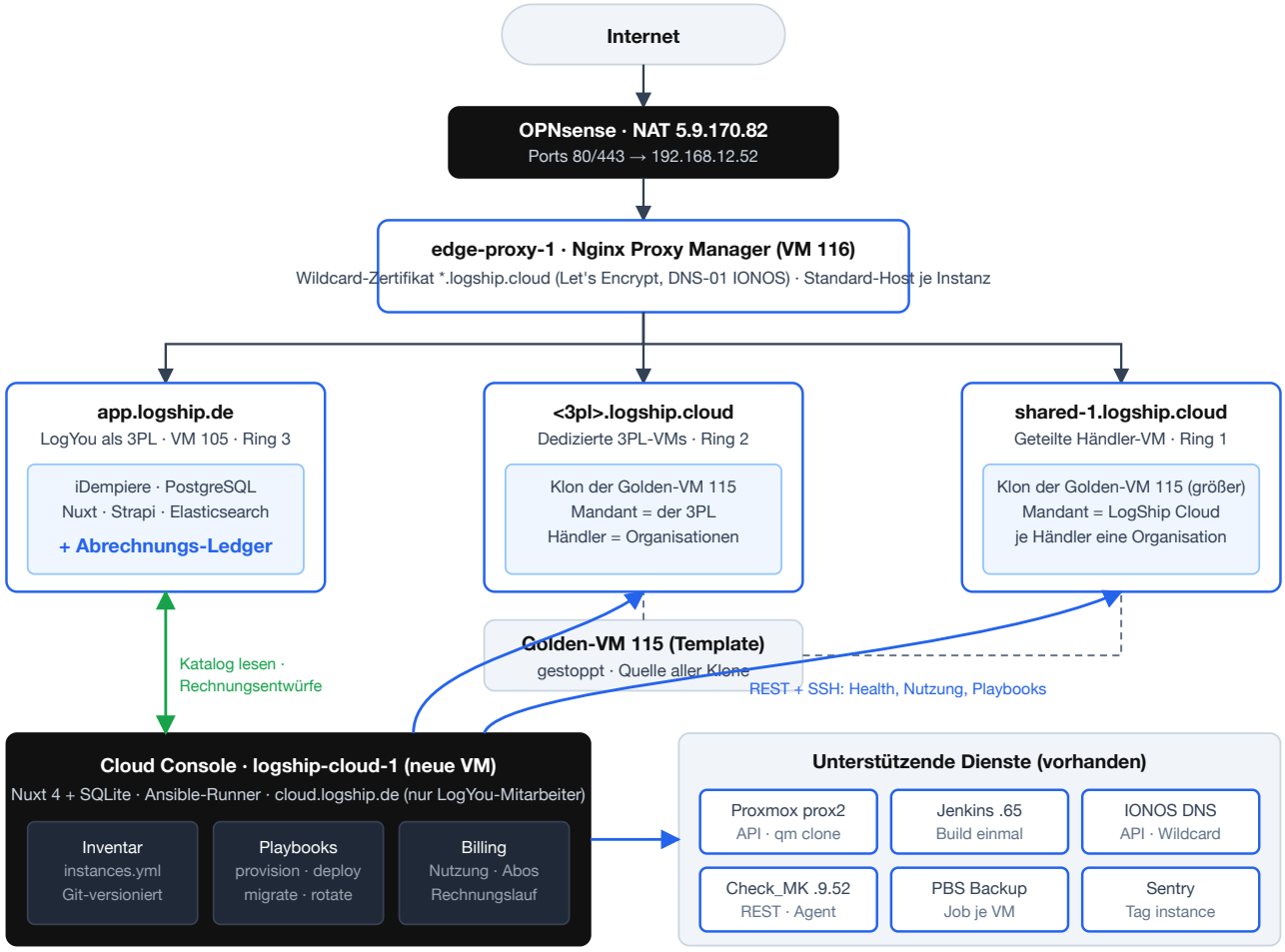
### Lücken, die das Konzept schließt

- Kein Instanz-Register – Wissen verteilt auf READMEs, NPM-Datenbank, `qm list`
- Rund acht Handschritte nach jedem Klon (Zugangsdaten, Secrets, Benutzer, Crons, SMTP, Monitoring, DNS)
- Instanzen erhalten nach dem Klonen **keine** iDempiere-Schema-/Dictionary-Änderungen mehr
- Service-Tokens laufen nach 7 Tagen ab
- Laravel-Middleware kennt nur ein iDempiere; AG-Grid-Lizenz ist Single-Application
- Keine Nutzungsmessung, keine Feature-Steuerung, keine automatische Abrechnung

2 · ZIELARCHITEKTUR

# Organigramm der Zielumgebung

Alle Instanzen stehen hinter dem zentralen Edge-Proxy. Die Cloud Console ist eine eigene, kleine VM außerhalb jeder Kundeninstanz und spricht über APIs mit Proxmox, Proxy, DNS, Jenkins, Monitoring und mit jeder Instanz. LogYou-Prod bleibt gewöhnliche Instanz *und* Buchhaltungs-Ledger.

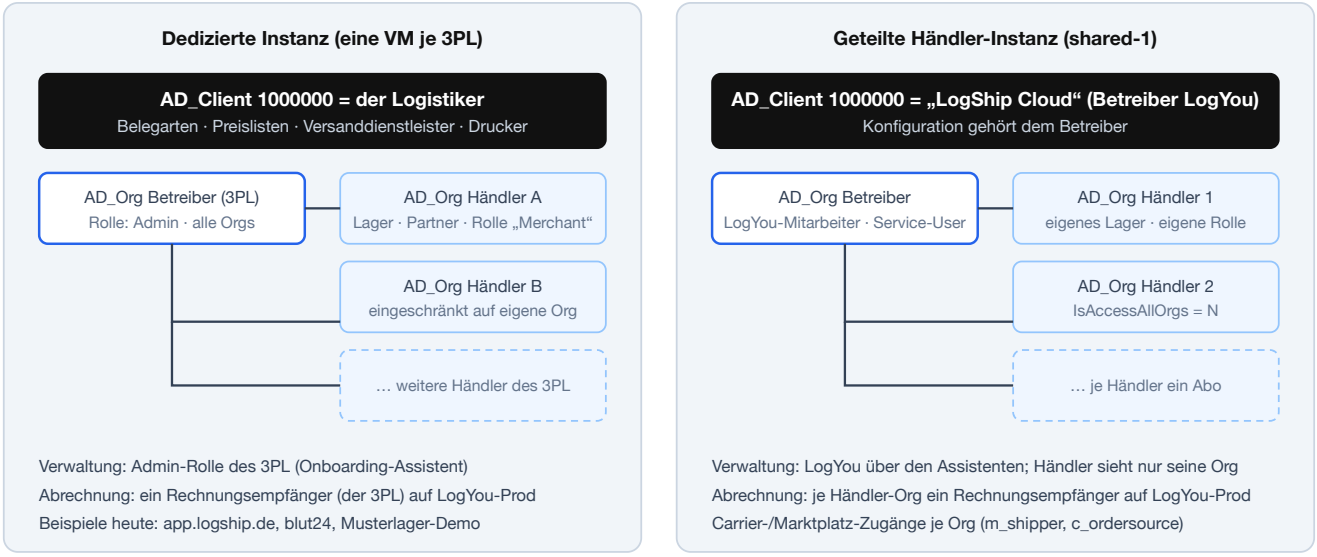



**Legende:** blaue Rahmen = Instanzen bzw. Dienste, dunkel = Netzwerk-/Steuerungsebene, grau = Template und Infrastruktur. Ringe 1–3 = Reihenfolge beim Ausrollen (Kapitel 5). Grüner Doppelpfeil = einziger Berührungspunkt mit LogShip: Katalog lesen, Rechnungen schreiben.

3 · MANDANTENMODELL

# Dedicated und Shared im iDempiere-Datenmodell

Beide Formen nutzen denselben Mandanten 1000000 aus der Golden-VM, weil die Anwendung diesen voraussetzt (Belegarten, Preisliste, Partnergruppe sind je Mandant konfiguriert). Die Trennung der Händler erfolgt über Organisationen und Rollen – das Modell, das heute schon zwischen LogYou und seinen Händlern gilt.




## Regeln für geteilte Instanzen

- **Rollen:** Händlerrollen ohne Zugriff auf alle Organisationen, Org-Zugriff nur auf die eigene. Der Assistent erzeugt das bereits; die bekannte Lücke in Schritt 10 (Org-Zugriff anderer Rollen bleibt stehen, wenn ein eingeschränkter Admin ausführt) wird behoben, indem Schritt 10 mit dem Service-Token läuft.
- **Konfiguration auf Mandantenebene** (Belegarten, Preisliste, Versanddienstleister-Stammdaten) gehört dem Betreiber; Händler ändern sie nicht.
- **Integrationen:** Vor dem Verkauf von Marktplatz-/Carrier-Funktionen auf shared-Instanzen prüfen, dass jede Integration ihre Zugangsdaten aus der Org des Händlers liest, nicht aus mandantenweiten Zeilen.
- **Bewusst nicht gewählt:** ein AD_Client je Händler. Das wäre die stärkere Trennung, erfordert aber eine Umstellung der Anwendung (feste Mandanten-IDs in der Konfiguration) und steht dem Nutzen bei unter zehn Instanzen nicht gegenüber.

4 · STEUERUNGSEBENE

# Repo `logship-cloud`: Inventar, Playbooks, Console

Ein Repo neben `erp-nuxt-frontend`. Das Inventar ist die einzige Quelle der Wahrheit über die Flotte und wird per Git nachvollziehbar geändert. Jede Lebenszyklus-Aktion ist ein Playbook, das dieses Inventar liest. Die Console ist die Bedienoberfläche für genau diese Playbooks – nicht mehr.

```
logship-cloud/
  inventory/
    instances.yml      # Quelle der Wahrheit
    group_vars/        # dedicated / shared / all
  playbooks/
    provision.yml      # Klon → Identität → Env
                       # → Secrets → DNS → Proxy
                       # → Monitoring → Abnahme
    deploy.yml         # Ring-Deploy, ein Build
    migrate.yml        # Schema-Migrationen
    rotate-service-token.yml
    os-update.yml · snapshot.yml
    decommission.yml
  roles/               # proxmox_clone, instance_env,
                       # npm_host, ionos_dns,
                       # checkmk_register, pbs_job,
                       # idempiere_migrate,
                       # frontend_deploy
  migrations/          # 0000-baseline.sql, 0001-…
  console/             # Nuxt 4 Cloud Console
  CLAUDE.md · README.md
```

```
- slug: shared-1
  kind: shared        # dedicated | shared
                      # demo | internal
  vmid: 119
  node: prox2
  ip: 192.168.12.69
  fqdn: shared-1.logship.cloud
  client_name: LogShip Cloud
  company: LogYou GmbH
  sizing: {memory_mb: 24576, cores: 4}
  app_ring: 1         # 0 Demo · 1 Shared
                      # 2 Dedicated · 3 Prod
  schema_version: 2026.11.0
  state: active       # provisioning |
                      # suspended | decommissioned
  created: 2026-11-05
```

**Inventar-Eintrag.** Nur Infrastrukturfakten. Abonnements, Add-ons und Nutzungsdaten liegen in der Console-Datenbank, weil Händler sie später selbst ändern (Kapitel 8).

### Inventar

Jede VM der Flotte, auch Prod, Demo, Edge-Proxy und Dev-VM, bekommt einen Eintrag. Prüfungen (VMID/IP frei, Slug eindeutig) laufen gegen diese Datei *und* gegen Proxmox.

### Playbooks

Phase 1 kapselt die vorhandenen Shell-Skripte (Klon, 03/05/04, add-instance.sh, deploy-instance.sh) unverändert; die Portierung in echte Ansible-Tasks folgt schrittweise. Idempotent: ein zweiter Lauf ändert nichts.

### Console

Startet Playbooks als Prozess auf derselben VM, streamt das Protokoll live, hält den Zustand langer Läufe in SQLite und committet Inventar-Änderungen ins Repo.

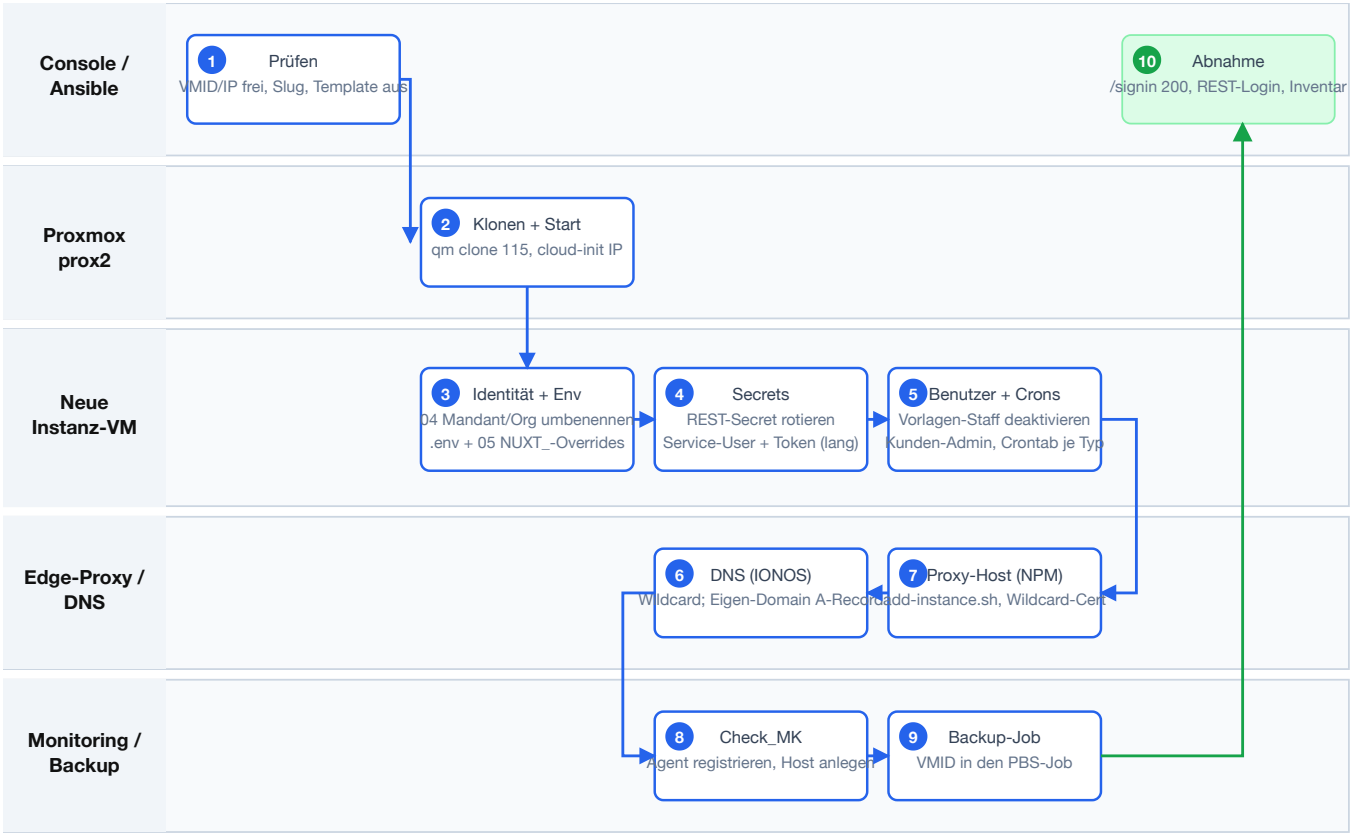
### Grundsatz: Das ERP bleibt schlank

Weder Inventar noch Playbooks noch Nutzungsmessung oder Rechnungslauf berühren den Code von `erp-nuxt-frontend`. Die Console nutzt ausschließlich, was jede Instanz ohnehin bereitstellt: die iDempiere-REST-API und SSH. Einzige geplante Ausnahme ist die Feature-Buchung im Einstellungsbereich (Kapitel 8), die der Nutzer ausdrücklich im Frontend des Händlers haben möchte – und auch dort nur als dünne Seite plus Weiche, die Logik liegt in der Console.

5 · PROVISIONIERUNG

# Von der Golden-VM zur fertigen Instanz in zehn Schritten

Ein Lauf von `provision.yml` ersetzt den heutigen Klon plus rund acht Handschritte. Ergebnis: eine erreichbare, überwachte, gesicherte Instanz mit frischen Secrets und einem Eintrag im Inventar.




## Wichtige Details

- **Env-Overrides (Schritt 3):** runtimeConfig wird beim Build eingebacken; `05-apply-env-overrides.sh` plus `pm2 delete` / `start` bleibt in jedem Pfad Pflicht.
- **Secrets (Schritt 4):** iDempiere nach Rotation wirklich durchstarten lassen, erst dann Token ausstellen (bekannte Falle aus der Demo). Token-Laufzeit je Instanz lang setzen oder wöchentlich rotieren.
- **Wildcard:** `*.logship.cloud` macht DNS und Zertifikat je Instanz zum Nullschritt; nur Eigen-Domains der 3PL brauchen A-Record + eigenes Let's-Encrypt-Zertifikat.

## Danach: Händler auf shared-Instanzen

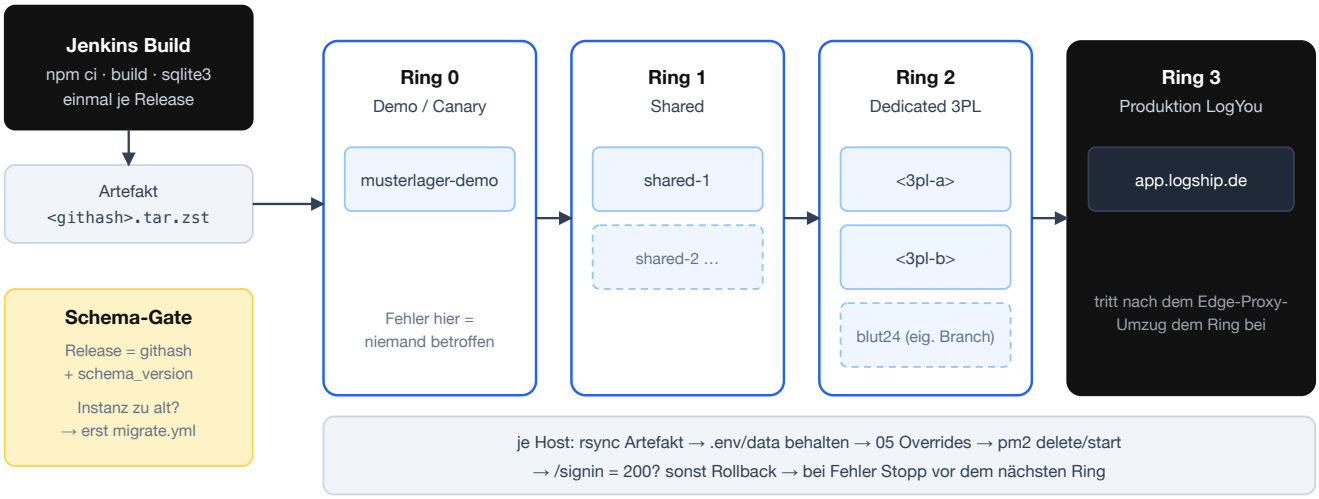
Jeder Händler wird im Onboarding-Assistenten der Instanz angelegt (Partner, Adresse, Org, Benutzer, Lager, Lagerplatz, Rolle). Die Console liest die entstandenen Organisationen live per REST und ordnet sie einem Rechnungsempfänger und Paket zu. Später kann die Landingpage denselben Weg automatisch auslösen.

**Rückbau:** `decommission.yml` = Proxy-Host deaktivieren, letztes Backup, VM stoppen, nach Frist löschen, Inventar-Status `decommissioned`.

6 · AUSROLLEN

# Ein Build, vier Ringe, ein Schema-Gate

Heute baut Jenkins je Zielhost neu. Künftig wird ein Build einmal auf dem Controller erzeugt, als Artefakt abgelegt und in fester Reihenfolge an alle Instanzen verteilt. Zwischen den Ringen stoppt der Lauf beim ersten Fehler; je Host bleiben Swap, /signin-Prüfung und automatischer Rollback wie im bestehenden Skript.




## Release-Begriff

Ein Release ist das Paar `githash` + `schema_version` (Git-Tag im ERP-Repo, kein Code). `deploy.yml` verweigert einen Build, dessen benötigte Schema-Version neuer ist als die der Instanz, bis `migrate.yml` gelaufen ist. Branch `main` = Flotten-Release.

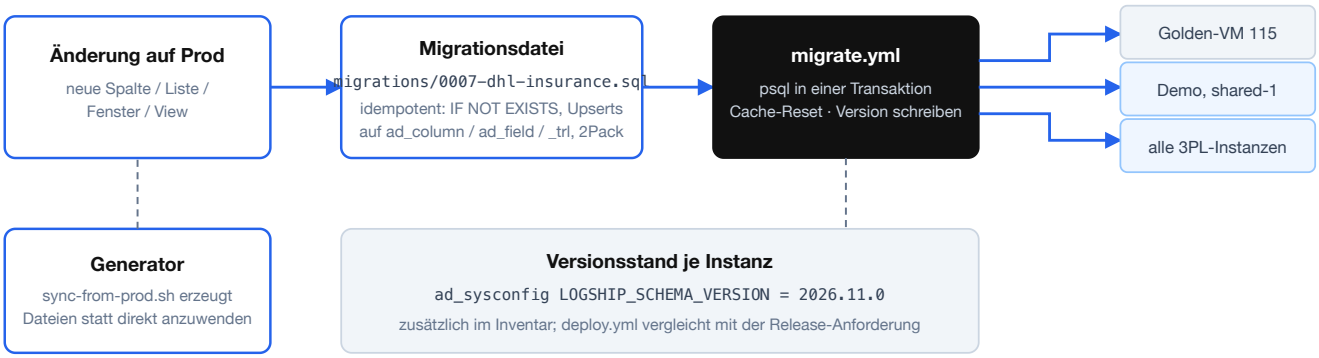
## Übergang

Phase 1 ruft den vorhandenen Jenkins-Job `erp-frontend-deploy-instance` je Ziel auf. Das „Build einmal"-Artefakt und der Ring-Lauf ersetzen ihn, sobald die Flotte mehr als zwei Instanzen hat. Produktion behält ihren eigenen Job bis zum DNS-Umzug auf den Edge-Proxy (Runbook vorhanden).

7 · SCHEMA- UND DICTIONARY-MIGRATIONEN

# Die größte Lücke: Instanzen müssen Änderungen nachziehen

Heute entstehen neue Spalten, Listenwerte und Fenster direkt auf Produktion; die Golden-VM holt sie per `sync-from-prod.sh` nach, bereits geklonte Instanzen nie. Für eine Flotte braucht jede Änderung eine versionierte, wiederholbar anwendbare Datei.




## Regeln

- Baseline `0000-baseline.sql` = aktueller Stand der Golden-VM; alles danach nummeriert.
- Migration immer **vor** dem App-Build, der sie braucht (Release = Paar).
- Ein zweiter Lauf ist ein Nullschritt; Fehler rollen die gesamte Transaktion zurück.
- Nach Dictionary-Änderungen REST-Cache leeren ( `DELETE /api/v1/caches` ), sonst „Spalte existiert nicht“.

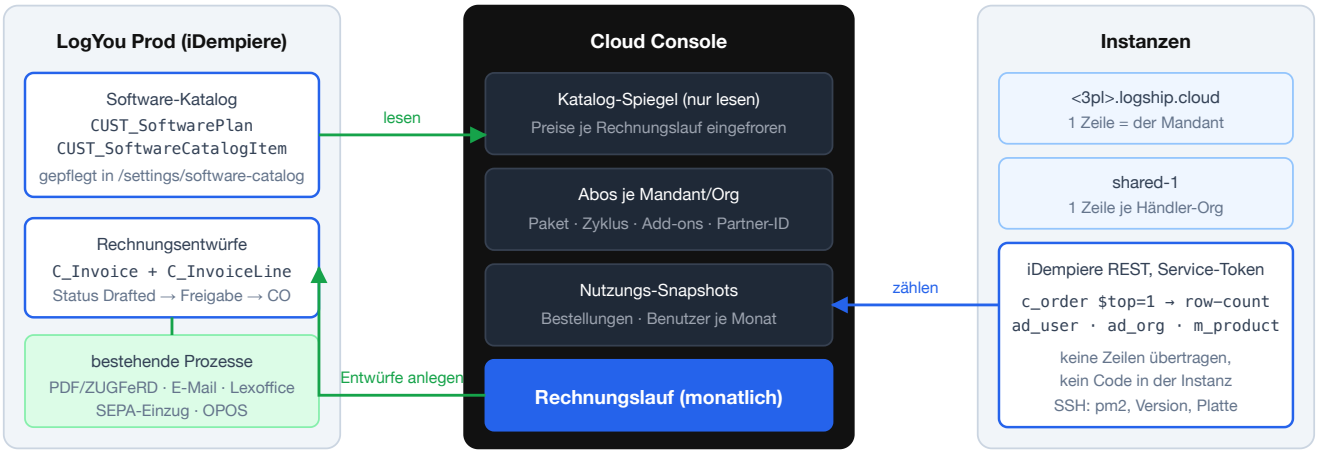
## Was sich dadurch vereinfacht

- Die Golden-VM ist nur noch ein weiteres Ziel; `sync-from-prod.sh` schrumpft auf OS-/Paket-Drift.
- Template-Refresh wird zur Routine statt zum 45-Minuten-Eingriff.
- Konfigurationszeilen, die heute nur „gemeldet“ werden (Fenster, Menüs, Prozesse), werden als 2Pack mitgeführt.

8 · ZENTRALE ABRECHNUNG

# Katalog in LogShip, Messung in der Console, Rechnung auf Prod

Pakete, Features, Add-ons und Preise werden weiterhin in LogShip gepflegt (Einstellungen → Software-Katalog). Die Console liest diesen Katalog nur, misst die Nutzung jeder Instanz bzw. Händler-Org per REST-Zähler, rechnet und legt je Rechnungsempfänger einen Rechnungsentwurf auf LogYou-Prod an. Ab dort gelten die vorhandenen Prozesse.




```
Betrag je Mandant und Monat =
Grundpreis (PriceMonthly bzw. PriceYearly / 12)
+ max(0, Bestellungen − IncludedOrders) × ExtraOrderPrice
+ max(0, aktive Benutzer − IncludedUsers) × ExtraUserPrice
+ Σ gebuchte Add-ons (PriceUnit = Monat, anteilig im Buchungsmonat)
```

Alle Feldnamen existieren bereits in `CUST_SoftwarePlan`; die Formel bildet genau das heutige Paketmodell ab: Monatspreis mit inkludierten Bestellungen, jede weitere Transaktion zum festen Satz.

8 · ZENTRALE ABRECHNUNG · DEFINITIONEN

# Was gezählt wird und wie die Rechnung aussieht

| MESSGRÖSSE | DEFINITION | QUELLE |
|---|---|---|
| **Bestellungen** | Verkaufsaufträge, die im Monat abgeschlossen wurden: `IsSOTrx = Y`, `DocStatus` CO oder CL, `DateOrdered` im Zeitraum, Org = Händler-Org. Erzeugte Gebührenaufträge ( `IsFulfillmentOrder = N` ) zählen nicht. | REST `models/c_order`, `$top=1`, Wert aus `row-count` |
| **Aktive Benutzer** | Benutzer der Org mit Passwort und mindestens einer Rolle; Service-Benutzer ausgenommen. | REST `models/ad_user` |
| **Paket / Zyklus** | Zuordnung je Mandant bzw. Händler-Org; monatlich oder jährlich (Jahrespreis / 12 je Monat ausgewiesen oder jährlich in einer Rechnung). | Console-Datenbank (Abo) |
| **Add-ons** | Katalogpositionen mit `PriceUnit = Monat`, Zeitraum aus der Buchung (Kapitel 9). Einmal-/Stundenpositionen bleiben manuell. | Katalog + Console-Abo |
| **Individualpreis** | `IncludedOrders = 0` oder `PriceMonthly = 0` bedeutet „individuell“: die Console verlangt einen manuellen Preis am Abo und rechnet sonst nicht ab. | Console-Abo |

## Beispiel: Rechnungsentwurf für einen Händler auf shared-1 (Oktober 2026)

| POSITION | TEXT | MENGE | EINZELPREIS | BETRAG |
|---|---|---|---|---|
| 1 | LogShip Cloud Abo „Business“ · 01.–31.10.2026 · shared-1 / Org Muster Shop | 1 | 189,00 € | 189,00 € |
| 2 | Zusätzliche Bestellungen (1 240 – 1 000 inkludiert) | 240 | 0,19 € | 45,60 € |
| 3 | Zusätzliche Benutzer (7 – 5 inkludiert) | 2 | 9,00 € | 18,00 € |
| 4 | Zusatzfunktion „Retourenportal“ · 12.–31.10.2026 (20/31 Tage) | 1 | 29,00 € | 18,71 € |
| | | | **Netto** | **271,31 €** |

Preise sind Platzhalter aus dem aktuellen Katalog-Default. Jede Zeile trägt den beim Lauf gültigen Katalogpreis als Snapshot (Nachvollziehbarkeit bei späteren Preisänderungen).

### Ablauf des Rechnungslaufs

1. Katalog von Prod lesen und einfrieren
2. Nutzung aller aktiven Instanzen/Orgs zählen, Snapshot speichern
3. Positionen je Rechnungsempfänger bilden, Vorschau in der Console
4. Je Empfänger eine `C_Invoice` als Entwurf anlegen
5. Freigabe auf der Rechnungsseite; Versand, Lexoffice, SEPA wie heute

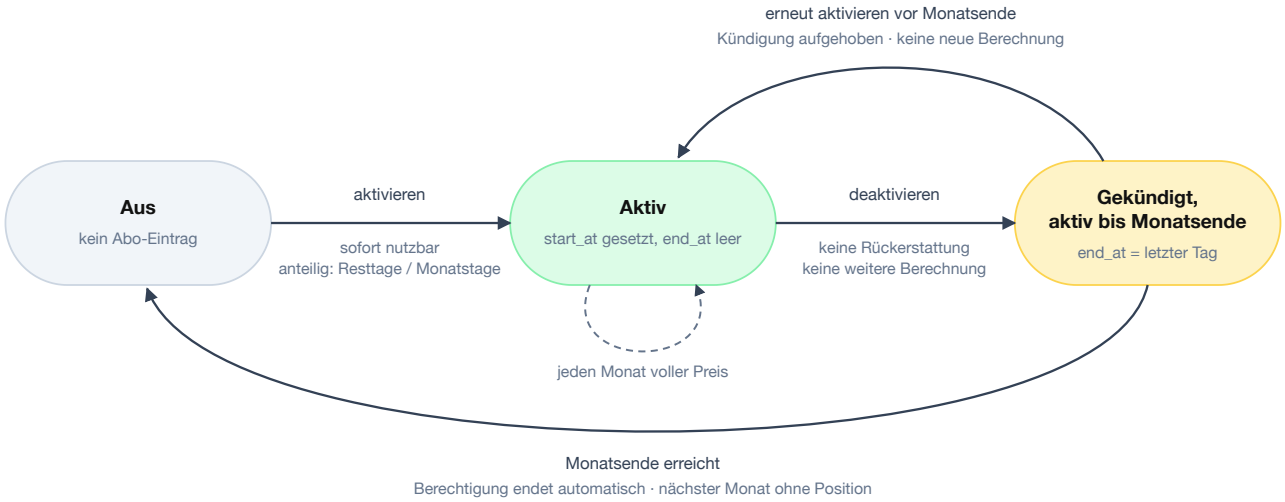
### Sicherungen

- **Idempotenz:** Console merkt sich `C_Invoice_ID` je Mandant und Monat; ein zweiter Lauf legt nichts doppelt an.
- **Nur Entwürfe:** Die Console schließt keine Rechnung ab; der Mensch gibt frei.
- **Partner-Zuordnung:** Rechnungsempfänger ist der Partner auf Prod, den der Lead-/Angebotsprozess ohnehin anlegt.
- **Rundung:** zwei Nachkommastellen je Position, kaufmännisch.

9 · FEATURE-BUCHUNG DURCH DEN KUNDEN

# Zuschalten, anteilig zahlen, bis Monatsende behalten

Künftig schaltet ein Kunden-Admin kostenpflichtige Funktionen im Einstellungsbereich selbst ein und aus. Eine Funktion wird monatlich berechnet; bei Aktivierung mitten im Monat nur anteilig nach Tagen; nach Deaktivierung bleibt sie bis zum Ende des bezahlten Monats aktiv. Noch nicht umgesetzt – hier als Zustandsmodell festgehalten, damit Daten und Abrechnung von Anfang an dazu passen.




## Rechenregeln

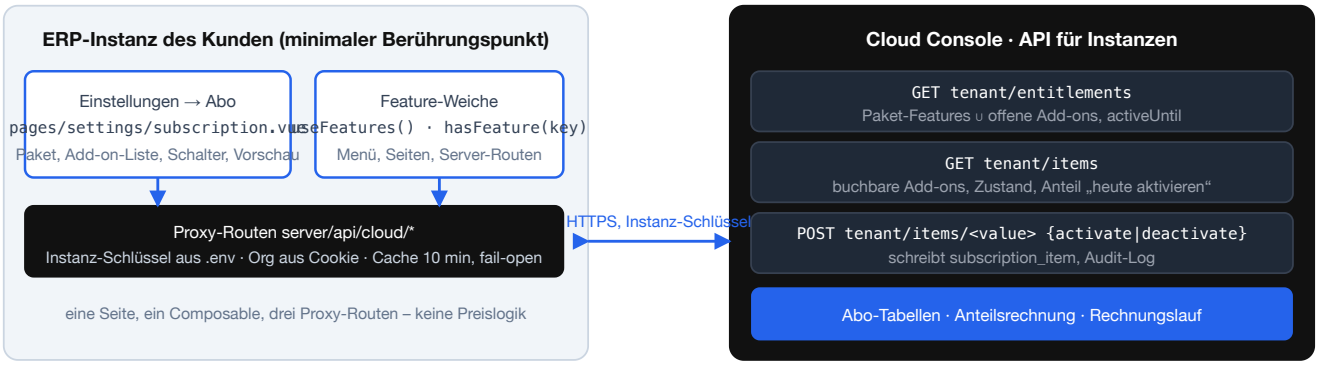
- **Anteil im Aktivierungsmonat:** Preis × verbleibende Tage / Kalendertage des Monats; der Aktivierungstag zählt mit (Zeitzone Berlin).
- **Folgemonate:** voller Monatspreis, solange kein `end_at` gesetzt ist.
- **Kündigung:** `end_at` = letzter Tag des laufenden Monats; der Monat ist bereits bezahlt oder wird voll berechnet; danach nichts mehr.
- **Reaktivierung vor `end_at`:** Kündigung wird gelöscht, kein neuer Eintrag, keine zweite Position.
- **Paketwechsel** (später): Upgrade sofort mit anteiliger Differenz, Downgrade zum 1. des Folgemonats.

## Beispiel

| DATUM | AKTION | WIRKUNG |
|---|---|---|
| 12.10. | „Retourenportal“ aktivieren | sofort aktiv · Okt.: 29 € × 20/31 = 18,71 € |
| 01.11. | – | Nov.: 29,00 € |
| 18.11. | deaktivieren | aktiv bis 30.11. · keine Erstattung |
| 25.11. | erneut aktivieren | Kündigung aufgehoben · Dez.: 29,00 € |
| 03.12. | deaktivieren | aktiv bis 31.12. · Januar: keine Position, Funktion aus |

9 · FEATURE-BUCHUNG · TECHNIK

# Logik in der Console, dünne Oberfläche im ERP




## Vertrag mit dem Katalog

- Der Feature-Schlüssel ist `CUST_SoftwareCatalogItem.Value`; das ERP prüft genau diesen Schlüssel. `Value` gilt als unveränderlich.
- Neue buchbare Funktion = Katalogposition auf Prod anlegen (Preis, Einheit Monat, Mindestpaket) und im ERP an der Stelle, die sie schützt, `hasFeature('…')` prüfen.
- Paket-Features (Mindestpaket ≤ gebuchtes Paket) sind ohne Buchung berechtigt; Add-ons nur mit offenem Abo-Eintrag.

## Robustheit

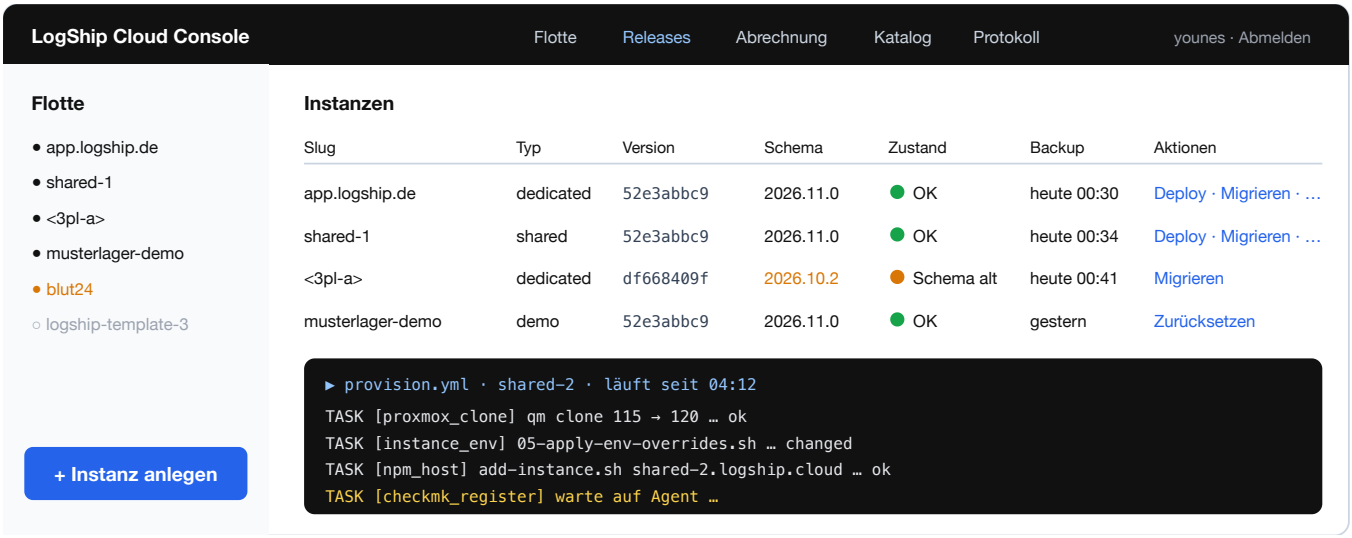
- Berechtigungen werden in der Instanz zwischengespeichert; ist die Console nicht erreichbar, gilt der letzte bekannte Stand (kein Ausfall beim Kunden).
- Der Instanz-Schlüssel verlässt den Server nie; der Browser spricht nur mit der eigenen Instanz.
- Menü ausblenden ist nicht die einzige Sperre – Server-Routen prüfen dieselbe Berechtigung.
- Jede Buchung landet im Audit-Log der Console (Benutzer, Org, Zeitpunkt, Aktion).

**Einordnung:** Das ist die einzige geplante Code-Berührung des ERP für den SaaS-Betrieb, ausdrücklich gewünscht, weil der Kunde die Buchung in seinem Frontend vornehmen soll. Sie kommt in Phase 7, nachdem Flotte, Migrationen und Rechnungslauf stehen.

10 · CLOUD CONSOLE

# Eine kleine Nuxt-4-Anwendung auf eigener VM

Nuxt 4 mit Nuxt UI (keine AG-Grid-Lizenzfrage), SQLite im `data/`-Verzeichnis, Zugang nur für LogYou-Mitarbeiter (YubiKey/WebAuthn, LAN/VPN bzw. IP-Allowlist am Edge-Proxy). Läuft mit pm2 wie die Instanzen, hat eigenen Backup-Job und Monitoring. VM-Vorschlag: `logship-cloud-1`, 2 vCPU / 4 GB, Debian 12 cloud-init.




## Seiten

- Flotte: Inventar + Live-Zustand (Version, Schema, Platte, Backup-Alter, Zertifikat)
- Instanz: Händler-Orgs, Nutzung, Tokens, Aktionen
- Releases: githash je Ring
- Abrechnung: Vorschau → Entwürfe
- Protokoll: jede Aktion mit Wer/Wann

## Aktionen

- Instanz anlegen (provision)
- Ring ausrollen (deploy)
- Migrieren (migrate)
- Token rotieren
- Snapshot · Pausieren/Fortsetzen (Proxy-Host aus + VM stopp)
- Rückbau

## Technik

- Playbook als Kindprozess, Log per SSE live
- Lauf-Zustand in SQLite, überlebt Neustart
- Inventar-Änderungen werden ins Repo committet
- Secrets (Tokens, API-Schlüssel) verschlüsselt auf der Console-VM

11 · FAHRPLAN

# Sieben Phasen, jede für sich nutzbar




Erst Flotte beherrschen (1–4), dann Geld (5), dann Komfort für den Kunden (7). Jede Phase endet mit einer prüfbaren Abnahme.

| # | PHASE | INHALT | ABNAHME |
|---|---|---|---|
| 1 | Fundament | Repo `logship-cloud`, Inventar der heutigen Flotte (Prod 105, blut24 114, Demo 118, Edge 116, Dev 130), `provision.yml` und `deploy.yml` als Hülle um die vorhandenen Skripte, Token-Rotation; shared-1 damit anlegen; Wildcard-DNS und -Zertifikat; app.logship.de auf den Edge-Proxy umziehen | shared-1 ohne Handschritt erreichbar, überwacht, gesichert; Eintrag im Inventar |
| 2 | Migrationen | Baseline, nummerierte Dateien, `migrate.yml`, Schema-Version in `ad_sysconfig`, Generator aus `sync-from-prod.sh` | Test-Spalte auf Demo + shared-1, zweiter Lauf ohne Änderung |
| 3 | Messung | REST-Zähler und SSH-Checks je Instanz in die Console-Datenbank | Zahlen einmal von Hand gegen die Demo geprüft |
| 4 | Console v1 | VM, Flottenliste, Instanzseite, Aktionen mit Live-Protokoll, Monitoring/Backup der Console selbst | Alle Aktionen aus der Oberfläche, Protokoll vollständig |
| 5 | Abrechnung | Katalog-Spiegel, Abos, Rechnungslauf mit Vorschau, Entwürfe auf Prod | Ein echter Entwurf für den Testpartner, dann storniert |
| 6 | Härtung | OS-Update-Playbook, Check_MK-Agenten in allen Gästen, Sentry je Instanz, vierteljährlicher Template-Refresh | Flotte auf gleichem Patch- und Schemastand |
| 7 | Feature-Buchung | Console-API, Abo-Positionen mit Anteilsrechnung, ERP-Seite + Weiche + Proxy-Routen, erste geschützte Funktion | Aktivieren am 12. → 20/31; Kündigung bis Monatsende aktiv; Console aus → Cache greift |

12 · OFFENE PUNKTE UND GRENZEN

# Was vor dem ersten zahlenden SaaS-Kunden geklärt sein muss

| THEMA | SACHSTAND | EMPFEHLUNG | STUFE |
|---|---|---|---|
| **Laravel-Middleware** | data.logship.de kennt genau ein iDempiere. Marktplatz-Importe, Versandbestätigungen und Finanzimporte laufen nur für Prod. | Mandantenfähig machen (Instanz → URL + Token, Queue je Instanz), sobald ein SaaS-Kunde Marktplätze braucht; sonst Laravel je dedizierter VM | hoch |
| **AG-Grid-Lizenz** | Single-Application-Lizenz für „logyou“, Version 35.2.1 festgepinnt. | Beim Anbieter bestätigen, dass mehrere Deployments derselben Anwendung abgedeckt sind; blut24 als eigene Marke vermutlich nicht | mittel |
| **REST-Token-Laufzeit** | Service-Tokens laufen nach 7 Tagen ab; Hintergrundaufrufe brechen dann ab. | Je Instanz lange Laufzeit für den Service-Benutzer oder wöchentliche Rotation aus der Console | mittel |
| **Feste Mandanten-IDs** | Belegart, Preisliste, Partnergruppe kommen aus der Umgebung und sind auf allen Klonen identisch. | Unkritisch, solange die Golden-VM diese IDs nie neu vergibt – als Regel festhalten | niedrig |
| **Kapazität** | Rund 20 GB RAM je vollständiger Instanz (JVM, Elasticsearch, Strapi, Node). prox2 trägt eine Handvoll. | shared-1 mit 24–32 GB; ab etwa zehn Instanzen ES je Knoten teilen oder kleine Stufe ohne ES; auf prox1/prox2 verteilen | mittel |
| **Integrationen je Org** | Nicht alle Carrier-/Marktplatz-Zugriffe sind nachweislich org-gebunden. | Vor dem Verkauf dieser Funktionen auf shared-Instanzen jede Integration prüfen | mittel |
| **Edge-Proxy-Umzug** | Produktionshostnamen zeigen noch auf ISPConfig (.81); nur edge.logship.de läuft über NPM. | Umzug je Hostname nach vorhandenem Runbook; Rückweg = DNS zurück | niedrig |
| **Self-Service** | Landingpage → Instanz ist für blut24 gekoppelt (Service-Token-Gleichheit), für LogShip nicht. | Später über die Console-API, nicht direkt Landingpage → Instanz | später |

## Bewusst nicht Teil des Konzepts

- Kubernetes / Container: kein Nutzen bei unter zehn VMs, hoher Umbau der Golden-VM-Pipeline.
- Ein AD_Client je Händler: stärkere Trennung, aber Anwendungsumbau.
- Stripe als Rechnungssteller: iDempiere bleibt Ledger; Zahlungsarten später ergänzbar.
- Terraform neben Ansible: ein Werkzeug reicht.

## Nächster Schritt

Freigabe dieses Entwurfs, dann Phase 1: Repo anlegen, Inventar der heutigen Flotte erzeugen, shared-1 als erste vollautomatisch provisionierte Instanz aufsetzen. Keine Phase verlangt Änderungen am Produktivsystem außer dem geplanten Edge-Proxy-Umzug.