XINITY

WHITEPAPER

On-Premises-Enterprise-Inferenz, entwickelt für Souveränität.

Die Open-Source-Plattform für den Betrieb großer Sprachmodelle innerhalb Ihrer eigenen Infrastruktur. OpenAI-kompatibel, Multi-Node, produktionsreif. Kein zwingender Datenabfluss.

Souverän durch Architektur, nicht durch Vertrag.

KI DORT BETREIBEN, WO IHRE DATEN LIEGEN.

Inhalt

01 Management Summary 03

02 Die On-Premises-Herausforderung 04

03 Architekturüberblick 05

04 Load Balancing & Durchsatz 08

05 Multi-Node-Skalierung 09

06 Hochverfügbarkeit & 24/7-Betrieb 10

07 Monitoring & Observability 11

08 Sicherheit & Compliance 13

09 Deployment-Optionen 15

10 Fazit 16

ABSCHNITT 01

Management Summary

Xinity AI ist eine quelloffene On-Premises-Plattform für die Verwaltung und Bereitstellung großer Sprachmodelle (LLMs) in Unternehmensumgebungen, in denen Daten die eigene Infrastruktur nicht verlassen dürfen. Entwickelt für Branchen wie Medien, Industrie, Gesundheitswesen, Finanzwesen und öffentliche Einrichtungen, liefert Xinity die vollständige Betriebsschicht rund um die Modell-Inferenz: eine OpenAI-kompatible API, ein Management-Dashboard, Multi-Node-GPU-Orchestrierung, Load Balancing, Observability und Zugriffskontrolle auf Enterprise-Niveau. Und das alles mit der Garantie, dass keine Daten abfließen.

Dieses Whitepaper beschreibt die technische Architektur, die Performance-Eigenschaften und das Betriebsmodell von Xinity AI. Im Mittelpunkt stehen der Vergleich mit dem direkten Betrieb von Inference-Engines wie Ollama sowie die Unterstützung hochverfügbarer 24/7-Produktivumgebungen.

Zentrale Unterscheidungsmerkmale

/01 Das Beste aus beiden Welten

Xinity verbindet die einfache Modellverwaltung von Ollama mit dem Durchsatz, der Geschwindigkeit und der Robustheit von vLLM – pro Deployment frei wählbar.

/02 Transparent, auditierbar und on-premises

Der gesamte Code ist quelloffen und wird selbst gehostet. Es verlassen keine Daten die Infrastruktur des Kunden.

/03 Produktionsreifer Betrieb

Integriertes Load Balancing, Canary-Deployments, Observability und Multi-Node-Skalierung. Diese Funktionen gehen weit über das hinaus, was eine Inference-Engine für sich genommen bietet.

ABSCHNITT 02

Die On-Premises-Herausforderung

Cloud-KI-Plattformen setzen voraus, dass Daten nach außen übertragen werden dürfen. Für viele Unternehmen trifft diese Annahme nicht zu – nicht aus Präferenz, sondern von Gesetzes wegen. DSGVO, Bankgeheimnis, journalistischer Quellenschutz, Anwaltsgeheimnis und Vorschriften zum Schutz von Geschäftsgeheimnissen machen die externe Datenverarbeitung für bestimmte Workloads rechtlich unmöglich.

Organisationen in diesen Branchen sind nicht cloud-skeptisch. Sie sind souveränitätsblockiert.

Sie benötigen die Fähigkeiten moderner KI-Plattformen – von Model Serving und API-Integration über Observability und Zugriffsverwaltung bis hin zum Data Labeling –, ohne dass Daten ihre Hardware verlassen. Xinity löst das mit einer vollständigen, selbst hostbaren KI-Plattform:

- Modell-Orchestrierung und Deployment-Management
- OpenAI-kompatible API, die mit bestehendem Client-Code funktioniert
- Multi-Node-GPU-Scheduling und Load Balancing
- Web-basiertes Management-Dashboard mit RBAC
- Nutzungserfassung, Datensammlung und Labeling-Pipelines
- SSO, 2FA und Enterprise-Authentifizierung
- Prometheus-Metriken und Grafana-Dashboards

> DIE UMKEHR DER AUSLASTUNG
>
> Cloud-Preismodelle gehen von rund 15 % Auslastung bei schwankenden Workloads aus. Für dauerhaft laufende interne KI-Agenten mit 80 bis 90 % Auslastung liefert dedizierte On-Premises-Infrastruktur rund **80 % Kostenersparnis** gegenüber vergleichbarer Cloud-Kapazität. Souveränität ist nicht nur eine Compliance-Anforderung. Sie ist ein wirtschaftlicher Vorteil.

ABSCHNITT 03

Architekturüberblick

Xinity AI ist eine modulare Plattform aus unabhängig deploybaren Diensten. Die Control Plane koordiniert sich über eine gemeinsame PostgreSQL-Datenbank; die Inferenzknoten sind über den Tether angebunden.

3.1 Komponentenübersicht

KOMPONENTE	RUNTIME	ROLLE
Gateway	Bun-natives HTTP	Öffentlicher API-Einstiegspunkt; Load Balancing; Request-Routing; Nutzungsprotokollierung
Dashboard	SvelteKit 2 + Bun	Management-UI; oRPC-API; Deployment-Orchestrierung; Data Labeling
Tether	Bun	Verbindungsschicht zwischen Control Plane und Inferenzknoten; Selbstregistrierung der Daemons; Zustandsverteilung und Fortschrittsmeldungen
Daemon	Bun	Läuft auf jedem Inferenzknoten; Modell-Lifecycle-Management; automatische GPU-Erkennung
Info Server	Bun	Zustandsloser Modellkatalog-Dienst; YAML-basierte Modelldefinitionen
PostgreSQL	Postgres 17	Gemeinsame Datenbank für die Control-Plane-Koordination
Redis	Redis 7	Flüchtiger Gateway-Zustand (Auth-Cache, Load Balancer, Response Store)

3.2 Ollama + vLLM: das Beste aus beiden Welten

Xinity AI unterstützt sowohl **Ollama** als auch **vLLM** als Inferenz-Backends, sodass Betreiber für jeden Anwendungsfall die passende Engine wählen können:

- **Ollama** bietet die Bequemlichkeit und Agilität, Modelle mit minimaler Konfiguration aus einer öffentlichen Registry zu beziehen und auszuführen. Ideal für schnelles Prototyping, kleinere Workloads und Umgebungen, in denen betriebliche Einfachheit im Vordergrund steht.
- **vLLM** liefert deutlich höheren Durchsatz und bessere Performance unter Last, mit Funktionen wie PagedAttention, Continuous Batching und Tensor-Parallelität. Es ist die Engine der Wahl für performancekritische Produktiv-Workloads und große Modelle.

Betreiber können eine der beiden Engines oder beide parallel betreiben. Der Daemon von Xinity verwaltet den vollständigen Lebenszyklus beider Backends, und das Gateway leitet Anfragen transparent an diejenige Engine weiter, die das angeforderte Modell bereitstellt. Diese Flexibilität erlaubt es Teams, mit Ollama agil zu starten und für die Skalierung auf vLLM zu wechseln – ohne Änderungen am Anwendungscode oder an der API-Schnittstelle.

3.3 Der Tether

Der Tether ist die Verbindungsschicht zwischen der Control Plane und den Inferenzknoten. Daemons halten keine eigene Datenbankverbindung mehr. Sie registrieren sich beim Start selbsttätig am Tether und halten eine dauerhafte Verbindung offen, über die sie ihren Soll-Zustand empfangen und ihren Ist-Zustand zurückmelden.

- **Verbindungen statt Datenbanksitzungen:** Zuvor war jeder Daemon ein eigener PostgreSQL-Client, und die Größe der Inferenzflotte war an die Verbindungsgrenzen der Datenbank gebunden. Der Tether bündelt mehrere Tausend Daemon-Verbindungen auf einen kleinen, festen Pool von Datenbankverbindungen – standardmäßig zehn aus Redundanzgründen, technisch genügt eine.
- **Ausfälle werden zuverlässig erkannt:** Schlägt eine Schreiboperation auf eine Verbindung fehl, bleibt sie länger als das Liveness-Zeitfenster stumm (standardmäßig 45 Sekunden) oder wird die Datenbank unerreichbar, trennt der Tether die Verbindung und führt den Knoten als nicht verfügbar. Der Daemon verbindet sich mit exponentiellem Backoff neu. Verwaiste Registrierungen nach einem harten Ausfall bleiben nicht zurück.

Die Verbindung baut der Daemon auf, nicht umgekehrt. Seine Registrierung – Knoten-ID, Adresse, GPUs, Treiberversionen, Kapazität und sein öffentlicher Schlüssel – ist der Rumpf einer langlebigen Anfrage, die der Tether als Event-Stream offen hält. Über diesen Stream empfängt der Knoten seinen vollständigen Soll-Zustand, sobald PostgreSQL eine Änderung meldet, sowie dynamische Konfigurationswerte aus dem Dashboard, die versiegelt durchgereicht werden. Fortschritt, Fehler und Protokolle meldet der Daemon signiert zurück; der Tether schreibt sie gebündelt in die Datenbank.

Für die Sicherheitsarchitektur heißt das: Auf den GPU-Knoten liegen keine Datenbank-Zugangsdaten mehr. Jeder Knoten registriert sich mit einem eigenen Schlüsselpaar, dessen öffentlichen Teil der Tether bei der ersten Registrierung fest an die Knoten-ID bindet. Ein Daemon kann die Identität eines anderen nicht übernehmen. Im Inferenzpfad liegt der Tether nicht; er trägt ausschließlich Steuerungsinformationen.

> **VON ZEHNERN AUF TAUSENDE**
>
> Die Obergrenze der Flottengröße lag früher nicht bei der Rechenleistung, sondern bei den Datenbankverbindungen: je nach Dimensionierung der PostgreSQL-Instanz 10 bis 40 Inferenzknoten. Seit die Knoten über den Tether angebunden sind, entfällt diese Grenze. Ein Tether ist darauf ausgelegt, mehrere Tausend Daemons zu bedienen. Skaliert wird in Knoten, nicht in Datenbanksitzungen.

3.4 Kommunikationsmodell der Dienste

Die Control-Plane-Koordination folgt einem **Shared-Database-Pattern**, die Anbindung der Inferenzknoten läuft über den Tether:

- Das **Dashboard** schreibt Deployment-Pläne (`modelInstallation`-Zeilen) nach PostgreSQL und löst über PostgreSQL `LISTEN/NOTIFY` sofortige Aktualisierungen aus.
- Der **Tether** liest diese Pläne und verteilt sie an die betroffenen Knoten. Der **Daemon** auf jedem GPU-Knoten empfängt seinen Soll-Zustand über die offene Tether-Verbindung, verwaltet Modellinstallationen (Pull über Ollama oder Start über vLLM) und meldet den tatsächlichen Zustand (`modelInstallationState`) auf demselben Weg zurück. Eine direkte Datenbankverbindung besteht nicht.
- Das **Gateway** liest die Datenbank für API-Key-Verifizierung, Modellauflösung und Host-Auswahl. Für Inferenzanfragen ist es der einzige Dienst, der direkt mit dem Daemon spricht: Anfragen werden über HTTP(S) an den Zielknoten weitergeleitet. Die Steuerungsebene läuft dagegen vollständig über den Tether.

Dieses Design hält jeden Dienst schlank, unabhängig deploybar und widerstandsfähig gegenüber dem Ausfall einzelner Dienste.

3.5 Ablauf einer Anfrage

01 Der Client sendet eine Anfrage an das Gateway (z. B. `POST /v1/chat/completions`).

02 Das Gateway authentifiziert die Anfrage per API-Key (in Redis zwischengespeichert).

03 Das angeforderte Modell wird über die Deployment-Registry aufgelöst.

04 Ein Inferenzknoten wird per Load Balancing ausgewählt.

05 Die Anfrage wird an den Ziel-Daemon weitergeleitet, der sie an den lokalen Inferenztreiber (Ollama oder vLLM) durchreicht.

06 Die Antwort wird über denselben Pfad zurück an den Client gestreamt.

07 Nutzungsereignisse und – optional – vollständige Request-/Response-Daten werden in PostgreSQL protokolliert.

ABSCHNITT 04

Load Balancing & Durchsatz

4.1 Overhead des Gateways

Die Gateway-Schicht ergänzt die reine Inferenz-Pipeline um Enterprise-Funktionen (Authentifizierung, Routing, Nutzungsprotokollierung, Retries, Admission Control). Sie ist darauf ausgelegt, minimalen Overhead zu verursachen und zugleich erheblichen betrieblichen Nutzen zu stiften. In Benchmark-Tests war der Token-Durchsatz pro Stream über das Gateway identisch mit dem der zugrunde liegenden Inference-Engine – das Gateway wird also nicht zum Flaschenhals.

Unter hoher Last steigert das Gateway den Durchsatz sogar gegenüber dem direkten Aufruf einer Inference-Engine. Die begrenzte Annahme von Anfragen hält die Inference-Engine in ihrem effizienten Arbeitsbereich, und die Retry-Schicht fängt transiente Fehler ab und stellt sie transparent erneut zu, bevor der Client überhaupt einen Fehler zu sehen bekommt. Das Ergebnis sind deutlich höhere Abschlussraten bei dauerhaft hoher Parallelität. Genau das unterscheidet ein nutzbares Produktivsystem von einem, das unter realem Traffic zusammenbricht.

4.2 Load-Balancing-Strategien

Das Gateway unterstützt drei Load-Balancing-Strategien, konfigurierbar über `LOAD_BALANCE_STRATEGY`:

STRATEGIE	VERHALTEN
least-connections `(Standard)`	Wählt den Knoten mit den wenigsten laufenden Anfragen, erfasst in Redis. Fällt bei Redis-Fehlern auf `random` zurück.
round-robin	Atomarer Redis-Zähler pro Modell, rotiert durch die Knoten.
random	Gleichverteilte zufällige Auswahl.

4.3 Prefix-Cache-Affinität

Für die Strategien `least-connections` und `random` implementiert das Gateway **Prefix-Cache-Affinität**: Präfixe von Konversationsnachrichten werden gehasht und in Redis Knoten zugeordnet (TTL: 5 Minuten). So werden wiederkehrende Konversationen an denselben Knoten geleitet, was die KV-Cache-Trefferquote der Inference-Engine erhöht. Bei `least-connections` wird die Affinität nur berücksichtigt, wenn der vorgemerkte Knoten innerhalb von 2 Verbindungen des am wenigsten ausgelasteten Knotens liegt.

4.4 Canary-Deployments

Xinity unterstützt schrittweise Modell-Rollouts über Canary-Deployments:

- **Manueller Modus:** Der Traffic wird zu einem festen Prozentsatz aufgeteilt, bis manuell weitergeschaltet wird.
- **Zeitbasierter Modus:** Der Fortschritt interpoliert linear über ein konfiguriertes Zeitfenster und leitet nach dessen Ablauf automatisch den gesamten Traffic auf das primäre Modell.

So lassen sich neue Modellversionen ohne Ausfallzeit sicher im Produktivbetrieb A/B-testen.

ABSCHNITT 05

Multi-Node-Skalierung

5.1 Deployment-Strategien

Der Deployment-Sync-Dienst des Dashboards unterstützt vier Strategien zur Knotenauswahl:

STRATEGIE	VERHALTEN
first-fit `(deterministisch)`	Erster verfügbarer Knoten, der die Anforderungen des Modells erfüllt.
balanced `(Standard)`	Wählt den Knoten mit dem meisten absolut freien VRAM und verteilt die Last so natürlich für HA.
bin-pack	Engste Packung zur Konsolidierung von Workloads; einzelne Knoten bleiben frei und lassen sich leeren.
proportional	Geringste prozentuale Auslastung für eine faire Verteilung über heterogene Hardware.

5.2 Unterstützte Hardware

Der Daemon erkennt GPUs herstellerübergreifend automatisch:

HERSTELLER	ERKENNUNGSMETHODE
NVIDIA	`nvidia-smi` (GPU-Name, VRAM pro Karte)
AMD	Sysfs (`/sys/class/drm`), Fallback auf `rocm-smi`
Intel	`xpu-smi`-Discovery

Systeme mit Unified Memory (z. B. DGX Spark) werden erkannt, wenn GPUs null VRAM melden; 90 % des System-RAM gelten dann als nutzbare Kapazität. Ohne GPU wird reine CPU-Inferenz unterstützt.

5.3 Lebenszyklus der Inference-Engines

ENGINE	MODUS	DETAILS
Ollama	Pull-basiert	Modelle werden mit Fortschrittsanzeige aus der Ollama-Registry gezogen. Bis zu 2 parallele Pulls.
vLLM `(systemd)`	Service-verwaltet	Verwaltet die Template-Units `vllm-driver@{id}.service`. Berechnet die GPU-Speicherauslastung inkl. Overhead automatisch, begrenzt auf 90 %.
vLLM `(Docker)`	Containerisiert	Läuft in Containern in einem Netzwerk ohne Egress (`HF_HUB_OFFLINE=1`). Modelle lädt der Daemon vorab herunter. Blockiert allen ausgehenden Internetverkehr des Inferenzprozesses.

Fehlgeschlagene vLLM-Prozesse starten automatisch bis zu 3-mal neu (konfigurierbar). Fatale Fehler (GPU-OOM, CUDA-Fehler) führen ohne Wiederholung sofort zum Abbruch.

ABSCHNITT 06

Hochverfügbarkeit & 24/7-Betrieb

6.1 Betriebsphilosophie

Da Xinity vollständig auf kundeneigener Infrastruktur läuft, hängen Verfügbarkeitsgarantien von Infrastruktur und Betriebspraxis des Kunden ab. Xinity gibt daher keine harten SLA-Zusagen, baut die Plattform aber so, dass **Hochverfügbarkeit unkompliziert umsetzbar ist**.

6.2 Eingebaute Resilienz

MERKMAL	BESCHREIBUNG
Horizontale Skalierbarkeit	Jeder Dienst (Gateway, Daemon, Dashboard) kann als mehrere Replicas parallel laufen.
Automatisches Failover	Fällt ein Daemon-Knoten aus, leitet der Load Balancer des Gateways den Traffic sofort daran vorbei und nimmt ihn aus der Host-Auswahl.
Zuverlässige Knotenabmeldung	Bricht die Tether-Verbindung ab oder bleibt sie über das Liveness-Zeitfenster stumm, gilt der Knoten als nicht verfügbar. Verwaiste Registrierungen bleiben nicht zurück.
Redundante Inferenz	Wird dasselbe Modell auf mehreren Knoten bereitgestellt, läuft die Inferenz beim Ausfall eines Knotens weiter.
Prozess-Wiederherstellung	Vom Daemon verwaltete Inferenzprozesse starten bei Fehlern bis zu 3-mal automatisch neu.
Health-Monitoring	Die Compute-Seite des Dashboards zeigt alle GPU-Knoten in Echtzeit, automatisch aktualisiert alle 12 Sekunden.

6.3 Empfohlenes HA-Setup

Für einen 24/7-Produktivbetrieb:

01 **Mehrere Gateway-Replicas** hinter einem Load Balancer (z. B. Caddy oder nginx).

02 **Managed-PostgreSQL-Cluster** (Replikation, Failover, automatisierte Backups).

03 **Hochverfügbares Redis** (Redis Sentinel oder Cluster-Modus).

04 **Multi-Node-Inferenzpool** mit denselben Modellen auf mehreren GPU-Maschinen über `DEPLOYMENT_STRATEGY=balanced`.

05 **Redundante Infrastruktur** für DNS, TLS-Terminierung und Netzwerkanbindung.

Mit dieser Redundanz werden Ausfälle höchst unwahrscheinlich. Die Plattform macht es möglich; die Umsetzung liegt beim Kunden.

6.4 Zuverlässigkeit der Deployment-Synchronisation

Der Deployment-Sync-Dienst des Dashboards läuft kontinuierlich (alle 5 Minuten, mit sofortigen Triggern über PostgreSQL `NOTIFY`). Er hält Soll-Zustand und Ist-Zustand konsistent und korrigiert Abweichungen selbsttätig.

ABSCHNITT 07

Monitoring & Observability

7.1 Prometheus-Metriken

Alle drei Kerndienste stellen Prometheus-Metriken unter `/metrics` bereit:

Gateway-Metriken umfassen:

- Request-Raten und Latenzen pro Endpunkt
- Time-to-First-Token-Histogramme pro Deployment
- Token-Zahlen (Eingabe/Ausgabe) pro Modell, API-Key und Organisation
- Generierungsdurchsatz (Token pro Sekunde)
- Backend-Fehlerraten und Client-Abbrüche

Daemon-Metriken umfassen:

- GPU-Auslastung, Speicher, Temperatur, Leistungsaufnahme
- GPU-Throttle-Status und ECC-Fehler
- Kumulierter Energieverbrauch (Wh)
- Knotenzustand (`daemon_up`-Gauge)

Dashboard-Metriken umfassen:

- HTTP-Request-Raten pro Route
- Node.js-Laufzeitmetriken (CPU, Speicher, Event-Loop-Lag, Heap, GC)

Betreiber können den Umfang der Metriken erweitern, indem sie zusätzliche Prometheus-Counter, -Histogramme oder -Gauges aus der eigenen Anwendungslogik exportieren. Der Metrik-Endpunkt von Xinity ist vollständig mit eigener Instrumentierung kompatibel.

7.2 Auto-Discovery

Das Dashboard stellt einen HTTP-Service-Discovery-Endpunkt bereit (`GET /metrics/sd/daemons`), der pro registriertem Knoten eine Target-Gruppe zurückgibt. Prometheus fragt ihn alle 3 Minuten ab und nimmt Knoten automatisch auf oder entfernt sie, sobald sie online oder offline gehen.

7.3 Vorgefertigte Grafana-Dashboards

Vier vorgefertigte Grafana-Dashboards sind enthalten:

DASHBOARD	INHALT
Xinity Overview	Kennzahlen auf hoher Ebene: Request-Rate, Fehlerrate, aktive Anfragen, aktive Daemons, GPU-Anzahl, gesamte GPU-Leistung
Xinity Gateway	Traffic-Panels (Rate, Latenz, Status, Fehler), Modellzustand (TTFT, Backend-Fehler, Fehlerrate), Nutzung auf Organisationsebene
Xinity GPU / Compute	GPU-Auslastung, Speicher, Temperatur, Leistung vs. Limit, Throttling-Verläufe
Xinity Logs `(nur NixOS)`	Systemd-Journal-Logs aller Dienste über Loki/Promtail

Alle Dashboards sind mit `xinity-ai` getaggt und über ein gemeinsames Navigations-Dropdown miteinander verlinkt.

ABSCHNITT 08

Sicherheit & Compliance

8.1 Authentifizierung

Xinity unterstützt mehrere Authentifizierungsverfahren:

- **E-Mail/Passwort** mit optionaler Registrierung nur per Einladung
- **SSO über OIDC:** konfigurierbar auf Instanz- oder Organisationsebene (Domain-Verifizierung über DNS-TXT-Einträge)
- **Passkeys (WebAuthn/FIDO2):** Hardware-Sicherheitsschlüssel oder Plattform-Authentifikatoren (Touch ID, Windows Hello)
- **TOTP-2FA:** zeitbasierte Einmalpasswörter mit Backup-Codes

8.2 Rollenbasierte Zugriffskontrolle (RBAC)

Sechs granulare Rollen steuern den Zugriff auf sieben Ressourcentypen (API-Keys, API-Aufrufe, Responses, Deployments, Modelle, Anwendungen, Audit-Log):

ROLLE	ZUGRIFF
owner	Voller Zugriff, einschließlich Löschen der Organisation
admin	Voller Zugriff auf Ressourcen, kann die Organisation nicht löschen
member	Voller Zugriff außer auf das Audit-Log
labeler	Lesen von Aufrufen, voller Labeling-Zugriff, Lesen von Modellen/Anwendungen
viewer	Nur-Lese-Zugriff auf alle Ressourcen
pending	Kein Zugriff (Platzhalter bis zur Rollenzuweisung)

8.3 Datensouveränität

> KEIN ZWINGENDER DATENABFLUSS
>
> Das grundlegende Designprinzip von Xinity AI ist **kein zwingender Datenabfluss**. Inferenzanfragen, Antworten, Labels, Modellgewichte und GPU-Telemetrie bleiben standardmäßig in der Infrastruktur des Kunden. Für den Kernbetrieb sind keine Analytics, keine externe Telemetrie und keine externen API-Aufrufe erforderlich.

Einige wenige optionale Dienste *können* ins Internet hinausreichen – aber nur, wenn der Betreiber sie aktiviert:

- **Info Server:** Hosten Kunden ihren eigenen Info Server (selbst gehosteter Modellkatalog), liest dieser aus einem lokalen YAML-Katalog. Ohne ihn greift das Dashboard auf einen von Xinity gehosteten Endpunkt für Modell-Metadaten zurück (`sysinfo.xinity.ai`). Er dient ausschließlich Katalogdaten (Namen, Spezifikationen, Tags, Treiber-Strings), nie der Inferenz oder Kundendaten.

- **Websuche & Seitenabruf:** Optionale Integrationen (Google, Bing, Brave, Tavily, SearXNG usw.) ermöglichen Websuche und Seitenabruf in der Responses-API des Gateways. Aktiviert benötigen sie Internetzugang, um ihre Funktion zu erfüllen. Sie sind Opt-in-Funktionen und niemals Teil des standardmäßigen Inferenzpfads.

Betreiber in Umgebungen mit striktem Zero-Egress können alles lokal ausrollen und sämtliche optionalen ausgehenden Integrationen deaktivieren – für vollständige Datensouveränität.

8.4 TLS-Unterstützung

Das Gateway unterstützt TLS-Terminierung mit kundeneigenen Zertifikaten. TLS wird pro Knoten in der Datenbank erfasst, und das Gateway handelt automatisch das passende Protokoll aus. Für Inferenzanfragen signiert das Gateway Methode und Pfad mit dem knotenspezifischen Token, das der Daemon bei jedem Start neu erzeugt, und der Daemon prüft die Signatur. Das Token selbst wird nicht mitgeschickt.

8.5 Inferenzknoten ohne Datenbankzugang

Seit die Inferenzknoten über den Tether angebunden sind, liegen auf den GPU-Maschinen keine Datenbank-Zugangsdaten mehr. Ein kompromittierter Inferenzknoten gibt damit keinen Weg in die Control-Plane-Datenbank frei. Vom Dashboard verwaltete Geheimnisse reicht der Tether versiegelt durch: Er transportiert sie, ohne sie zu lesen. Das verkleinert die Angriffsfläche genau dort, wo in vielen Umgebungen die am wenigsten kontrollierte Hardware steht.

ABSCHNITT 09

Deployment-Optionen

Xinity empfiehlt drei Wege für den Produktivbetrieb. Andere Setups sind ebenfalls möglich:

WEG	AM BESTEN GEEIGNET FÜR
Xinity CLI	Jeder Linux-Server mit systemd. Installation mit einem Befehl: `xinity up all`
Docker Compose	Server mit Docker oder Cloud-VMs. Enthält optionale Profile für Caddy (HTTPS), SeaweedFS (S3-Storage), Infoserver (selbst gehostete Modell-Registry) und Prometheus/Grafana (Monitoring)
NixOS Flake	NixOS-Infrastruktur mit deklarativer, reproduzierbarer Konfiguration

9.1 Inferenzknoten hinzufügen

Die Control Plane (Gateway, Dashboard, Tether, Datenbank, Redis) wird einmal ausgerollt. Jeder weitere GPU-Knoten kommt unabhängig davon hinzu:

```
xinity up daemon   # auf jeder GPU-Maschine
```

Der Daemon verbindet sich mit dem Tether, registriert sich selbst und beginnt, Modell-Deployments entgegenzunehmen. Dieser Ablauf ist derselbe, unabhängig davon, wie die Control Plane ausgerollt wurde.

9.2 Secret-Management

Alle Deployment-Ziele unterstützen die `_FILE`-Konvention für Secrets: Jede Umgebungsvariable `KEY` kann über `KEY_FILE` auf einen Dateipfad verweisen. Der Dienst liest die Datei beim Start und verwendet ihren getrimmten Inhalt.

ABSCHNITT 10

Fazit

Xinity AI bietet eine vollständige On-Premises-Inferenzplattform für Unternehmen, die Daten nicht nach außen leiten können. Die Architektur – modulare Dienste, koordiniert über eine gemeinsame Datenbank – ermöglicht horizontale Skalierbarkeit, Fehlertoleranz und betriebliche Unabhängigkeit.

Durch die Kombination der Bequemlichkeit und Agilität von Ollama mit dem Durchsatz, der Geschwindigkeit und der Robustheit von vLLM gibt Xinity Betreibern die Freiheit, für jeden Workload die passende Engine zu wählen. Die Plattform ergänzt die Betriebsschicht, die produktive KI verlangt: Load Balancing, Observability, Zugriffskontrolle, Data Labeling und Multi-Node-Skalierung. Und das alles ohne Datenabfluss, mit vollständig auditierbarem Quellcode und Deployment-Flexibilität von Docker über NixOS bis hin zu Bare-Metal-systemd. Mit dem Tether als Verbindungsschicht skaliert die Plattform in Tausenden von Inferenzknoten, ohne dass ein einziger davon Zugriff auf die Datenbank benötigt.

Weiterführende Dokumentation

Architekturdokumentation

`github.com/xinity-ai/xinity-ai/blob/main/docs/architecture.md`

Gateway (Load Balancing, Canary, Metriken)

`github.com/xinity-ai/xinity-ai/blob/main/docs/features/gateway.md`

Daemon (GPU-Verwaltung, Modell-Lifecycle)

`github.com/xinity-ai/xinity-ai/blob/main/docs/features/daemon.md`

Deployment-Leitfaden

`github.com/xinity-ai/xinity-ai/blob/main/deployment/README.md`

Monitoring (Prometheus, Grafana)

`github.com/xinity-ai/xinity-ai/blob/main/docs/features/monitoring.md`

KI dort betreiben, wo Ihre Daten liegen. Ohne Ausnahme.

SOUVERÄN DURCH ARCHITEKTUR, NICHT DURCH VERTRAG.