

PRAXISLEITFADEN

KI-Workloads auf Kubernetes und Talos

Scheduling, Ressourcenlimits, Model Serving und Observability für
LLM- und agentische Workloads auf einer immutablen Linux-
Plattform.

KI-Workloads sind gewöhnliche Workloads - fast

Ein LLM-Endpoint, eine Vektordatenbank, eine Agenten-Laufzeit: alles Pods mit CPU, Speicher und Storage. Der Unterschied liegt in der Burstigkeit (ein Agent kann Minuten Denkzeit pro Anfrage verbrennen), der Zustandshaftung (Gedächtnisgraphen und Indizes sind wertvoll) und der Kostenkonzentration (ein Modelserver kann schwerer wiegen als alle anderen Dienste zusammen). Die Plattformantworten sind die bekannten - mit schärferem Augenmerk angewandt.

Scheduling und Ressourcenlimits

- Inference-Server erhalten explizite Requests UND Limits - ein Modell, das um 3 Uhr nachts Nachbarn verdrängt, ist ein Plattformfehler, kein Modellproblem
- Agenten-Laufzeiten von Model Serving trennen - Agenten skalieren mit Anfragerate, Modelle mit VRAM/RAM; vermischt werden beide unschedulbar
- Node-Pools oder Labels für Schwergewichte nutzen - speicherhungrige Pods auf geeignete Knoten pinnen vermeidet stille Pending-Zustände
- Realistische Probes setzen - Modell-Kaltstarts dauern Sekunden; eine Startup-Probe, die zu früh aufgibt, erzeugt Restart-Schleifen, die wie Abstürze aussehen

Talos hilft hier: Das immutable, API-verwaltete OS verhindert Konfigurationsdrift - Scheduling-Probleme sind damit Workload-Probleme, sichtbar und in Git behebbar.

Model-Serving-Muster, die sich bewähren

01 Embeddings und Retrieval zuerst

Beginnen Sie mit Embedding-Endpoints und einem Vector Store - sie liefern private Suche und RAG zu moderatem Preis und vermitteln die Betriebsmuster, bevor grössere Modelle kommen.

02 Kleine Modelle hinter stabiler API

Bedienen Sie Modelle über ein internes Gateway, damit Konsumenten nie den Modellnamen kennen. Ein Backend-Tausch wird so zu einem Deployment statt zu einem Refactoring.

03 Persistenten Kontext in eine Wissensschicht

UHL Cloud Halten Sie Agenten-Gedächtnis in einem dedizierten Dienst (wir betreiben Cogne) ^{uhl.cloud} statt in Prompts. Es ist prüfbar, gesichert und sicher über Agenten teilbar.

04 GPU erst bei gemessenem Bedarf

GPU-Knoten kommen, wenn Latenz oder Wartezeiten den Bedarf beweisen - nicht vorher. Pool labeln, bei Mischbetrieb mit Taints isolieren und als Plattform-Posten budgetieren.

Observability für agentische Arbeit

- ☐ Anfragen und Tool-Aufrufe loggen, nicht nur HTTP-Status - ein Agent in einer Schleife ist in Uptime-Metriken unsichtbar
- ☐ Tokens, Latenz und Queue-Tiefe pro Modell verfolgen - Kostenüberraschungen kommen über diese Kanäle
- ☐ Wachstum der Gedächtnisschicht selbst beobachten - Graphen und Indizes wachsen; Retention braucht einen Verantwortlichen
- ☐ Auf Drift zwischen Git und Cluster alarmieren - GitOps macht 'wer hat was geändert' zur beantworteten Frage
- ☐ KI-Dienste in dieselben Dashboards und Runbooks wie alles andere aufnehmen - separates Tooling verrottet

Dieses Dokument ist Teil der UHL-Ressourcen. Direkter Download und verwandte Arbeiten:
<https://uhl.cloud/de/resources/>

