

FIELD GUIDE

AI Workloads on Kubernetes and Talos

Scheduling, resource limits, model serving and observability for LLM and agentic workloads on an immutable Linux platform.

AI workloads are ordinary workloads - almost

An LLM endpoint, a vector database, an agent runtime: all are pods with CPU, memory and storage. What makes them different is burstiness (an agent can burn minutes of thinking per request), statefulness (memory graphs and indexes are precious) and cost concentration (one model server can outweigh every other service combined). The platform answers are the familiar ones - applied with stricter attention.

Scheduling and resource limits

- Give inference servers explicit requests AND limits - a model that evicts neighbours at 3am is a platform failure, not a model problem
- Separate agent runtimes from model serving - agents scale on request rate, models scale on VRAM/RAM; mixing them makes both unschedulable
- Use node pools or labels for heavyweight workloads - pinning memory-hungry pods to suitable nodes avoids silent pending states
- Set realistic probes - model cold-starts take seconds; a startup probe that gives up early causes restart loops that look like crashes

Talos helps here: the immutable, API-managed OS removes configuration drift, so scheduling problems are workload problems - visible and fixable in Git.

Model serving patterns that hold up

01 Embeddings and retrieval first

Start with embedding endpoints and a vector store - they deliver private search and RAG at modest cost, and teach you the operational patterns before larger models arrive.

02 Small models behind a stable API

Serve models through one internal gateway so consumers never learn a model's name. Swapping the backend becomes a deployment, not a refactor.

03 Persistent context in a knowledge layer

Keep agent memory in a dedicated service (we run Cognee) rather than in prompts. It is inspectable, backable-up, and shared safely across agents.

04 GPU only when measured

Add GPU nodes when latency or queue times prove the need - not before. Label the pool, isolate with taints if mixed, and budget for it as a platform line item.

Observability for agentic work

- ☐ Log requests and tool calls, not just HTTP status - an agent that loops is invisible in uptime metrics
 - ☐ Track tokens, latency and queue depth per model - cost surprises arrive through these channels
 - ☐ Watch memory growth on the memory layer itself - graphs and indexes grow; retention needs an owner
 - ☐ Alert on drift between Git and cluster - GitOps makes 'who changed what' an answered question
 - ☐ Include AI services in the same dashboards and runbooks as everything else - separate tooling rots
-

This document is part of the UHL resources collection. Direct download and related work:

<https://uhl.cloud/en/resources/>

