

评估当前组件部署状态，并给出 K8s 迁移路线图。

[4 tools called]

基于当前架构，以下是 K8s 迁移评估与路线图：

当前组件部署状态

已在 K8s(无需迁移)

- * MinIO (S3)
- * Redis (Cache)
- * Spark Jobs (通过 Operator 管理)
- * DataAlchemy Operator

仍在 Windows(需要迁移)

- * Agent B (LoRA 训练/推理) — 需要 GPU
- * Agent C (RAG/FAISS) — 需要 GPU(embedding)
- * Agent D (Finalist) — LLM API 调用(无 GPU 需求)
- * WebUI (FastAPI) — 无 GPU 需求
- * Coordinator — 编排器(无 GPU 需求)
- * Quant Stack (Polars) — CPU 密集型(无 GPU 需求)
- * SFT Generator — LLM API 调用(无 GPU 需求)

K8s 迁移路线图(分阶段)

阶段 1: 无 GPU 依赖组件(优先)

这些组件迁移风险低，可快速完成：

1. Quant Stack (Polars)

- * 仅需 CPU, 容器化简单
 - * 建议：创建 `quant-processor` Job/Deployment
2. SFT Generator (LLM API)

- * 仅调用外部 API, 无本地 GPU
- * 建议：创建 `sft-generator` Job

3. Agent D (Finalist)

- * 仅 LLM API 调用
 - * 建议：作为 `agent-d` Deployment
4. WebUI (FastAPI)

- * 纯 Web 服务

* 建议: 作为 `webui` Deployment + Service

5. Coordinator

* 编排逻辑, 可容器化

* 建议: 作为 `coordinator` Deployment

阶段 2: GPU 依赖组件 (需 K8s GPU 支持)

需要 K8s 集群支持 GPU(如 NVIDIA Device Plugin 或 AMD GPU Operator):

6. Agent C (RAG/FAISS)

* 需要 GPU(embedding 模型)

* 建议: 使用 AMD GPU 资源

7. Agent B (LoRA 训练/推理)

* 需要 GPU(训练与推理)

* 建议: 训练用 Job, 推理用 Deployment

技术挑战与解决方案

挑战 1: ROCm 在 K8s 中的支持

* 现状: 代码面向 Windows + ROCm

* 方案:

* 选项 A: 使用支持 AMD GPU 的 K8s 节点(如 AMD GPU Operator)

* 选项 B: 训练保留在 Windows, 推理迁移到 K8s(CPU/GPU 混合)

挑战 2: 依赖管理

* 现状: `pyproject.toml` 包含 Windows 特定依赖

* 方案: 创建多阶段 Dockerfile, 区分 Windows/Linux 依赖

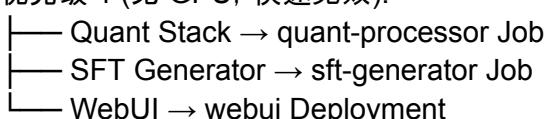
挑战 3: 数据持久化

* 现状: 使用 hostPath 映射 Windows 路径

* 方案: 迁移到 PVC(PersistentVolumeClaim)或继续使用 hostPath(开发环境)

建议的迁移顺序

优先级 1 (无 GPU, 快速见效):



优先级 2 (编排层):

 └─ Coordinator → coordinator Deployment

优先级 3 (GPU 组件, 需集群支持):

 └─ Agent C → agent-c Deployment (with GPU)
 └─ Agent B → agent-b Job/Deployment (with GPU)

* * *

实施建议

1. 先迁移无 GPU 组件, 验证基础流程
2. 确认 K8s 集群的 GPU 支持能力(NVIDIA/AMD)
3. 统一日志格式, 为后续 Loki 集成做准备
4. 使用 Operator 扩展, 将新组件纳入 `DataAlchemyStack` CRD 管理

需要我生成具体的 Deployment/Job YAML 模板, 或先实现某个组件的迁移吗 ?