Dagster

URL: https://main.taila597c2.ts.net:10000

Что это

Dagster — **оркестратор данных** для flavor-ml. Он представляет данные и артефакты моделей как граф «assets»: каждый asset — это какой-то кусок данных (датасет, обученная модель, файл метрик), произведённый Python-кодом и потребляемый другими assets. Dagster запускает код, отслеживает зависимости, хранит логи и lineage, позволяет перезапустить любой отдельный шаг.

Если знаком Airflow: Dagster занимает похожую нишу, но он asset-centric (объявляешь «я хочу чтобы этот датасет существовал», а не «запусти этот DAG»). Хорошо стыкуется с MLflow: Dagster знает *как* модель была получена; MLflow — *какая версия как себя показала*.

Что внутри в этом проекте

Три asset'а, определены в `pipelines/assets/recipes.py`:

Asset	Группа	Что делает
`raw_recipes`	ingest	Тянет сырой корпус рецептов в pandas DataFrame (источник — скрипт в `scripts/fetch_dataset.py`).
`normalized_recipes`	ingest	Берёт `raw_recipes`, прогоняет нормализацию имён ингредиентов (`flavor_ml/data/normalization.py`), возвращает чистый DataFrame.
`ingredient_embedding_model`	model	Берёт `normalized_recipes`, обучает Item2Vec, пишет `./artifacts/ingredient-embeddings.bin`, логирует run в MLflow (коммит `8e60e93`).

Зависимости: `raw_recipes` → `normalized_recipes` → `ingredient_embedding_model`. Повторная материализация модели пересчитывает только запрошенные шаги, а не весь граф.

Типичные задачи

Переобучить эмбеддинги (самое частое): 1. Открыть Dagster → вкладка **Assets** слева. 2. Найти `ingredient_embedding_model` (группа `model`). 3. Клик →

Materialize selected. 4. Смотришь страницу run'а: логи каждого шага в реальном времени.

Заново скачать и нормализовать корпус: 1. Выбрать `raw_recipes` и `normalized_recipes` в Assets. 2. Materialize. (Или из asset'а модели — «Materialize all upstream».)

Разобраться с упавшим run'ом: 1. Вкладка **Runs** → красная строка. 2. Кликнуть в красный шаг. 3. Tab Logs — полные stderr/stdout, включая Python traceback.

Посмотреть lineage (визуальный граф): 1. Assets → **Asset graph**. 2. Drag/zoom — assets показывают последнее время материализации и текущий статус.

Чего ещё нет

- **Нет schedules.** Все запуски ручные. Чтобы запускать по ночам — добавить `ScheduleDefinition` в `pipelines/definitions.py`.
- **Нет sensors.** Никто не следит за внешним состоянием (типа «появился новый файл в S3»), чтобы триггерить запуски автоматически.
- **Нет partitioning.** Asset'ы — единые блобы; если корпус вырастет, partitioning по дате/источнику будет следующим рефакторингом.

Что под капотом

- Контейнер: `flavor-ml-dagster-1` из `docker-compose.yml`
- Внутренний порт 3000; host-mapped: 3001 (чтобы не конфликтовало с Next.js dev на 3000)
- Публичный доступ: Tailscale Funnel на `:10000` → `localhost:3001`
- Code location: Python-пакет `pipelines/` примонтирован в контейнер
- State: SQLite в `/tmp` внутри контейнера по умолчанию — **история run'ов исчезает, если контейнер пересоздать** через `docker compose down`. Чтобы сохранять — монтировать volume на Dagster instance dir.

Подводные камни

- **Нет аутентификации.** Кто угодно с URL может смотреть assets, читать логи и **триггерить материализацию**. Посетитель, запустивший materialize, потребляет CPU/RAM/disk твоего PC. Для экспериментальной стадии это принятый компромисс; для всего серьёзнее — добавить auth до публикации URL.
- **Материализация не бесплатна.** `ingredient_embedding_model` действительно обучается; от десятков секунд до минут в зависимости от размера корпуса.

- **Логи эфемерны по умолчанию.** Пересоздание контейнера = потеря старой истории runs. MLflow при этом сохраняет training metrics независимо.
- **Долгие runs переживают refresh браузера**, но не рестарт контейнера. Не делай `make stack-down`, пока что-то материализуется.