OST
Ostschweizer
Fachhochschule

AnkiGenerator

Challenge Task FS2026

Diego Salazar

18.05.2026

Department Informatik

WO WISSEN WIRKT.

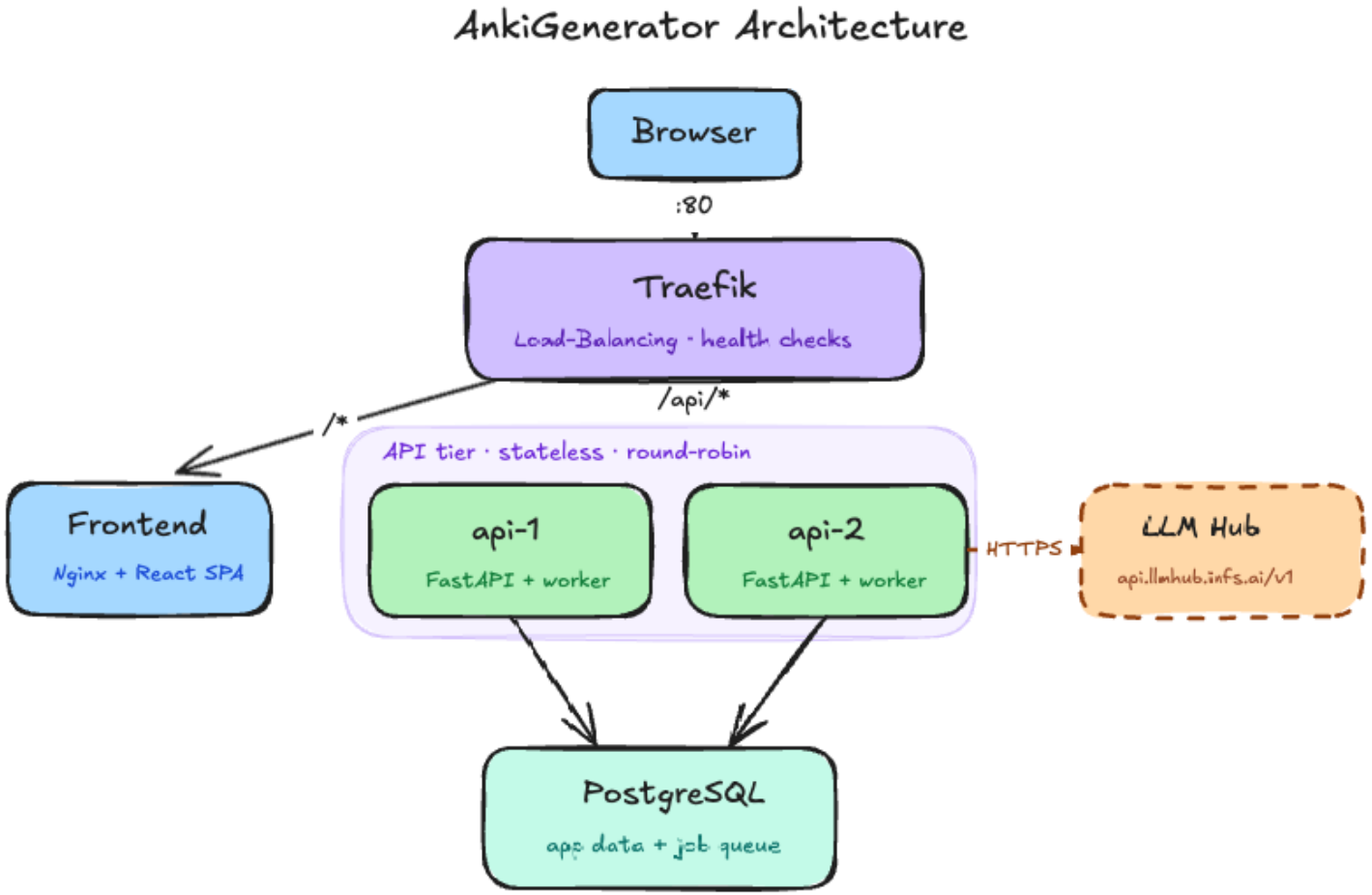
Architektur & Design

Teil 1 von 3

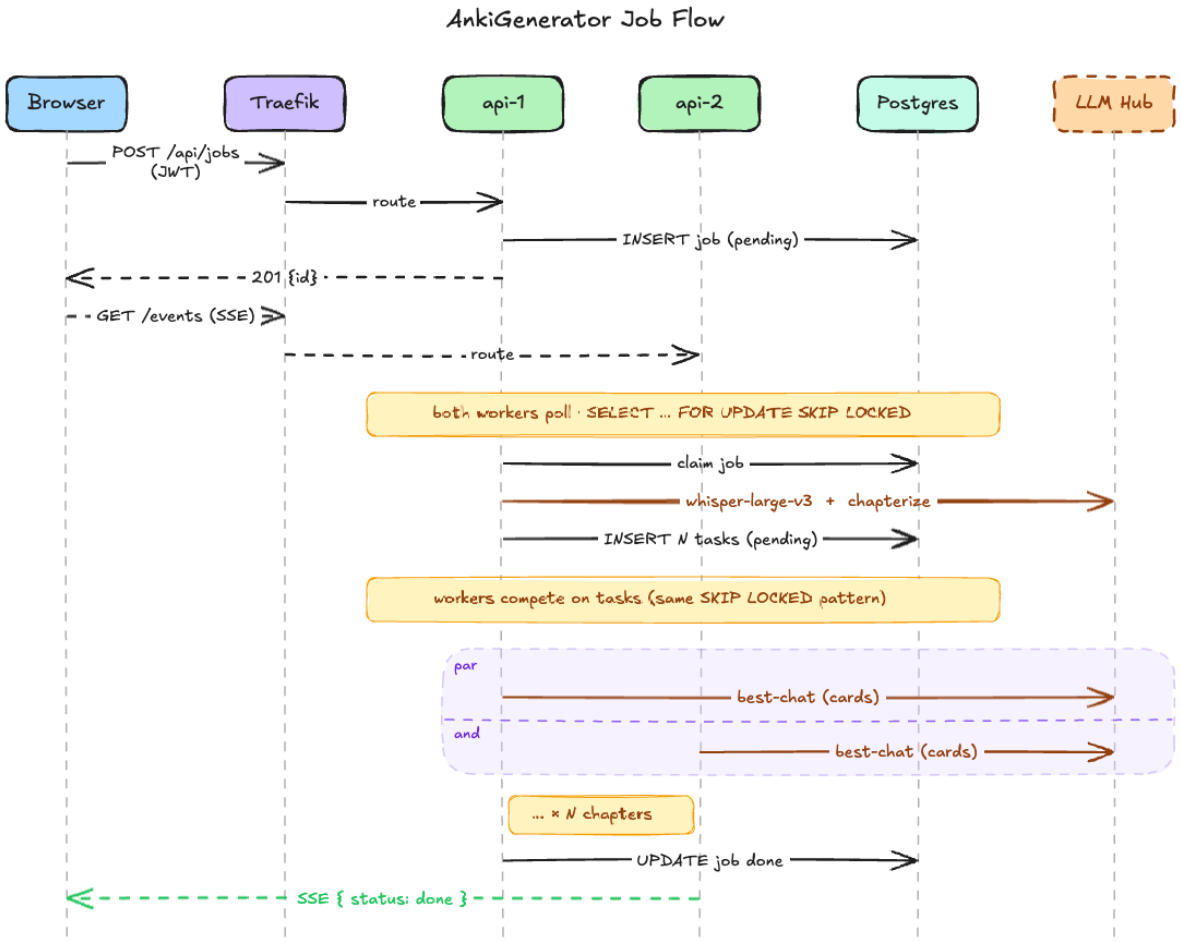
WO WISSEN WIRKT.

1 Überblick

- **5 Container**
  - Start mit docker compose up
- **Traefik**
  - Round Robin für API calls
  - Health checks (/api/health) im 5 Sekunden Takt
  - Failover
- **2x FastAPI + Worker**
  - Identisches Image & .env
  - Stateless
- **PostgreSQL**
  - Application Data & Job Queue
- **Frontend**
- **LLM Hub (Dependency)**



2 Flow



- Browser
  - Job mit Videofile wird gestartet (JWT im Header)
  - Öffnet SSE-Verbindung für Progress
- Traefik
  - Verteil via RR
- HTTP Request Handler
  - Validiert JWT
  - Insert Job Row (status='pending')
  - Antwortet sofort mit 201 {id}
- Worker (beide Instanzen pollen parallel)
  - Job Level
    - Claimed Job (via SELECT … FOR UPDATE SKIP LOCKED
    - Konvertiert mp4 → mp3 (ffmpeg), splittet falls > 25 MB
    - Whisper-Aufruf → Audio-Transkript
    - LLM-Aufruf → Chapterization (JSON)
    - Ein Task pro Kapitel in DB einfügen
  - Task Level
    - Claimed Task via SELECT … FOR UPDATE SKIP LOCKED
    - LLM-Aufruf für Flashcard Generierung
    - Flashcards in «flashcards» table einfügen
    - Letzter Task im Job → job.status = 'done' → SSE { status: done }

3 Authentication

- **JWT-Mechanismus**
  - HS256
  - Access-Token 15min
  - Refresh-Token 7d
- **Auth-Endpoints**
  - Register, Login, Refresh
  - Passwort reset & Email verify
  - Bcrypt mit auto-salt für Passwörter
- **OWASP**
  - Type-Claim-Enforcement zentral → verhindert Token-Confusion
  - User-Enumeration-Prevention auf /forgot-password → immer 204, kein Email-Probing
  - Minimale Payload (user_id, type, exp) → kein Leak von Email / Rolle bei Token-Abgriff
- **DS-Payoff**
  - Token ist self-contained → kein Session-State auf den Instanzen
  - API-Tier stateless → Round-Robin ohne Sticky Sessions
  - Horizontale Skalierung ohne Auth-Änderungen
- **Tradeoffs**
  - Tokens im localStorage (XSS-exposed)
  - httpOnly-Cookie-Migration dokumentiert, nicht implementiert

4 Design Choices

- **Postgres-as-Queue (statt Redis / RabbitMQ)**
  - Workers claimen via SELECT … FOR UPDATE SKIP LOCKED
  - Eine Komponente weniger zu deployen / monitoren
  - Row-Lock löst sich beim Disconnect → Crash-Recovery gratis
- **Worker-Thread im API-Container (statt Celery)**
  - Ein Image, ein docker compose up
  - Blocking LLM-Calls laufen in dediziertem Thread → Request-Latenz bleibt flach
  - Bricht: wenn Worker und API unabhängig skalieren müssten
- **Zwei-Level-Decomposition: jobs → tasks**
  - Ein Upload → N Task-Rows nach Chapterization
  - Beide Worker konkurrieren auf tasks → echter Fan-Out über Instanzen
  - Mehr Instanzen = mehr parallele Chapter
- **SSE statt WebSocket**
  - One-way reicht (Server → Client)
  - Funktioniert durch Traefik mit disabled Buffering
  - @microsoft/fetch-event-source erlaubt JWT im Header (statt native EventSource da Token im local storage)

5 Bottlenecks & Load Tests

- **Bottleneck-Hierarchie**
  - **LLM Hub:** >5 min Wall-Time pro 1-Std-Lecture · hub-bound · extern, nicht kontrollierbar
  - **DB Connection Pool:** ThreadedConnectionPool(2, 10) pro Instanz · 20 Connections total
  - **Whisper 25 MB Limit:** Workaround via 10-min mp3-Chunking
- **Was skaliert, was nicht**
  - Throughput (Jobs/h) ← skaliert *linear* mit Instanz-Anzahl
  - Per-Job-Latenz (>5 min) ← (llm-)hub-bound · unabhängig von Instanz-Anzahl
- **Load Test mit hey** (Endpoint: GET /api/jobs · JWT-protected · DB-Query)
  - 2 Instanzen, c=10: **1109 req/s · 100 % Success · P99 = 24.5 ms**
  - c=50+: Pool-Exhaustion sichtbar · ~1000 req/s Steady State unter Overload
  - /api/health ohne DB: **5291 req/s** → bestätigt: Pool ist der Bottleneck, nicht das Framework
- **Demo zeigt**
  - Live Load Test gegen Bottleneck #2 (DB-Pool)
  - Failover: docker compose stop api-2 → Traefik dropped Instanz nach Health-Check-Fail (~5 s)
  - Bottleneck #1 (LLM Hub) nicht in 5 min reproduzierbar, daher nur für demo: OpenAI API

Demo

Teil 2 von 3

WO WISSEN WIRKT.

Q&A

Teil 3 von 3

WO WISSEN WIRKT.