Meeting 15: Production Engineering & Deployment

AI-40X: Generative AI & Large Language Models

Hendri Karisma, M.T.
Dosen Teknik Informatika STMIK Tazkia
VP Engineering at jejakin.com, 2026

Semester 4 — 2025/2026

Outline

1. Dari Notebook ke Production
2. API Design dengan FastAPI
3. Containerization dengan Docker
4. Monitoring & Evaluation
5. MLOps Essentials
6. Kesimpulan & Referensi

Gap antara Prototype dan Production

Notebook / Prototype

- Berjalan di laptop sendiri
- Single user, single request
- Tidak ada error handling
- Data hardcoded / lokal
- "Kalau di saya jalan..."

Production System

- Berjalan di server 24/7
- Multi-user, concurrent requests
- Logging, monitoring, alerting
- Database, API keys, secrets management
- "Harus jalan di mana saja"

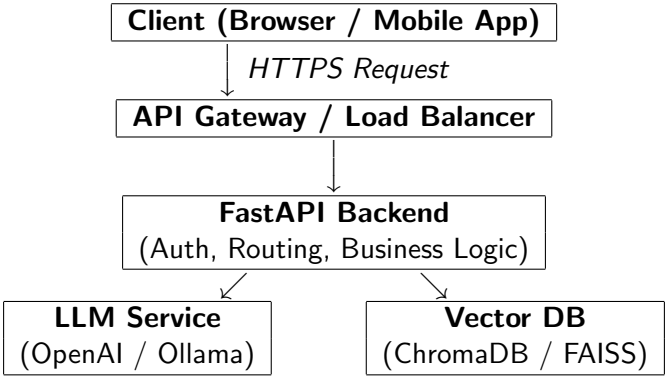
Fakta Industri

Menurut riset Google, hanya ~5% kode di sistem ML adalah kode model — sisanya adalah infrastruktur pendukung (data pipeline, serving, monitoring).

Production Readiness Checklist

1. **Reliability (Keandalan):**
   - Error handling & retry logic
   - Graceful degradation jika LLM down
   - Health check endpoints
2. **Scalability (Skalabilitas):**
   - Horizontal scaling (tambah instance)
   - Rate limiting & request queuing
   - Caching hasil yang sering ditanyakan
3. **Security (Keamanan):**
   - API key management (environment variables)
   - Input validation & prompt injection defense
   - HTTPS, authentication, authorization
4. **Monitoring (Pemantauan):**
   - Structured logging
   - Response time, error rate, token usage
   - Alerting jika ada anomali

Arsitektur Overview: Sistem LLM Production



Mengapa FastAPI?

- **Async Native:** Mendukung `async/await` — ideal untuk I/O-bound task seperti memanggil LLM API (tidak memblokir server saat menunggu respons).
- **Type Hints & Pydantic:** Validasi request/response otomatis menggunakan Python type hints. Jika user kirim data salah, langsung ditolak dengan error yang jelas.
- **Auto Documentation:** Swagger UI otomatis tersedia di `/docs` — tidak perlu menulis dokumentasi API manual.
- **Performa Tinggi:** Berbasis Starlette & Uvicorn — salah satu framework Python tercepat, setara Node.js/Go untuk I/O tasks.
- **Standar Industri:** Digunakan oleh Microsoft, Uber, Netflix untuk ML serving.

Basic Chat Endpoint

POST /chat — Endpoint Sederhana

```
from fastapi import FastAPI
from pydantic import BaseModel
from openai import AsyncOpenAI

app = FastAPI(title="Smart Knowledge Assistant API")
client = AsyncOpenAI()

class ChatRequest(BaseModel):
    message: str
    model: str = "gpt-4o-mini"
    temperature: float = 0.7

class ChatResponse(BaseModel):
    answer: str
    tokens_used: int

@app.post("/chat", response_model=ChatResponse)
async def chat(request: ChatRequest):
    response = await client.chat.completions.create(
        model=request.model,
        messages=[{"role": "user", "content": request.message}],
        temperature=request.temperature,
    )
    return ChatResponse(
        answer=response.choices[0].message.content,
        tokens_used=response.usage.total_tokens,
    )
```

Streaming Response dengan SSE

- **Problem:** LLM butuh 5–30 detik untuk generate jawaban penuh. User menunggu tanpa feedback.
- **Solusi: Server-Sent Events (SSE)** — kirim token satu per satu saat di-generate (seperti efek mengetik di ChatGPT).

Streaming Endpoint dengan SSE

```
from fastapi.responses import StreamingResponse
import json

@app.post("/chat/stream")
async def chat_stream(request: ChatRequest):
    async def generate():
        stream = await client.chat.completions.create(
            model=request.model,
            messages=[{"role": "user", "content": request.message}],
            stream=True,
        )
        async for chunk in stream:
            delta = chunk.choices[0].delta.content
            if delta:
                yield f"data: {json.dumps({'token': delta})}\n\n"
        yield "data: [DONE]\n\n"

    return StreamingResponse(generate(), media_type="text/event-stream")
```

Request/Response Schema dengan Pydantic

Schema Design yang Baik

```
from pydantic import BaseModel, Field
from typing import Optional
from enum import Enum

class DomainType(str, Enum):
    pendidikan = "pendidikan"
    kesehatan = "kesehatan"
    umkm = "umkm"

class RAGRequest(BaseModel):
    question: str = Field(..., min_length=3, max_length=1000,
                          description="Pertanyaan user")
    domain: DomainType = Field(..., description="Domain knowledge base")
    top_k: int = Field(default=5, ge=1, le=20,
                       description="Jumlah dokumen yang diambil")

class Source(BaseModel):
    document: str
    page: Optional[int] = None
    relevance_score: float

class RAGResponse(BaseModel):
    answer: str
    sources: list[Source]
    tokens_used: int
    latency_ms: float
```

Apa itu Docker dan Mengapa Penting?

- **Masalah Klasik:** "Di laptop saya jalan, tapi di server tidak jalan."
  - Versi Python berbeda
  - Library yang hilang
  - Konfigurasi OS berbeda
- **Docker = Kontainer:** Membungkus aplikasi + semua dependensinya ke dalam satu paket yang portabel.
- **Analogi:** Docker itu seperti kontainer kapal — isinya apapun, bisa dikirim ke mana saja selama ada pelabuhan (Docker runtime).

Konsep Kunci

Image:	Blueprint / cetak biru aplikasi (read-only).
Container:	Instance yang berjalan dari sebuah image.
Dockerfile:	Resep untuk membuat image.
Docker Compose:	Mengatur beberapa container sekaligus.

Dockerfile untuk Aplikasi ML

Dockerfile — Multi-stage Build

```
# Stage 1: Builder (install dependencies)
FROM python:3.11-slim AS builder
WORKDIR /app
COPY requirements.txt .
RUN pip install --no-cache-dir --user -r requirements.txt

# Stage 2: Runtime (image final yang ringan)
FROM python:3.11-slim
WORKDIR /app

# Copy dependencies dari builder stage
COPY --from=builder /root/.local /root/.local
ENV PATH=/root/.local/bin:$PATH

# Copy source code
COPY ./app ./app
COPY ./data ./data

# Expose port dan jalankan
EXPOSE 8000
CMD ["uvicorn", "app.main:app", "--host", "0.0.0.0", "--port", "8000"]
```

- **Multi-stage build:** Image builder terpisah dari runtime — image final lebih kecil (dari ~1.2GB menjadi ~400MB).

Docker Compose: Multi-Service Setup

docker-compose.yml — API + ChromaDB

```
version: "3.9"
services:
  api:
    build: .
    ports:
      - "8000:8000"
    environment:
      - OPENAI_API_KEY=${OPENAI_API_KEY}
      - CHROMA_HOST=chromadb
      - CHROMA_PORT=8001
    depends_on:
      - chromadb
    restart: unless-stopped

  chromadb:
    image: chromadb/chroma:latest
    ports:
      - "8001:8000"
    volumes:
      - chroma_data:/chroma/chroma
    restart: unless-stopped

volumes:
  chroma_data:
```

Jalankan: `docker compose up -d` — kedua service langsung berjalan dan terhubung.

Docker Best Practices

1. **Gunakan `.dockerignore`:**
   - Jangan masukkan `venv/`, `.git/`, `__pycache__/`, `.env` ke dalam image.
   - Mempercepat build dan mengurangi ukuran image.
2. **Multi-stage Builds:**
   - Pisahkan stage instalasi (besar) dari stage runtime (kecil).
3. **Pin Versi Dependency:**
   - `transformers==4.41.0`, bukan `transformers` (tanpa versi).
   - Mencegah build yang tiba-tiba rusak karena update library.
4. **Environment Variables untuk Secrets:**
   - API keys JANGAN di-hardcode — gunakan `.env` file atau secret manager.
5. **Health Check:**
   - Tambahkan `HEALTHCHECK` di Dockerfile agar orchestrator tahu apakah container sehat.

RAGAS: Evaluasi Kualitas RAG

RAGAS (Retrieval Augmented Generation Assessment) — framework untuk mengukur kualitas sistem RAG menggunakan LLM sebagai juri.

Retrieval Metrics

1. **Context Precision:**
   - Apakah dokumen yang diambil *relevan*?
   - Semakin sedikit "sampah" di context, semakin baik.
2. **Context Recall:**
   - Apakah kita berhasil menemukan *semua* dokumen relevan?
   - Tidak ada informasi penting yang terlewat.

Generation Metrics

1. **Answer Relevancy:**
   - Apakah jawaban *relevan* dengan pertanyaan?
   - Bukan sekadar benar, tapi menjawab apa yang ditanya.
2. **Faithfulness:**
   - Apakah jawaban *grounded* (berdasarkan) konteks?
   - Tidak ada halusinasi — setiap klaim bisa ditelusuri ke dokumen sumber.

RAGAS: Contoh Evaluasi

Evaluasi RAG dengan RAGAS

```
from ragas import evaluate
from ragas.metrics import (
    context_precision, context_recall,
    answer_relevancy, faithfulness
)
from datasets import Dataset

# Siapkan data evaluasi
eval_data = Dataset.from_dict({
    "question": ["Apa syarat beasiswa Tazkia?"],
    "answer": ["Syarat beasiswa: IPK min 3.5, aktif organisasi."],
    "contexts": [["Beasiswa diberikan kepada mahasiswa dengan IPK
minimal 3.5 dan aktif dalam kegiatan kampus."]],
    "ground_truth": ["IPK minimal 3.5 dan aktif organisasi."],
})

result = evaluate(
    dataset=eval_data,
    metrics=[context_precision, context_recall,
             answer_relevancy, faithfulness],
)
print(result)
# {'context_precision': 0.95, 'faithfulness': 0.90, ...}
```

Logging & Monitoring di Production

Structured Logging

- Gunakan format JSON untuk log — mudah di-parse oleh tools monitoring.
- Setiap request harus punya `request_id` unik untuk tracing.
- Log minimal: timestamp, level, message, request_id, latency_ms.
- Tools: `structlog`, `loguru` (Python).

Metrik yang Dimonitor

- **Response Time:** P50, P95, P99 latency — berapa lama rata-rata respons?
- **Error Rate:** Persentase request yang gagal (target: $< 1\%$).
- **Token Usage:** Total token per hari — untuk kontrol biaya.
- **Throughput:** Requests per second (RPS).
- Tools: Prometheus + Grafana.

Model Versioning & A/B Testing

Model Versioning

- Beri nama/tag yang jelas untuk setiap versi model atau konfigurasi:
  - `rag-v1.0-gpt4o-mini`
  - `rag-v1.1-chunk512-rerank`
  - `rag-v2.0-finetuned-llama`
- Simpan konfigurasi: model name, chunk size, embedding model, temperature, system prompt.
- Tools: Git tags, MLflow, Weights & Biases.

A/B Testing

- Bandingkan dua versi model secara bersamaan di production:
  - 80% traffic $\rightarrow$ Model A (stabil)
  - 20% traffic $\rightarrow$ Model B (eksperimen)
- Ukur metrik: akurasi jawaban, latency, user satisfaction.
- Jika Model B lebih baik $\rightarrow$ rollout 100%.
- Jika lebih buruk $\rightarrow$ rollback tanpa downtime.

CI/CD untuk ML Pipeline

CI/CD = Continuous Integration / Continuous Deployment — otomatisasi proses build, test, dan deploy.

1. **Continuous Integration (CI):**
   - Setiap kali kode di-push ke Git → otomatis jalankan:
   - Unit tests (apakah fungsi bekerja?)
   - Integration tests (apakah API merespons dengan benar?)
   - Evaluation tests (apakah RAGAS score di atas threshold?)
   - Tools: GitHub Actions, GitLab CI.
2. **Continuous Deployment (CD):**
   - Jika semua test lulus → otomatis build Docker image baru.
   - Deploy ke staging environment untuk review.
   - Setelah approval → deploy ke production.

Pipeline Sederhana

`git push` → `Run Tests` → `Build Docker Image` → `Deploy to Staging` → `Deploy to Prod`

Cost Management: Mengelola Biaya LLM

- **Masalah:** LLM API itu mahal — GPT-4o bisa $2.50–$10 per 1M token. Tanpa kontrol, biaya bisa meledak.

1. **Token Counting:**
   - Hitung token sebelum dan sesudah request.
   - Set `max_tokens` di setiap request untuk membatasi output.
   - Monitor total token harian/bulanan.
2. **Rate Limiting:**
   - Batasi jumlah request per user per menit (misal: 10 req/menit).
   - Mencegah abuse dan mengontrol biaya.
3. **Strategi Hemat:**
   - **Caching:** Simpan jawaban untuk pertanyaan yang sering diulang.
   - **Model Routing:** Pertanyaan sederhana → model murah (GPT-4o-mini). Pertanyaan kompleks → model besar (GPT-4o).
   - **Prompt Optimization:** Kurangi panjang system prompt yang tidak perlu.

Kesimpulan

1. **Notebook $\neq$ Production:** Ada gap besar antara prototype dan sistem yang siap digunakan user — reliability, scalability, security, monitoring.
2. **FastAPI** adalah standar industri untuk serving ML model — async, type-safe, auto docs, mendukung streaming.
3. **Docker** memecahkan masalah "works on my machine" — kontainerisasi membuat deployment konsisten di mana saja.
4. **Evaluasi RAG** dengan RAGAS memberikan metrik objektif: Context Precision, Context Recall, Answer Relevancy, Faithfulness.
5. **MLOps** menjaga kesehatan sistem jangka panjang: versioning, A/B testing, CI/CD, dan cost management.

Pesan Kunci

Membangun model AI itu 10% dari pekerjaan. 90% sisanya adalah engineering — dan itulah yang membedakan demo dari produk nyata.

Referensi

- Tiangolo, S. (2019). FastAPI Documentation. `https://fastapi.tiangolo.com/`
- Docker Documentation. `https://docs.docker.com/`
- Es, S., et al. (2023). "RAGAS: Automated Evaluation of Retrieval Augmented Generation." *arXiv:2309.15217*.
- Sculley, D., et al. (2015). "Hidden Technical Debt in Machine Learning Systems." *NeurIPS*.
- Huyen, C. (2022). *Designing Machine Learning Systems*. O'Reilly Media.
- Kreuzberger, D., et al. (2023). "Machine Learning Operations (MLOps): Overview, Definition, and Architecture." *IEEE Access*.