Gmail **Fillipe Guerra <fillipe.backup@gmail.com>**

IA_Autonoma_Parte15
1 mensagem

Fillipe Guerra <fillipe.backup@gmail.com> 28 de outubro de 2025 às 11:35
Para: Fillipe Augusto Gomes Guerra <fillipe182@hotmail.com>, Fillipe Guerra <fillipe.backup@gmail.com>

Fechando o pacote “nível produção” com **(1) Dockerfile do app Node/Express (build prod)**, **(2) NGINX reverse-proxy com HTTP/2 + SSE/WebSocket + TLS pronto pra Let’s Encrypt**, e **(3) Observabilidade completa (Prometheus + Grafana + métricas no Node e no FastAPI)**.
Tudo abaixo já vem **para colar**. Se algum arquivo existir, **substitui** por estes (são supersets compatíveis).

---

1) App Node/Express – Dockerfile (produção)

> Multi-stage com PNPM, build “slim”, variáveis de ambiente e saúde.

`Dockerfile` (na raiz do projeto)

```
# --- Stage 1: builder ---
FROM node:20-alpine AS builder
WORKDIR /usr/src/app
RUN corepack enable && corepack prepare pnpm@9.12.0 --activate
COPY package.json pnpm-lock.yaml ./
RUN pnpm fetch
COPY . .
RUN pnpm install -r --offline
RUN pnpm -w build

# --- Stage 2: runner ---
FROM node:20-alpine AS runner
WORKDIR /usr/src/app
ENV NODE_ENV=production
RUN corepack enable && corepack prepare pnpm@9.12.0 --activate

# Copia só o necessário
COPY --from=builder /usr/src/app/package.json /usr/src/app/pnpm-lock.yaml ./
COPY --from=builder /usr/src/app/node_modules ./node_modules
COPY --from=builder /usr/src/app/dist ./dist

# Usuário não-root
RUN addgroup -S app && adduser -S app -G app
USER app
EXPOSE 3000
HEALTHCHECK --interval=30s --timeout=3s --start-period=30s --retries=3 \
  CMD node -e "fetch('http://localhost:3000/health').then(r=>{ if(r.ok) process.exit(0); else
process.exit(1)}).catch(()=>process.exit(1))"

CMD ["node","dist/server/index.js"]
```

> Se seu app inicia via `pnpm start:prod`, troque o `CMD` conforme seu entrypoint.
> O `dist/server/index.js` é o exemplo típico após `pnpm build`.

---

2) NGINX reverse-proxy (HTTP/2 + SSE/WebSocket + TLS)

2.1 docker-compose com NGINX

No seu `docker-compose.yml` (que já temos), **adicione** o serviço `nginx`:

```
services:
  nginx:
    image: nginx:1.27-alpine
    depends_on: [app, inference]
    ports:
      - "80:80"
      - "443:443"
    volumes:
      - ./ops/nginx/nginx.conf:/etc/nginx/nginx.conf:ro
      - ./ops/nginx/conf.d:/etc/nginx/conf.d:ro
      - ./ops/nginx/certs:/etc/letsencrypt
    environment:
      - NGINX_ENVSUBST_OUTPUT_DIR=/etc/nginx
    restart: unless-stopped
```

2.2 Estrutura NGINX

```
ops/
 └── nginx/
     ├── nginx.conf
     └── conf.d/
         ├── aion.conf
         └── gzip.conf
```

`ops/nginx/nginx.conf`

```
user  nginx;
worker_processes auto;

events {
  worker_connections  4096;
  multi_accept        on;
}

http {
  include       /etc/nginx/mime.types;
  default_type  application/octet-stream;
  sendfile      on;
  tcp_nopush    on;
  tcp_nodelay   on;

  # Timeouts generosos p/ SSE/WebSocket
  keepalive_timeout  65;
  proxy_read_timeout 300s;
  proxy_send_timeout 300s;

  # Logs
  access_log  /var/log/nginx/access.log;
  error_log   /var/log/nginx/error.log warn;

  # Compressão
  include /etc/nginx/conf.d/gzip.conf;

  # Limites básicos
  client_max_body_size 64m;

  # Map para headers realip
  map $http_upgrade $connection_upgrade {
    default upgrade;
    ''      close;
  }

  include /etc/nginx/conf.d/*.conf;
}
```

`ops/nginx/conf.d/gzip.conf`

```
gzip on;
gzip_comp_level 5;
gzip_min_length 256;
gzip_proxied any;
gzip_types
  text/plain text/css application/json application/javascript
  text/xml application/xml application/xml+rss text/javascript
  font/ttf font/otf image/svg+xml;
```

ops/nginx/conf.d/aion.conf

> Ajuste `server_name` para seu domínio (ex.: aion.suaempresa.com). O arquivo cobre **HTTP→HTTPS**, **SSE** e **WebSocket**.

```
# Redireciona HTTP -> HTTPS (Let's Encrypt pode validar por /.well-known/acme-challenge)
server {
  listen 80;
  listen [::]:80;
  server_name aion.local aion.seudominio.com.br;
  # Descomente se for emitir cert com Certbot:
  # location /.well-known/acme-challenge/ { root /var/www/certbot; }
  location / { return 301 https://$host$request_uri; }
}

server {
  listen 443 ssl http2;
  listen [::]:443 ssl http2;
  server_name aion.local aion.seudominio.com.br;

  # TLS - substitua pelos caminhos reais do Let's Encrypt
  ssl_certificate     /etc/letsencrypt/live/aion.seudominio.com.br/fullchain.pem;
  ssl_certificate_key /etc/letsencrypt/live/aion.seudominio.com.br/privkey.pem;
  ssl_protocols TLSv1.2 TLSv1.3;
  ssl_ciphers HIGH:!aNULL:!MD5;

  # Proxy para App (Node)
  location / {
    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header X-Forwarded-Proto https;
    proxy_read_timeout 300s;

    # WebSocket/SSE
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection $connection_upgrade;

    proxy_pass http://app:3000;
  }

  # Proxy explícito para SSE do chat (opcional; já cobre com /)
  location /api/ai/chat/stream {
    proxy_http_version 1.1;
    proxy_set_header Connection "";
    chunked_transfer_encoding off;
    proxy_buffering off;           # importante p/ SSE
    proxy_cache off;
    proxy_read_timeout 300s;
    proxy_pass http://app:3000;
  }

  # Proxy para microserviço de inferência (se expuser rota pública, opcional)
  location /llm/ {
    rewrite ^/llm/?(.*)$ /$1 break;
    proxy_set_header Host $host;
    proxy_set_header X-Forwarded-Proto https;
    proxy_pass http://inference:8008/;
  }
}
```

TLS/Certificado: use `certbot` no host e monte `/etc/letsencrypt` dentro do container, ou rode um sidecar `certbot` no compose. Para Replit/ambiente sem domínio, você pode manter apenas porta 80 (sem TLS).

---

3) Observabilidade Completa (Prometheus + Grafana)

Vamos:

- Expor **/metrics** no **Node** (prom-client).
- Expor **/metrics** no **FastAPI** (prometheus_fastapi_instrumentator).
- Subir **Prometheus & Grafana** via compose.
- Fornecer **prometheus.yml** e um **dashboard básico**.

3.1 Métricas no Node (Express)

Instale

```
pnpm add prom-client
```

`server/metrics/prom.ts`

```
import client from "prom-client";
import type { Express, Request, Response } from "express";

const Registry = client.Registry;
const register = new Registry();
client.collectDefaultMetrics({ register });

export const counters = {
  http_requests_total: new client.Counter({
    name: "aion_http_requests_total",
    help: "Total de requisições HTTP",
    labelNames: ["method","route","status"]
  }),
  answers_total: new client.Counter({
    name: "aion_answers_total",
    help: "Respostas geradas",
    labelNames: ["source"] // local | fallback
  }),
  fallback_total: new client.Counter({
    name: "aion_fallback_total",
    help: "Qtde de fallbacks",
    labelNames: ["reason"]
  })
};
register.registerMetric(counters.http_requests_total);
register.registerMetric(counters.answers_total);
register.registerMetric(counters.fallback_total);

export function metricsMiddleware(app: Express){
  // contador por request
  app.use((req: Request, res: Response, next)=>{
    const start = Date.now();
    res.on("finish", ()=>{
      counters.http_requests_total.labels(req.method, req.route?.path || req.path,
String(res.statusCode)).inc();
    });
    next();
  });

  // endpoint /metrics
  app.get("/metrics", async (_req, res)=>{
    res.set("Content-Type", register.contentType);
    res.end(await register.metrics());
  });
}

export { register };
```

No bootstrap do servidor (onde cria o app):

```
import { metricsMiddleware } from "./metrics/prom";
metricsMiddleware(app);
```

Dentro do seu `answer.router.ts`, incremente métricas:

```
import { counters } from "../metrics/prom";
// ...
if (used === "local") counters.answers_total.labels("local").inc();
else counters.answers_total.labels("fallback").inc();

if (used === "fallback") counters.fallback_total.labels("local_failed_or_low_conf").inc();
```

3.2 Métricas no FastAPI (inferência)

Adicionar dependência:

```
pip3 install prometheus-fastapi-instrumentator
```

(se estiver com Docker, adicione em `requirements.txt` do microserviço)

Em `trainer/inference/app.py` (adicionar no topo):

```
from prometheus_fastapi_instrumentator import Instrumentator
```

No `@app.on_event("startup")`, após `load_model()`:

```
Instrumentator().instrument(app).expose(app, endpoint="/metrics", include_in_schema=False)
```

Pronto: microserviço expõe `/metrics`.

3.3 Prometheus + Grafana no compose

Adicione no `docker-compose.yml`:

```
services:
  prometheus:
    image: prom/prometheus:v2.55.0
    volumes:
      - ./ops/prometheus/prometheus.yml:/etc/prometheus/prometheus.yml:ro
      - promdata:/prometheus
    ports: ["9090:9090"]
    restart: unless-stopped

  grafana:
    image: grafana/grafana:11.1.0
    ports: ["3001:3000"]
    environment:
      - GF_SECURITY_ADMIN_USER=admin
      - GF_SECURITY_ADMIN_PASSWORD=admin
    volumes:
      - grafana-data:/var/lib/grafana
      - ./ops/grafana/provisioning:/etc/grafana/provisioning:ro
    depends_on: [prometheus]
    restart: unless-stopped

  node_exporter:
    image: prom/node-exporter:v1.8.1
    pid: host
    network_mode: host
    restart: unless-stopped

  cadvisor:
    image: gcr.io/cadvisor/cadvisor:v0.49.1
    ports: ["8080:8080"]
    volumes:
      - /:/rootfs:ro
      - /var/run:/var/run:rw
      - /sys:/sys:ro
      - /var/lib/docker/:/var/lib/docker:ro
    restart: unless-stopped

volumes:
  promdata:
  grafana-data:
```

Estrutura Prom/Grafana

```
ops/
├── prometheus/
│   └── prometheus.yml
└── grafana/
    └── provisioning/
        ├── datasources/
        │   └── prometheus.yaml
        └── dashboards/
            ├── dashboards.yaml
            └── aion-overview.json
```

`ops/prometheus/prometheus.yml`

```
global:
  scrape_interval: 10s
  evaluation_interval: 10s

scrape_configs:
  - job_name: "aion-app"
    static_configs:
      - targets: ["app:3000"]

  - job_name: "aion-inference"
    static_configs:
      - targets: ["inference:8008"]

  - job_name: "node-exporter"
    static_configs:
      - targets: ["node_exporter:9100"]

  - job_name: "cadvisor"
    static_configs:
      - targets: ["cadvisor:8080"]
```

`ops/grafana/provisioning/datasources/prometheus.yaml`

```
apiVersion: 1
datasources:
  - name: Prometheus
    type: prometheus
    access: proxy
    url: http://prometheus:9090
    isDefault: true
```

`ops/grafana/provisioning/dashboards/dashboards.yaml`

```
apiVersion: 1
providers:
  - name: "AION Dashboards"
    folder: "AION"
    type: file
    options:
      path: /etc/grafana/provisioning/dashboards
```

`ops/grafana/provisioning/dashboards/aion-overview.json`
(dashboard simples com gráficos de requisições, answers, fallback e latência; pode importar depois algo mais elaborado)

```
{
  "title": "AION Overview",
  "uid": "aion-overview",
  "timezone": "browser",
  "schemaVersion": 36,
  "version": 1,
  "editable": true,
  "panels": [
    {
      "type": "graph",
      "title": "HTTP Requests",
      "targets": [
        { "expr": "sum(rate(aion_http_requests_total[1m]))", "legendFormat": "req/s" }
      ],
      "gridPos": { "x": 0, "y": 0, "w": 8, "h": 8 }
    },
    {
      "type": "graph",
      "title": "Answers (local vs fallback)",
      "targets": [
        { "expr": "sum(rate(aion_answers_total{source=\"local\"}[1m]))", "legendFormat": "local" },
        { "expr": "sum(rate(aion_answers_total{source=\"fallback\"}[1m]))", "legendFormat": "fallback"
}
      ],
      "gridPos": { "x": 8, "y": 0, "w": 8, "h": 8 }
    },
    {
      "type": "graph",
      "title": "Fallbacks by reason",
      "targets": [
        { "expr": "sum(rate(aion_fallback_total[1m])) by (reason)", "legendFormat": "{{reason}}" }
      ],
      "gridPos": { "x": 16, "y": 0, "w": 8, "h": 8 }
    },
    {
      "type": "graph",
      "title": "App CPU (node_exporter)",
      "targets": [
        { "expr": "100 - (avg by(instance) (irate(node_cpu_seconds_total{mode=\"idle\"}[5m])) * 100)",
"legendFormat": "CPU %" }
      ],
      "gridPos": { "x": 0, "y": 8, "w": 12, "h": 8 }
    },
    {
      "type": "graph",
      "title": "Containers CPU (cAdvisor)",
      "targets": [
        { "expr": "sum(rate(container_cpu_usage_seconds_total{image!=\"\"}[1m])) by (name)",
"legendFormat": "{{name}}" }
      ],
      "gridPos": { "x": 12, "y": 8, "w": 12, "h": 8 }
    }
  ]
}
```

---

4) Export JSONL + Treino: rotas e cron (recap)

Você já tem:

- `POST /api/ai/metrics/export.jsonl` (gera `trainer/events.jsonl`)
- `POST /api/ai/train/lora/start` (dispara QLoRA)
- `GET /api/ai/train/lora/status` / `cancel`

Se quiser **automatizar tudo**, adicione um **cron** diário que:

1. Exporta JSONL

2. (Opcional) checa metas (fallback↑/nDCG↓) e dispara treino

Ex.: já entreguei `watcher.ts` (metas) + `export.cron.ts`.
No **bootstrap** do servidor, garanta:

```
startExportCron();
startTrainWatcher();
```

---

5) Como subir tudo (passo-a-passo rápido)

1. **Copie** todos os arquivos acima.
2. Ajuste `.env` do app e microserviço (modelos, portas, thresholds).
3. **Build & Up**:

```
docker compose up -d --build
# → app:3000, inference:8008, nginx:80/443, prometheus:9090, grafana:3001
```

4. Acesse **Grafana** em http://localhost:3001 (ou via NGINX, se expor)
   - user/pass: `admin/admin` (troque depois)
   - Dashboard **AION/AION Overview** já aparece provisionado.
5. Teste **SSE**:
   - Front: `EventSource('/api/ai/chat/stream?q=...')`
   - NGINX faz pass-through sem buffer.

---

6) Resultado

- **Deploy padrão** (CPU/GPU), com **reverse-proxy robusto**, **SSE/WebSocket**, **TLS**, **observabilidade completa**, e **painel pronto**.
- A AION agora roda **autônoma**, treina **LoRA** com seus próprios logs, **usa cada vez menos OpenAI**, e você tem **controle total** (budget, metas, governança, telemetria).