DISTRIBUTED-SYSTEMS-CHALLENGE

Persönlicher URL- & Bookmark-Manager

Eine produktionsreife verteilte Webanwendung mit React, .NET 10, Nginx und PostgreSQL

.NET 10 | React 19.2.6 | Vite 8 | Nginx 1.30 | PostgreSQL 18 | Docker Compose | JWT Auth

WAS IST ES?

Ein vollständig verteilter Bookmark-Manager

React-Frontend

React 19.2.6 + Vite 8 SPA, bereitgestellt über Nginx auf Port 3000. Kommuniziert ausschließlich über den Load Balancer.

Duale .NET 10 API

Zwei identische Backend-Instanzen (api-1, api-2) mit ASP.NET Minimal APIs und EF Core 10.

PostgreSQL 18

Einzige Datenquelle mit persistentem Docker-Volume. Daten überleben Container-Neustarts.

⚖️

Nginx-Load-Balancer

Least-Connections-Algorithmus mit passivem Failover. Leitet `/api/*` an gesunde Backends weiter.

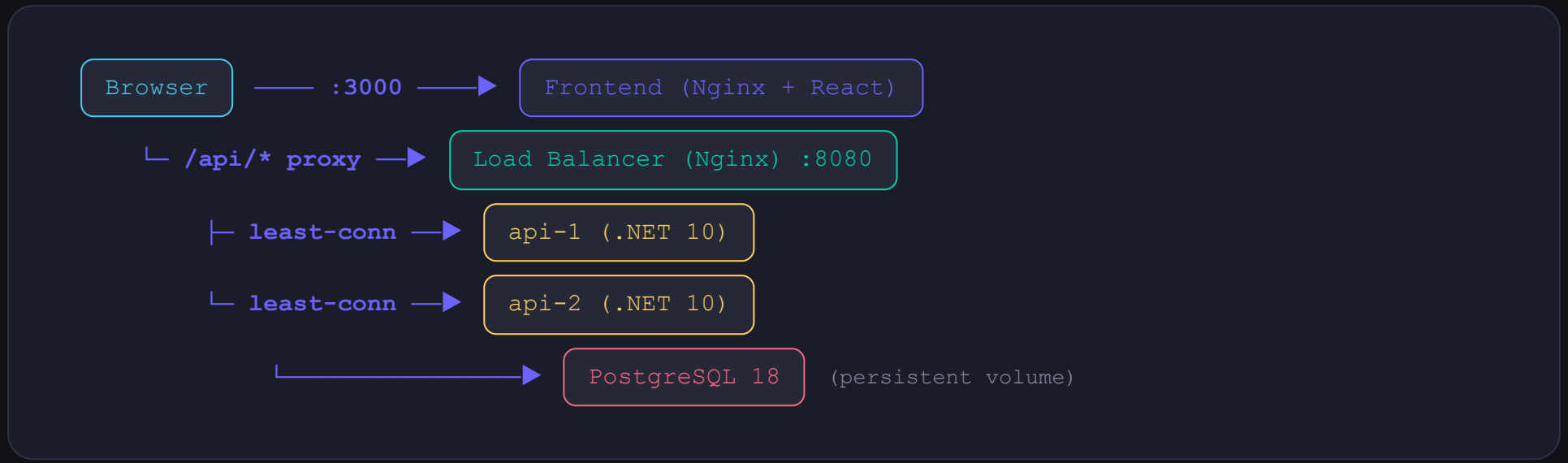
📦

Ein-Befehl-Start

`docker compose up --build` – alles startet automatisch mit Health Checks.

ARCHITEKTUR

Wie die Komponenten zusammenspielen



Client · Präsentation · Routing · Compute · Persistenz

SICHERHEIT

JWT-Authentifizierung & Rate Limiting

🔐 **JWT Access- & Refresh-Token** – kurze Access-Token werden beim Login ausgestellt und über rotierende Refresh-Token erneuert.

🚦 **Rate Limiting auf Auth-Endpunkten** – `/api/auth/register`, `/api/auth/login` und `/api/auth/refresh` werden gedrosselt, um Missbrauch zu verlangsamen.

👤 **Nutzerspezifische Daten** – jede Bookmark-Abfrage filtert nach der aus dem JWT extrahierten Benutzer-ID. Nutzer können nie gegenseitig Daten lesen oder ändern.

🌍 **Umgebungsbasierte Secrets** – `POSTGRES_PASSWORD` und `JWT_KEY` liegen in `.env` (git-ignoriert). Eine `.env.example` liefert sichere Platzhalterwerte.

🏥 **Health- & Readiness-Endpunkte** – `/health` und `/ready` sind öffentlich, nicht authentifiziert und werden von Docker Health Checks verwendet.

API-DESIGN

Minimale API-Endpunkte

Öffentlich

```
POST /api/auth/register
POST /api/auth/login
POST /api/auth/refresh
GET /health
GET /ready
```

Geschützt – JWT erforderlich

```
GET /api/bookmarks/
POST /api/bookmarks/
PUT /api/bookmarks/{id}
DELETE /api/bookmarks/{id}
```

Bookmark-Felder

```
{ "id": 1, "title": "k6 baseline", "url": "https://k6.io", "category": "load-test" }
```

Das Backend gibt außerdem einen `X-Served-By`-Header zurück, der anzeigt, welche Instanz die Anfrage bearbeitet hat.

RESILIENZ

Load Balancing & passives Failover

Nginx verwendet **Least-Connections**-Routing über beide API-Instanzen mit aggressiven Failover-Einstellungen:

```
upstream api_backend {
  least_conn;
  server api-1:8080 max_fails=1 fail_timeout=5s;
  server api-2:8080 max_fails=1 fail_timeout=5s;
}

proxy_connect_timeout 1s;
proxy_next_upstream error timeout http_502 http_503 http_504;
```

⚡ 1 Fehler in 5 s markiert ein Backend als *ausgefallen* – der Load Balancer schaltet sofort um.

🔄 `proxy_next_upstream` wiederholt automatisch bei Netzwerkfehlern und 5xx-Antworten.

👁️ Der `X-Served-By: api-1 | api-2`-Antwort-Header macht das Routing auf einen Blick sichtbar.

INFRASTRUKTUR

Docker Compose Orchestrierung

- **5 Services** – `db`, `api-1`, `api-2`, `load-balancer`, `frontend`. Jeder in einem eigenen Container.
- **Gesundheitsgeprüfte Startreihenfolge** – PostgreSQL muss `pg_isready` bestehen, bevor die APIs starten; beide APIs müssen `/health` bestehen, bevor der Load Balancer Traffic annimmt.
- **Persistentes Volume** – `postgres_data`-Volume bewahrt Daten über `docker compose down`-Neustarts hinweg. Nur mit dem `-v`-Flag löschen, wenn beabsichtigt.
- **Gemeinsames Image** – `api-1` baut das Image; `api-2` verwendet es wieder. Unterschied nur durch die `INSTANCE_NAME`-Umgebungsvariable.
- **Konfiguration per Umgebung** – Connection Strings, JWT-Key und DB-Zugangsdaten werden zur Laufzeit aus `.env` injiziert. Keine Secrets in Images.

TECH-STACK

Was steckt dahinter

Frontend

- React 19.2.6
- Vite 8.0.13
- Nginx 1.30 (Auslieferung)

Backend

- .NET 10 / ASP.NET
- Minimal APIs
- EF Core 10.0.8
- JWT Bearer 10.0.8

Infrastruktur

- Docker Compose
- Nginx 1.30 LB
- PostgreSQL 18

Observability & Tests

- k6 Lasttests
- hey 10.000 Requests
- Health Checks
- X-Served-By-Header

ZUSAMMENFASSUNG

Was dieses Projekt zeigt

Verteilte Architektur

Multi-Instanz-Backend hinter einem echten Load Balancer mit gesundheitsgeprüfter Orchestrierung.

Produktionsstandards

Secrets in Umgebungsvariablen, rate-limitierte Auth, nutzerspezifische Daten, Health-Checks.

Resilienz

Passives Failover mit abgestimmten Timeouts – ein Knoten fällt aus, kein Einfluss auf den Nutzer.

Messbare Performance

k6- und hey-Lasttests validieren p95-Latenz, Fehlerrate und 10.000 JWT-geschützte Requests.

Danke fürs Zuhören · `localhost:3000`