



RaumVote Infrastructure Dashboard

Visibility in the Age of Containers and AI

Distributed Systems · FS26 · OST

Jovin Risch

Was ist RaumVote?

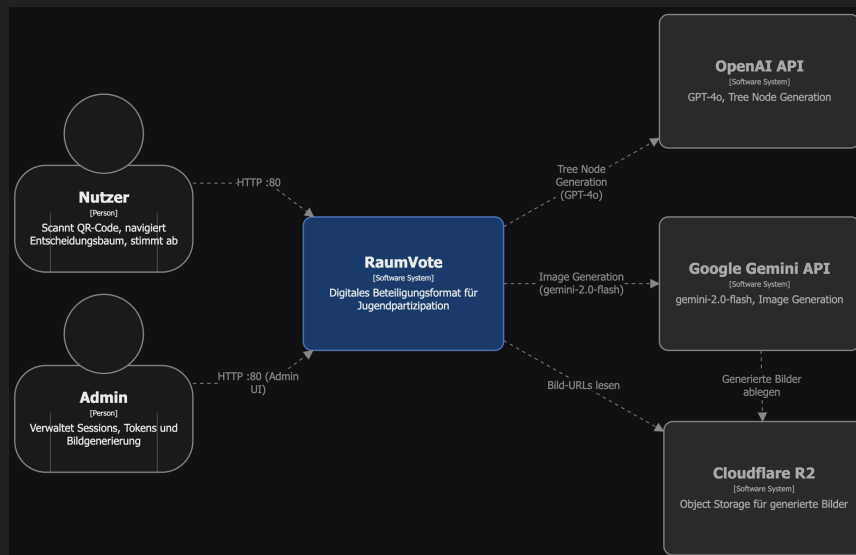
Mobile-first App für **Jugendpartizipation** im öffentlichen Raum. Nutzer navigieren per Swipe durch binäre Entscheidungsbäume und stimmen für eine Raum-Option ab.

Technologien

- › Next.js 16 (App Router), React 19
- › PostgreSQL via Prisma ORM
- › AI-Bildgenerierung (Worker-Prozess)
- › Token-basierte Zugangskontrolle via QR-Code

Cloud-Deployment

- › **Vercel** — Next.js App
 - **Railway** — Worker (lange KI-Antworten)
 - **Neon** — PostgreSQL
- › **OpenAI** — GPT-4o Baumgenerierung
- **Gemini** — Bildgenerierung
- **Cloudflare R2** — Bildspeicher



DSys Requirements

R1 Load Balancing + Failover

nginx Round-Robin, APP_REPLICAS konfigurierbar

R2 Containerisierung

Multi-Stage Dockerfile · 4 Container: DB, 2× App, Worker

R3 Simple Frontend

Infrastructure Dashboard (Grafana, in App eingebettet)

R4 JWT-Authentifizierung (OWASP)

jose, HS256, httpOnly Cookie

R5 Persistente Datenhaltung

PostgreSQL, Docker Volumes (Monitoring)

R6 Single-Command Setup

docker compose up inkl. DB-Migration

R7 Load Test

hey, 6 Szenarien gegen JWT-geschützte Endpoints

Warum der Monitoring-Stack?

Das bestehende RaumVote-Frontend (Voting-UI) wurde bereits im Rahmen von IKTS entwickelt und konnte daher nicht als DSys-Frontend angerechnet werden. Das

Infrastructure Dashboard wurde gezielt für DSys gebaut und erforderte den vollständigen Monitoring-Stack.

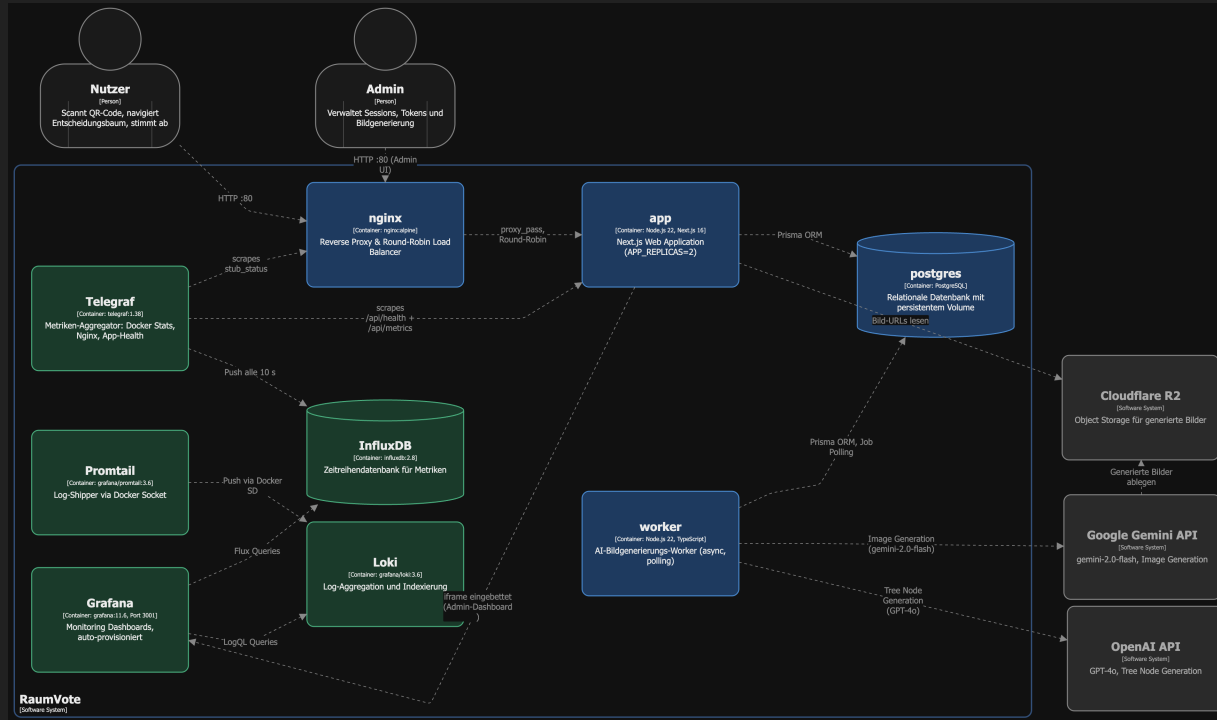
Prod vs. Dev (Docker)

	PROD (IKTS)	DEV (DSys)
Env	Cloud Services	Docker Compose
App	Vercel (Serverless)	2 × Next.js (alpine)
Worker	Railway	1 × Node.js (alpine)
DB	Neon (Cloud PostgreSQL)	1 × PostgreSQL 18.4
NLP API	OpenAI GPT-4o	OpenAI GPT-4o *
T2I API	Google Gemini	Google Gemini *
Monitoring	—	Grafana · InfluxDB · Loki

(*) Getestet, jedoch nicht produktiv eingesetzt: Lokale Alternativen scheiterten an der Komplexität (Referenzbild + Text kombiniert bei mind. 80 % Gemini-Qualität nicht erreichbar).

Systemarchitektur

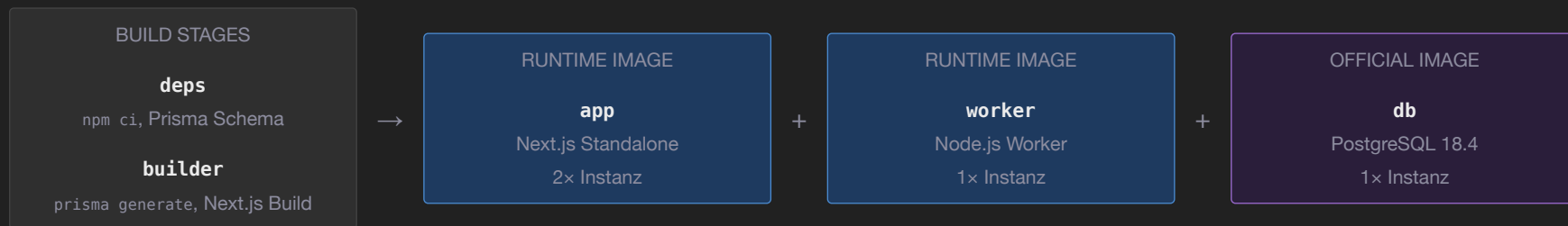
R2



Containerisierung: Multi-Stage Dockerfile

R2

Ein einziges Dockerfile mit vier Stages — davon starten **vier Container**: 1× PostgreSQL (DB), 2× app (Next.js, skalierbar), 1× worker. Die Build-Stages `deps` und `builder` produzieren Artefakte und werden danach verworfen.



- › Build-Tools (TypeScript-Compiler, Next.js Build) landen nie im finalen Image
- › `entrypoint.sh` führt `prisma db push` beim Container-Start aus (automatische Migration)

Load Balancing + Failover

R1

Load Balancing (R1)

- › Docker Embedded DNS (127.0.0.11) für automatische Service-Auflösung
- › Neue Instanzen werden ohne Nginx-Neustart erkannt
- › APP_REPLICAS steuert die horizontale Skalierung
- › X-Forwarded-For Header zeigt die antwortende Instanz

Failover

- › Healthcheck: App-Container melden sich via /api/health
- › Nginx leitet erst weiter, wenn Container healthy
- › Fällt eine Instanz aus, übernehmen die verbleibenden
- › Getestet: docker stop raumvote-app-1, Last läuft weiter

Instanz-Lebenszyklus

```
docker compose up
entrypoint: prisma db push → node server.js
Docker prüft :3000/api/health alle 10 s
nginx startet erst wenn healthy
```

Instanz fällt aus

Docker DNS entfernt IP aus app-Eintrag
nginx löst beim nächsten Request neu auf
Traffic geht nur noch an verbleibende Instanz

Instanz kommt zurück

prisma db push läuft erneut (idempotent)
nach Healthcheck: DNS registriert neue IP
Round-Robin läuft wieder auf beiden Instanzen

Voter-Flow · rv-jwt · 30 Tage

1. Admin generiert UUID-Token (QR-Code)
2. Nutzer scannt QR → `POST /api/auth/login`
3. UUID wird in der DB validiert
4. `voterHash = SHA-256(PEPPER:UUID)`
5. `JWT sub = voterHash, HS256`
6. `httpOnly-Cookie rv-jwt (30 Tage)`
7. Alle API-Routen verifizieren Cookie

Admin-Flow · rv-admin-jwt · 8 h

1. Admin gibt `ADMIN_SECRET` im Login ein
2. `POST /api/admin/auth/login`
3. Server prüft gegen Env-Variable
4. `JWT Issuer raumvote-admin, HS256`
5. `httpOnly-Cookie rv-admin-jwt (8 h)`
6. Alle `/api/admin/*` Routen prüfen Cookie
7. `POST /api/admin/auth/logout` löscht Cookie

Privacy by Design — Votes, Likes, Kommentare speichern nur `voterHash`. Rohe UUID nie persistiert.

30 Tage — Nutzer sollen während der gesamten Beteiligungsphase eingeloggt bleiben, ohne sich erneut anzumelden.

OWASP — `httpOnly · SameSite: strict · secure` in Prod · getrennte Issuer verhindern Token-Verwechslung.

8 Stunden — deckt eine Admin-Arbeitssitzung ab; danach automatischer Logout.

Persistente Datenhaltung + Single-Command Setup

R5

R6

Persistent Storage (R5)

- › **PostgreSQL 18.4** im Docker-Stack — lokaler Container, kein externe Abhängigkeit
- › Prisma ORM: Schema-Migrationen, typsichere Queries
- › Docker Volumes für Monitoring-Daten:
 - › `influxdb-data`: Zeitreihendaten
 - › `loki-data`: Log-Archive
 - › `grafana-data`: Dashboard-Konfiguration

Monitoring-Daten überleben Container-Neustarts. Erst `docker compose down -v` löscht Volumes.

Single-Command Setup (R6)

```
# App-Stack
docker compose up

# App + Monitoring
docker compose -f docker-compose.yml -f docker-
compose.monitoring.yml up
```

- › `prisma db push` läuft automatisch beim Start
- › Grafana Datasources und Dashboards auto-provisioniert
- › Einzige Voraussetzung: `.env`-Datei mit Secrets

Monitoring Stack

R3

Zwei Datentypen brauchen zwei Pipelines — beide landen in **Grafana** (Port 3001).

METRIKEN — Zahlen über Zeit

Telegraf

Collector

Liest CPU, RAM und Netzwerk aller Docker-Container alle 10 Sekunden aus



InfluxDB

Datenbank für Metriken

Speichert die Messwerte zeitgestempelt — optimiert für Abfragen wie «letzten 30 Min. CPU»



Grafana

Dashboard

Liest aus InfluxDB und zeigt Echtzeit-Grafiken: CPU, RAM, Req/s, Latenz

LOGS — Text-Ereignisse

Promtail

Log-Collector

Liest den Text-Output (stdout) aller Container in Echtzeit — z.B. nginx-Requests, Fehler



Loki

Datenbank für Logs

Speichert und indiziert Logs nach Container und Status-Code — durchsuchbar in Grafana



Grafana

Dashboard

Zeigt Log-Einträge gefiltert nach Zeit, Container oder Fehlermeldung

Load Test Ergebnisse

R7

Tool: hey (externer CLI-Benchmark). 2× App-Instanzen, nginx Round-Robin, DB Pool 10 pro Instanz, PostgreSQL (Docker). Zusätzlich: integrierter Admin-Benchmark für Pre/Post-Failover-Vergleich.

Szenario	Was wird gemessen?	Req	Conc.	Req/s	Avg	P99
Baseline (/api/auth/me)	JWT-Check, kein DB-Zugriff	1000	50	1813	24 ms	58 ms
High Concurrency	200 gleichzeitige Verbindungen	2000	200	2682	64 ms	137 ms
Sustained Load (5000 Req)	Dauerlast, höherer Gesamtdurchsatz	5000	100	3132	30 ms	74 ms
Vote Status (JWT + DB)	JWT + PostgreSQL-Query (End-to-End)	1000	50	2350	20 ms	36 ms

Beobachtungen

- › Vote Status nur 3 ms langsamer als Baseline — lokale DB eliminiert Netzwerk-Roundtrip
- › High Concurrency: P99 springt auf 137 ms → Connection-Pool-Queueing bei > 20 gleichzeitigen DB-Queries sichtbar
- › 0 Fehler über alle 9000 Requests

Failover-Verhalten (Admin-Benchmark)

- › docker stop raumvote-app-1 während Pre/Post-Test
- › nginx leitet sofort an App-2 weiter
- › 0 fehlgeschlagene Requests
- › Durchsatz -14 bis -31% (nicht -50%: Pool-Effizienz pro Instanz steigt)

Design-Entscheidungen

Infrastructure as Code

- › Grafana Datasources und Dashboards werden beim Start automatisch provisioniert, keine manuelle Konfiguration nach `docker compose up`
- › Alle Credentials über Umgebungsvariablen

Separation of Concerns

- › Zwei `docker-compose`-Files: App-Stack vs. Monitoring-Stack
- › Monitoring kann unabhängig gestartet und gestoppt werden

Observability First

- › Strukturiertes JSON-Logging in Nginx, Next.js und Worker
- › `X-Instance-Id` und `X-Upstream-Addr` für Request-Traceability
- › Telegraf als universeller Pull-basierter Metriken-Aggregator

Privacy by Design

- › Raw UUID verlässt nie den Server
- › Alle DB-Records verwenden `voterHash` (SHA-256)
- › `VOTER_PEPPER` verhindert Brute-Force-Angriffe auf Hashes

Demo

docker compose up · Grafana · Load Test

Demo

1 Stack starten

```
docker compose -f docker-compose.yml -f docker-compose.monitoring.yml up
```

App · Worker · nginx · PostgreSQL · Grafana · InfluxDB · Loki

2 Was ist RaumVote?

<http://localhost> — QR-Login scannen · Entscheidungsbaum navigieren · Option wählen und abstimmen

3 Admin Panel & Statistiken

<http://localhost/admin> — Tokens · Session · Infra-Dashboard · Grafana-Embed (Nginx, Load Test, Logs)

4 Load Test — Failover

Admin › Infra › Load Balancer: **Pre-Test starten**

In neuem Terminal: `docker stop raumvote-app-1`

Post-Test starten — Vergleich: Durchsatz –14 bis –31%, 0 Fehler

KI-Unterstützung

Inhalte dieser Präsentation sowie Teile des Codes wurden mit Unterstützung von KI-Werkzeugen erstellt, überprüft und überarbeitet:

Anthropic Claude Code	Architekturentscheide, Implementierung, Debugging, Präsentationsstruktur
OpenAI ChatGPT	Konzeptfragen, inhaltliche Recherche, Formulierungen

Alle Inhalte wurden vom Autor geprüft und verantwortet. Die Verantwortung für Richtigkeit und Vollständigkeit liegt beim Autor.

Fragen?

Jovin Risch