Architektur & Skalierungsplan für Next.js Cybersicherheits-Scanner

Dieses Dokument erklärt, wie du eine Cybersicherheits-Scanner-App mit Next.js entwickeln kannst, die zunächst 100.000 Nutzer unterstützt und später auf 1.000.000 Nutzer skaliert.

1. Start mit 100.000 Nutzern

Für 100.000 Nutzer reicht eine einfachere Architektur: - Next.js als Frontend & API für Status-Abfragen - Separater Node.js-Service für die eigentliche Scanner-Logik - Eine zentrale Datenbank (z. B. PostgreSQL) - Redis als Cache für schnelle Statusupdates - Job-Queue (BullMQ oder RabbitMQ) für asynchrone Verarbeitung der Scans - Hosting: Vercel/Netlify für Frontend, AWS/GCP/DigitalOcean für Backend

Komponente	Technologie	Funktion
Frontend	Next.js	UI, Statusabfragen
Backend	Node.js (Express/NestJS)	Scanner-Logik, API für Scans
Datenbank	PostgreSQL/MongoDB	Speichert Nutzer & Ergebnisse
Queue	BullMQ/RabbitMQ	Verwaltet Scan-Jobs
Cache	Redis	Schnelle Statusupdates

2. Skalierung auf 1.000.000 Nutzer

Ab 1.000.000 Nutzern ist eine horizontale Skalierung notwendig: - Load Balancer vor dem Frontend & Backend - Mehrere Instanzen für Node.js-Scanner (Microservices) - Automatisches Skalieren mit Kubernetes oder AWS ECS - Separate Datenbank-Cluster (Read-Replica) - CDN für statische Inhalte - Erweitertes Monitoring (Prometheus, Grafana) - Logging-System (ELK-Stack)

Schicht	Technologie	Erweiterung für 1.000.000 Nutzer
Frontend	Next.js + CDN	Globale Verteilung & Caching
Backend	Node.js Microservices	Horizontale Skalierung & Queue-Verarbeitung
Orchestrierung	Kubernetes / Docker Swarm	Automatisches Skalieren
Datenbank	PostgreSQL Cluster	Read Replicas & Sharding
Queue & Cache	Redis/RabbitMQ Cluster	Mehrere Worker-Nodes