🚀 Performanceeinbruch im Gateway – Analyse und Optimierung

🧭 Hintergrund

Nach der Anbindung eines neuen Nutzungskanals traten am zentralen API-Gateway deutliche Performanceeinbrüche auf: Der Durchsatz sank, einzelne Requests blockierten länger als erwartet und die Latenzverteilung wurde zunehmend instabil. Da der Bereich geschäftskritisch war, musste die Ursache schnell eingegrenzt werden.

🔍 Analyse

Ich führte eine **systematische Analyse** der Gateway-Metriken und Trace-Daten durch. Dabei fiel ein Kanal mit außergewöhnlich vielen blockierenden Aufrufen auf. Die Untersuchung der relevanten **Codepfade** und **Prozessabläufe** deutete darauf hin, dass sich ein möglicher Engpass im Kommunikationspfad zwischen dem Gateway und dem Authentifizierungsdienst befunden haben könnte.

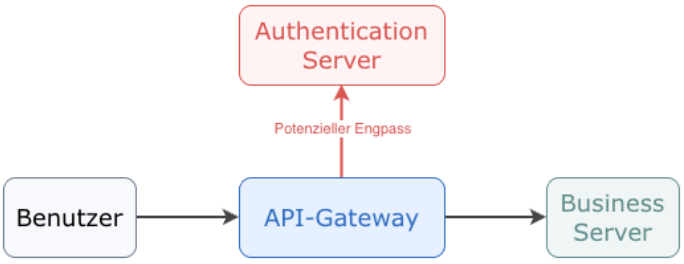

Abbildung 1: Potenzieller Engpass

🔧 Maßnahmen

Um diese Beobachtungen reproduzierbar zu validieren, entwickelte ich eine **kompakte Testumgebung auf Komponentenebene**. Sie kombinierte ein **kontrollierbares Lastmodell**, **klar definierte Testketten** und **reproduzierbare Messpunkte** und bildete die **Grundlage** für eine strukturierte Analyse des kritischen Request-Pfads.

Für die technische Analyse setzte ich einen kombinierten Werkzeug- und Methodenstapel ein:

- **wrk** für realitätsnahe Last- und Request-Simulation
- **Docker** zur standardisierten Bereitstellung einer Testumgebung
- **JProfiler** zur Identifikation blockierender oder ineffizienter Abschnitte
- **JMH** für Microbenchmarks einzelner Codepfade
- **Confluence** zur sauberen Dokumentation aller Messpunkte, Testketten und Analyseergebnisse

Die Untersuchungen zeigten mehrere Ursachen, die gemeinsam zur Performanceverschlechterung beitrugen:

- **Unnötige** oder **blockierende Synchronisation**, die unter Last blockierende Effekte erzeugte
- **Synchrone Log-Ausgaben**, die IO-bedingte Verzögerungen verursachten
- **Ineffiziente UUID-Generierung unter Java 8**, die bekannte Entropie- und Performanceprobleme aufweist
- **Wachsende Routing-Tabellen** im Gateway erhöhten die Lookup-Kosten

Auf Basis der Analyse leitete ich die notwendigen Maßnahmen ein. Ich übernahm die **technische Federführung**: identifizierte die kritischen Stellen, definierte die Lösungsansätze und nahm selbst einen Teil der **Code- und Konfigurationsanpassungen** vor. Weitere Änderungen wurden von Teamkollegen umgesetzt, wobei ich die technische Richtung, Priorisierung und Verifikation koordinierte.

Die Wirksamkeit aller Anpassungen überprüfte ich in **mehreren Lasttest-Iterationen** und stimmte die Ergebnisse teamübergreifend ab.

📈 Ergebnisse

- Rund **200 %** höherer Durchsatz und **deutlich stabilere Latenzverteilung**
- **Etablierung** eines reproduzierbaren **Analyse- und Testverfahrens** für zukünftige Gateway-Untersuchungen
- Veröffentlichung eines **technischen Leitfadens**, der teamübergreifend zur schnelleren Identifikation ähnlicher Engpässe beiträgt

Als Weiterentwicklung dieser Arbeit entstanden zwei **Open-Source-Projekte**:

- **gateway-bottleneck-lab** – offiziell veröffentlichte Open-Source-Version meiner Gateway-Testmethodik

- **uuid-benchmark** – Analyse verschiedener UUID-Implementierungen und deren Performanceverhalten

📎 **Weiterführend: Gateway-Bottleneck-Lab** (Open Source)