🔧 Graceful Shutdown

⭐ Kurzüberblick

- **Problem**: Rolling Deployments führten trotz Spring-Cloud-Retries zu **unnötigen Latenzen** und **potenziellen Kaskadeneffekten**.
- **Beitrag**: Entwicklung eines **event-driven** Shutdown-Moduls zur kontrollierten Traffic-Abmeldung.
- **Wirkung**: Stabilere Deployments ohne Fehlversuche, reduzierte Latenzspitzen und geringeres Betriebsrisiko.

🧭 Hintergrund

Spring Cloud bot eingebaute Retry-Mechanismen bereit, um kurzfristige Nichtverfügbarkeiten während eines Rolling Deployments abzufangen. In der Praxis führte dies jedoch zu vermeidbaren Verzögerungen und konnte in großen Umgebungen Kaskadeneffekte auslösen.

In typischen Standardkonfigurationen kann es bis zu 179–239 Sekunden dauern, bis eine abgeschaltete Instanz vollständig aus allen beteiligten Caches und Clients verschwunden war:

1. Instanz meldet sich ab: 0 s
2. Erste Ablaufprüfung: 29 s
3. Zweite Ablaufprüfung: 89 s
4. Dritte Ablaufprüfung: 149 s
5. Schreib-Cache → Read-Only-Cache: 179 s
6. Client aktualisiert Read-Only-Cache: 209 s
7. Load-Balancer aktualisiert lokale Routingtabelle: 239 s

> Erklärung zum Mindestwert (179 s):
> Bei einem kontrollierten Shutdown (kein `Kill -9` , kein Stromverlust) übersprang der Service die ersten Prüfzyklen. Der Prozess begann direkt bei Schritt 4.

Während dieses Zeitfensters konnten weiterhin Requests an die bereits abgeschaltete Instanz gesendet werden. Retries verhinderten zwar Ausfälle, erhöhten aber die Latenz und erzeugten potenzielle Kaskaden.

In großen Architekturen war es daher **technisch sinnvoll** und **wirtschaftlich gerechtfertigt**, einen event-driven Graceful-Shutdown-Mechanismus einzuführen.

🛠️ Technische Umsetzung

Der Mechanismus bestand aus drei abstrahierten Bausteinen:

1. CI/CD-Pipeline – Integration über Shell-Skript
2. Event-Bus – verteilt Ereignisse zwischen den Komponenten
3. Service-Registry - Service und Client

Diese Struktur ermöglichte einen vollständig event-driven Ablauf.

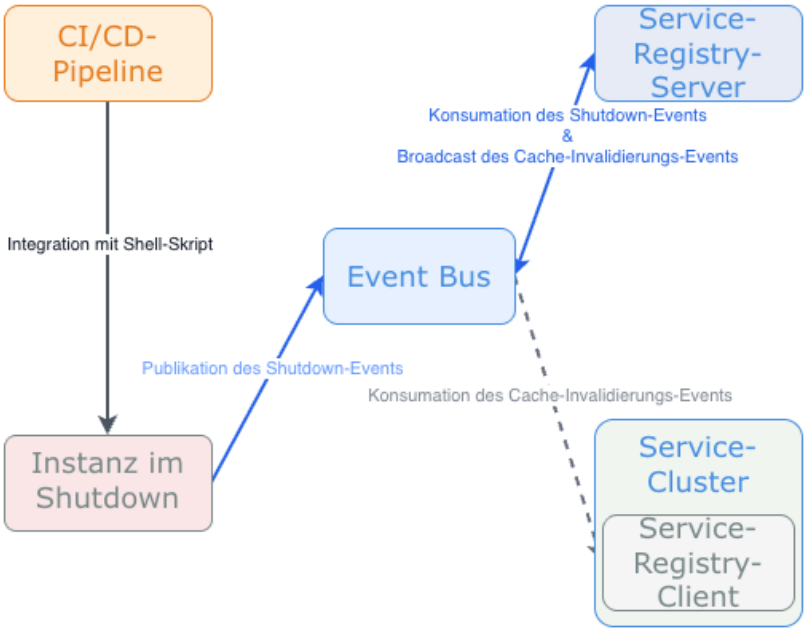

Abbildung 1: Abstrakt Ablauf des event-driven Graceful-Shutdown-Mechanismus

🔄 Ablauf

1. CI/CD-Pipeline → Instanz (im Shutdown)
   Ein **Shell-Skript** stieß ein Shutdown-Event an, sobald ein Deployment startete.
2. Instanz (im Shutdown) → Event-Bus
   Diese Instanz publizierte **Shutdown-Event** an den Event-Bus.
3. Event-Bus → Registry-Server
   Der Registry-Server konsumierte das Shutdown-Event und entfernte die Instanz sofort aus seinen Registry-tabellen. Anschließend broadcastete er ein Cache-Invalidierungs-Event.
4. Event-Bus → Registry-Clients im Cluster
   Alle Clients konsumierten das Cache-Invalidierungs-Event und aktualisierten ihre Routingtabellen unmittelbar.
5. Instanz (im Shutdown) fuhr kontrolliert herunter

📈 Ergebnisse

- Keine unnötigen Retries mehr während Deployments
- Stabilere & planbare Deployments, selbst unter Last
- Weniger Betriebsrisiko, da abgeschaltete Instanzen sofort aus dem Traffic entfernt werden