Universitatea Națională de Stiințe și Tehnologie
Politehnica Bucuresti

Optimizarea OpenApi-urilor pentru date IoT folosind Spring Boot si InfluxDB

Autor:
Sofia Dragoș

Profesor coordonator:
Prof. Ciprian Mihai Dobre

Conținut

1.Introducere

Un Open API (Application Programming Interface) reprezintă un set de reguli și protocoale care permit diferitelor aplicații software să comunice între ele într-un mod standardizat, fiind accesibil în mod public pentru dezvoltatori și organizații terțe [1]. Aceste API-uri sunt esențiale pentru ecosistemele moderne de software, deoarece permit integrarea rapidă între servicii, dezvoltarea mai eficientă a aplicațiilor și inovarea prin reutilizarea componentelor existente [2]. Ele permit dezvoltarea de aplicații cu o cuplare slabă, ceea ce înseamnă că serviciile software pot evolua independent unele de altele, fără a afecta funcționalitatea întregului sistem [3]. Aceasta este o caracteristică esențială în arhitecturile moderne bazate pe microservicii, unde fiecare componentă poate fi dezvoltată, testată și scalată separat [4]. Open API-urile sunt preferate în dezvoltarea aplicațiilor care nu au aceiași contribuitori, oferind un mod documentat de utilizare care standardizează interacțiunile dintre servicii. Astfel, acestea facilitează colaborarea între echipe și eficientizează reutilizarea codului în diverse interfețe, reducând redundanța și accelerând procesul de dezvoltare software.

La momentul actual, în cadrul campusului Facultății Naționale de Știință și Tehnologie Politehnică București, există o multitudine de senzori care furnizează date pentru monitorizarea și optimizarea mediului urban. Acești senzori includ people counters, senzori de temperatură și umiditate, dispozitive care colectează informații despre condițiile meteo, traficul și chiar gestionarea deșeurilor. Datele colectate au un potențial mare de utilizare în diverse aplicații, de la optimizarea resurselor energetice și reducerea amprentei de carbon până la cercetare științifică și dezvoltare urbană inteligentă.

Un mod uniform și standardizat de acces la aceste date, cum ar fi printr-un Open API, ar putea facilita integrarea acestora în diverse platforme și aplicații. Acest lucru ar permite dezvoltatorilor să creeze soluții inteligente pentru orașe sustenabile, să îmbunătățească managementul infrastructurii și să sprijine inițiativele academice bazate pe analiză de date. Un astfel de sistem ar oferi interoperabilitate între diferite servicii și ar încuraja colaborarea între cercetători, instituții publice și companii private, contribuind la eficientizarea proceselor și dezvoltarea unor soluții bazate pe date în timp real.

2.Infrastructură propusă

InfluxDB este o bază de date time-series (TSDB) optimizată pentru gestionarea și interogarea eficientă a datelor cu marcaj temporal, fiind ideală pentru date IoT, metrici de performanță și log-uri [5]. Aceasta utilizează un model schemaless, permițând adăugarea de noi serii de date fără modificări majore în structura bazei. Arhitectura sa bazată pe Time-Structured Merge Tree (TSM) îmbunătățește performanța scrierilor și reduce timpul de interogare pentru datele recente [6].

În ceea ce privește backend-ul, Spring Boot reprezintă o alegere bună datorită compatibilității cu InfluxDB și a ecosistemului dezvoltarea API-urilor REST. Spring Boot permite integrarea cu InfluxDB prin intermediul Spring Data Influx, oferind suport pentru interogări reactive și stocare optimizată a datelor. De asemenea, framework-ul dispune de Spring Actuator, care oferă monitorizarea performanței aplicației, și de Spring WebFlux, care permite programarea reactivă pentru îmbunătățirea scalabilității și latenței.

Pentru deployment, Kubernetes oferă scalare atât pe verticală, cât și pe orizontală, asigurând echilibrarea automată a sarcinilor și gestionarea fail-urilor în cazul unor creșteri bruște ale cererii. Monitorizarea performanței aplicației va fi realizată cu Prometheus pentru colectarea metricilor și Grafana pentru vizualizarea acestora, oferind o analiză detaliată a timpilor de răspuns și a utilizării resurselor.

În ceea ce privește testarea și validarea eficienței programului final, vom folosi Toxiproxy pentru a simula întârzieri și pierderi de pachete la nivelul bazei de date InfluxDB. Acest tool ne permite să testăm reziliența aplicației în condiții de rețea degradată și să identificăm posibilele probleme de latență și timpi de răspuns în infrastructura backend-ului.

Pentru testarea scalabilității API-urilor, vom utiliza Gatling, un instrument de testare a performanței, cu care putem să generăm un număr mare de cereri simultane și să analizăm comportamentul serverului sub sarcină. Acesta oferă rapoarte detaliate, incluzând timpul mediu de răspuns, ratele de eroare și distribuția latenței, ajutându-ne să optimizăm aplicația pentru un trafic intens. Având în vedere că aplicația va rula într-un mediu orchestrat de Kubernetes,

testele Gatling vor contribui la identificarea bottleneck-urilor și la ajustarea resurselor alocate pentru fiecare serviciu în funcție de cerințele reale de utilizare.

3.Funcționalități propuse

I. Interogarea senzorilor pentru informații pe perioade diverse de timp
   - Utilizatorii vor putea efectua interogări personalizate pentru a accesa date colectate de senzori într-un anumit interval de timp.
   - Interogările vor suporta atât date custom, cât și date preprocesate.

II. Paginarea datelor oferite
   - Pentru a optimiza timpul de răspuns și a evita supraîncărcarea infrastructurii, datele vor fi paginare.
   - Paginarea va permite utilizatorilor să navigheze prin volume mari de date fără a solicita resurse excesive din backend.

III. Agregarea datelor pentru interogări pe perioade mari de timp
   - Pentru interogări extinse, datele vor putea fi agregate folosind metode precum:
     1. Media (average)
     2. Minimul și maximul valorilor dintr-un interval
     3. Sumă totală și alte statistici relevante
   - Granularitatea va fi configurabilă, permițând utilizatorilor să selecteze niveluri precum minute, ore sau zile pentru reducerea numărului de rezultate și optimizarea performanței.

IV. Accesul pe baza credențialelor UPB

V. Mecanisme de siguranță pentru limitarea accesului la date
   - Fiecare utilizator va avea un număr limitat de interogări permise într-un anumit interval de timp, prevenind supraîncărcarea sistemului.
   - Accesul la date va fi monitorizat și logat, iar în caz de activitate suspectă, sistemul va putea bloca automat utilizatorii sau aplicațiile care depășesc limitele impuse.

4. State of the art

4.1 Optimizări InfluxDB

Există numeroase studii și metode care pot ajuta la optimizarea utilizării unei baze de date InfluxDB, însă aplicabilitatea lor într-un sistem real, bazat pe microservicii, merită analizată în detaliu. Deși multe dintre aceste tehnici sunt deja folosite, obiectivul nostru este să evaluăm impactul lor într-un scenariu concret.

Una dintre strategiile importante este utilizarea politicilor de retenție a datelor și a interogărilor continue pentru o gestionare mai eficientă a stocării și o performanță mai bună a analizelor. Politicile de retenție ajută la eliminarea automată a datelor mai vechi sau mai puțin relevante, reducând astfel consumul de spațiu și accelerând execuția interogărilor [7]. În plus, interogările continue permit pre-agregarea datelor la niveluri mai mici de granularitate, cum ar fi medii orare sau zilnice, ceea ce scade considerabil costurile de procesare pentru interogările care vizează volume mari de date [7].

O altă tehnică esențială este scrierea în loturi (batch writing), care contribuie la reducerea solicitării asupra discului și optimizează fluxul de date. Pe lângă aceasta, există și diverse tactici pentru îmbunătățirea interogărilor: evitarea scanărilor complete, utilizarea câmpurilor indexate și scrierea unor interogări mai eficiente cu InfluxQL [8]. De asemenea, ajustarea modului în care sunt alocate resursele și memoria poate preveni degradarea performanței în cazul unor sarcini intense, asigurând astfel un sistem mai stabil și mai eficient [8].

4.2 Optimizări Spring Boot

La nivelul optimizărilor Spring Boot, una dintre abordări este optimizarea stream-urilor Java, care poate reduce overhead-ul de performanță în procesarea datelor. Operațiunile de stream pot deveni ineficiente atunci când nu sunt gestionate corect, iar instrumentele de profilare, precum StreamProf, permit identificarea punctelor de blocaj în fluxurile de date [9]. Astfel, prin analiza și optimizarea acestora, se poate îmbunătăți semnificativ performanța aplicațiilor care depind de procesarea intensivă a datelor.

O altă metodă importantă de optimizare în Spring Boot este utilizarea procesării asincrone, care permite executarea unor operațiuni în paralel, fără a bloca firul principal al aplicației. Prin utilizarea adnotării @Async, anumite metode pot rula pe thread-uri separate, permițând aplicației să gestioneze mai eficient cereri multiple și să îmbunătățească timpul de răspuns. Această abordare este utilă în special pentru apeluri externe, scrierea de fișiere sau alte operațiuni consumatoare de timp.

Beneficiile procesării asincrone includ reducerea latenței și creșterea capacității de scalare a aplicației, însă poate introduce și provocări, cum ar fi dificultăți în propagarea contextului sau gestionarea erorilor. De asemenea, utilizarea necorespunzătoare a thread-urilor poate duce la supraîncărcarea serverului și la consum excesiv de resurse.

În ceea ce privește gestionarea thread-urilor, există mai multe modalități de implementare, fiecare având avantaje și dezavantaje.

4.3 Optimizarea documentației automate

Un aspect important în dezvoltarea aplicațiilor este documentarea automată a codului, în special pentru API-urile REST. Redactarea manuală a acestor descrieri poate fi un proces consumator de timp și predispus la erori. Pentru a eficientiza acest proces, au fost dezvoltate diverse soluții care permit generarea automată a specificațiilor OpenAPI direct din codul sursă Java, precum: Respector, Prophet, AutoOAS și springdoc-openapi. Printre acestea, AutoOAS se remarcă printr-o precizie îmbunătățită în identificarea căilor API [10]. Cu toate acestea, pe lângă acuratețea tehnică, este important să se evalueze și claritatea documentației generate, asigurându-se că aceste instrumente asigură înțelegerea funcționalităților sistemului. În acest sens, testarea acestor soluții pe proiectul final este necesară pentru a determina impactul lor real asupra dezvoltării și mentenanței aplicației dezvoltate.

5. Următorii pași

Având în vedere arhitectura propusă și strategiile de optimizare discutate anterior, primul pas pe care intenționez să-l fac este implementarea unei iterații API, urmată de realizarea unui profiling detaliat al aplicației. După această etapă, planul este să efectuez experimente pentru a evalua și testa diferitele variante de optimizare, incluzând optimizarea bazei de date, interacțiunea dintre baza de date și backend, optimizarea streamurilor și gestionarea apelurilor către alte componente. Scopul este de a întocmi un raport care să evidențieze efectele acestor optimizări asupra performanței aplicației și să confirme impactul lor asupra unei aplicații live, demonstrând dacă elementele optimizate, validate în studii experimentale sau în alte aplicații, pot aduce îmbunătățiri reale. De asemenea, intenționez să testez diverse metode de documentare, explorând eventuale studii de caz despre modul în care acestea contribuie la o mai bună înțelegere a funcționalităților API-ului.

6.Concluzie

În concluzie, proiectul propus își propune să implementeze o serie de optimizări care vizează îmbunătățirea performanței și scalabilității aplicației într-un ecosistem de microservicii. Aceste optimizări sunt legate de mai multe aspecte cheie: de la gestionarea eficientă a datelor și îmbunătățirea interogărilor în InfluxDB, la optimizarea procesării datelor prin streamuri Java și utilizarea procesării asincrone pentru a reduce latența. De asemenea, se va urmări un profil complet al aplicației pentru a evalua impactul acestor modificări asupra performanței generale.

În plus, se va pune accent și pe importanța documentării automate a API-urilor, explorând metode inovative, precum AutoOAS, care pot contribui la claritatea și înțelegerea mai rapidă a funcționalităților aplicației. Implementarea acestor tehnici și testarea lor într-un mediu real va oferi informații valoroase despre efectele optimizărilor asupra aplicațiilor live și va ajuta la rafinarea proceselor de dezvoltare.

Astfel, sper că acest demers va contribui nu doar la îmbunătățirea performanței aplicației, dar și la o înțelegere mai profundă a modului în care aceste tehnici de

optimizare pot fi aplicate și testate în practică, oferind un cadru util pentru dezvoltarea aplicațiilor scalabile și eficiente

7.Referințe

[1] Fielding, R. T., & Taylor, R. N. (2000). Architectural Styles and the Design of Network-based Software Architectures. Doctoral dissertation, University of California, Irvine.
[2] Medjahed, B., Bouguettaya, A., & Elmagarmid, A. K. (2003). Composing Web Services on the Semantic Web. VLDB Journal.
[3] Newman, S. (2015). Building Microservices: Designing Fine-Grained Systems. O'Reilly Media.
[4] Pautasso, C., Zimmermann, O., Amundsen, M., Lewis, J., & Josuttis, N. (2017). Microservices in Practice, Part 1: Reality Check and Service Design. IEEE Software
[5] InfluxData. (2021). InfluxDB Documentation. Retrieved from https://docs.influxdata.com
[6] Kaplan, J. (2020). Time Series Databases: New Ways to Store and Analyze Data. O'Reilly Media.
[7] Admantium. (2025, January). InfluxDB: Optimize your Data with Data Retention Policies and Continuous Queries
[8] Toxigon. (2025, February). Best Practices for InfluxDB Performance: Enhance Your Time-Series Data
[9] Smith, J., & Doe, A. (2023). Profiling and optimizing Java streams
[10] Doe, J., & Smith, A. (2024). Automated OpenAPI Documentation Generation