Arhitectura Monolitică vs. Microservicii

Grigore Cosmin, Valentin Radu, Cristian Bodocan

# Descrierea Scenariilor

Proiectul a fost conceput pentru a compara performanța arhitecturilor monolitice și microservicii, folosind o aplicație de e-commerce ca exemplu practic. Aplicația include funcționalități precum autentificarea utilizatorilor, gestionarea produselor, plasarea comenzilor și utilizarea unui coț de cumpărături.

Scopurile principale au fost:

* Evaluarea performanței și scalabilității fiecărei arhitecturi.
* Identificarea avantajelor și dezavantajelor fiecărei abordări.
* Propunerea unor recomandări pentru utilizarea optimă a fiecărei arhitecturi.

## Scenarii Implementate

1. **Arhitectura Monolitică**: Toate funcționalitățile sunt integrate într-o singură aplicație.
2. **Arhitectura Microservicii**: Aplicația este divizată în module independente, fiecare servind o funcționalitate specifică și comunicând prin protocoale REST.

# State of the Art

## Arhitectura Monolitică

* Integrează toate funcționalitățile într-un singur proces.
* Simplifică dezvoltarea inițială și implementarea.
* Limită scalabilitatea datorită dependințelor complexe și dificultății de a extinde părți individuale.

## Arhitectura Microservicii

* Aplicația este divizată în servicii independente care pot fi dezvoltate, scalate și implementate individual.
* Oferă flexibilitate, dar introduce complexitate în gestionarea comunicațiilor și coordonarea serviciilor.

# Detalii de Implementare

Proiectarea design-ului proiectului a constat în realizarea design-ului aplicației folosind Arhitectura **Monolit** pentru ca mai apoi, aceasta sa fie transformată folosind **Microservicii**.

## Specificațiile aplicației

* **Sistem de autentificare și înregistrare a utilizatorilor:** Utilizatorii pot accesa aplicaţia folosind credențiale, ceea ce ajută la protejarea informaţiilor sensibile şi îmbunătăţeşte experienţa utilizatorului.
* **Dashboard simplist de căutare a produselor**: O interfaţă care oferă o privire de ansamblu asupra produselor disponibile, facilitând alegerea rapidă şi eficientă a acestora.
* **Publicare de produse:** Permite utilizatorilor să introducă produse noi în aplicaţie, actualizând astfel oferta disponibilă pentru clienţi.
* **Plasare si Editare de comenzi**: Oferă o modalitate simplă de a selecta produse şi a finaliza comenzi, esenţială pentru orice aplicaţie e-commerce.

### Monolit

Partea de **Backend** este realizată folosind **NodeJs**, prin intermediul căruia am construit practic un server Restful care comunică cu frontend-ul în mod sincron, prin intermediul protocolului **HTTP.**

Astfel, am creat endpoint-uri pentru autentificare și înregistrare cât și pentru diverse operații precum adaugare de produse și plasare sau editare de comenzi. Partea de autentificare am realizat-o folosind token **JWT**.

Pentru stocarea datelor am folosit baza de date **SQLite** în care am reținut detaliile atât despre utilizatori cât și despre produse și comenzi.

Partea de **Frontend** a fost implementată folosind **Angular**, prin intermediul căruia am reușit să implementăm o interfață simplistă care să respecte specificațiile proiectului și care să ne ajute să obținem măsurătorile și metricile pe care ni le-am propus.

### Microservicii

Backend-ul a fost divizat în mai multe servicii independente, containerizate folosind Docker:

* **Microserviciu de autentificare**: Gestionează conturile utilizatorilor.
* **Microserviciu de produse**: Administrează catalogul de produse.
* **Microerviciu de comenzi**: Procesează comenzile plasate.
* **Microserviciu pentru baza de date**: Stochează datele.
* **Microserviciu Api-Gateway:** Abstractizează conectarea cu fiecare microserviciu.
* **Microservicu Prometheus**: Preia metricile din aplicației și le transmite către Grafana.
* **Microserviciu Grafana:** Vizualizarea metricilor sub diferite forme cum ar fi Dashboard-uri sau grafice.
* **Microserviciu Node-Exporter:** Creeaza metricii despre utilizarea sistemului.

Comunicarea se realizează prin HTTP/REST.

Containerizarea a fost realizată folosind Docker-Compose.

# Despre Metrici și Măsurători

## Metrici Folosite:

1. **Timpul de răspuns**: Durata necesară pentru procesarea solicitărilor utilizatorilor.

|  |  |  |
| --- | --- | --- |
| NUMĂR DE CERERI | tIMP de RăSpUNS (MS) monolit | tIMP de RăSPUNS (mS) microservicii |
| 500 | 225 | 170 |
| 2000 | 510 | 360 |
| 4000 | 1100 | 490 |
| 6000 | 1540 | 760 |
| 8000 | 2050 | 1050 |
| 10000 | 2700 | 1450 |

1. **Utilizarea CPU**.

|  |  |  |
| --- | --- | --- |
| NUMĂR DE CERERI | uTILIZARE CPU (%) monolit | utilizare cpu (%) microservicii |
| 500 | 7 | 3 |
| 2000 | 12 | 5 |
| 4000 | 18 | 12 |
| 6000 | 32 | 20 |
| 8000 | 41 | 26 |
| 10000 | 48 | 35 |

1. **Utilizarea Memorie RAM**.

|  |  |  |
| --- | --- | --- |
| NUMĂR DE CERERI | uTILIZARE memorie (MB) MONOLIT | utilizare memorie (MB) microservicii |
| 500 | 30 | 45 |
| 2000 | 46 | 58 |
| 4000 | 75 | 80 |
| 6000 | 88 | 95 |
| 8000 | 105 | 109 |
| 10000 | 120 | 135 |

1. **Throughput**: Numărul de solicitări procesate pe secundă.

## Metode de Măsurare:

* Monitorizarea în timp real cu Prometheus și vizualizarea datelor în Grafana.
* Testarea performanței utilizând scenarii de trafic simulate folosind tool-uri precum Apache JMeter.

# Rezultate

Aceste rezultate sunt relative la procesarea a 10000 de cereri.

|  |  |  |
| --- | --- | --- |
| Metrică | Arhitectura monolit | Arhitectura microservicii |
| Timpul de răspuns | 2700 ms | 1450 ms |
| Throughput | 3700 cereri/sec | 6900 cereri/sec |
| Utilizare memorie | 120 MB | 135 MB |
| UTILIZARE CPU | 48% | 35% |
| Scalabilitate | Limitată | Ridicată |
| Mentenanță | Greoaie | Facilă |
| Costuri | Scăzute | Moderate |

# Discuție Specifică Scenariului

## Beneficii

* **Monolitic**: Performanță mai bună pentru trafic scăzut; implementare mai simplă.
* **Microservicii**: Scalabilitate și reziliență superioare, ideale pentru aplicații mari.

## Dezavantaje

* **Monolitic**: Scalabilitate limitată; mentenanță dificilă pe termen lung.
* **Microservicii**: Complexitate ridicată în configurare și coordonare.

# Concluzii

Ambele arhitecturi au avantaje și dezavantaje specifice:

* Arhitectura monolitică este ideală pentru proiecte mici sau cu resurse limitate.
* Arhitectura microservicii este mai potrivită pentru aplicații complexe, cu trafic mare și cerințe ridicate de scalabilitate.

Proiectul a demonstrat importanța alegerii arhitecturii în funcție de contextul aplicației și obiectivele acesteia.

# Referințe

* Konrad Gos, Wojciech Zabierowski - “The Comparison of Microservice and Monolithic Architecture” [<https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9109514>]
* Omar Al-Debagy, Peter Martinek - “A Comparative Review of Microservices and Monolithic Architectures” [<https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8928192>]
* Bernardo Andrade, Samuel Santos and Antonio Rito Silva - “From Monolith to Microservices Static and Dynamic Analysis Comparison” [<https://arxiv.org/pdf/2204.11844>]
* Refactor a monolith into microservices [https://cloud.google.com/architecture/microservices-architecture-refactoring-monoliths]
* Docker Documentation - **"Overview of Docker Compose" [https://docker-docs.uclv.cu/compose/]**
* Prometheus Documentation - **"Prometheus Overview" [https://nsrc.org/workshops/2021/sanog37/nmm/netmgmt/en/prometheus/prometheus-overview.pdf]**
* Grafana Documentation - **"Getting Started with Grafana" [**[**https://grafana.com/docs/grafana/latest/getting-started/**](https://grafana.com/docs/grafana/latest/getting-started/)**]**
* Apache JMeter Documentation [https://jmeter.apache.org/]