**Proiect Calcul Paralel Distribuit**

Studenți: Popa Cristian Mihai, Stoian Sorin

Grupa: 30241

Profesor Îndrumător: Dr. Ing. Cosmina Ivan

Cuprins

[Introducere 3](#_Toc167747511)

[1. Aspecte funcționale și de arhitectură 3](#_Toc167747512)

[1.1 Logica de rutare 3](#_Toc167747513)

[1.2 Garantarea livrării 4](#_Toc167747514)

[1.3 Ordonarea mesajelor 4](#_Toc167747515)

[1.4 Transmitere multicast 4](#_Toc167747516)

[1.5 Modul de stocare 5](#_Toc167747517)

[1.6 Suport tranzacțional 5](#_Toc167747518)

[1.7 Gradul de securizare 6](#_Toc167747519)

[2. Aspecte de calitate a serviciului oferit 6](#_Toc167747520)

[2.1 Disponibilitate 6](#_Toc167747521)

[2.2 Performanță 6](#_Toc167747522)

[2.3 Scalabilitate dinamică 6](#_Toc167747523)

[2.4 Latența și debitul 7](#_Toc167747524)

[3. Caracteristici distinctive a celor două tehnologii 7](#_Toc167747525)

[4. Implementare proprie a unei aplicații de chat folosind RabbitMQ 7](#_Toc167747526)

[4.1 Descrierea aplicație 7](#_Toc167747527)

[4.2 Implementare 8](#_Toc167747528)

[Concluzie 9](#_Toc167747529)

# Introducere

În lumea actuală a comunicațiilor digitale, sistemele de mesagerie sunt esențiale pentru diverse aplicații de business și personale. RabbitMQ și Kafka sunt două tehnologii populare în domeniul mesageriei, fiecare având propriile avantaje și dezavantaje. Acest document își propune să compare aceste două tehnologii din perspectiva aspectelor funcționale, de arhitectură și a calității serviciului oferit. De asemenea conține și o secțiune de implementare unde este detaliată și descrisă funcționalitatea unei aplicații de Chat realizată cu ajutorul tehnologiei RabbitMQ.

RabbitMQ este un broker de mesaje open-source care implementează protocolul AMQP (Advanced Message Queuing Protocol). Funcționează prin trimiterea mesajelor de la producători (producers) la consumatori (consumers) prin intermediul unor entități numite exchange-uri și cozi (queues). Producătorii trimit mesaje către exchange-uri, care apoi direcționează mesajele către cozi conform unor reguli de rutare predefinite. Consumatorii preiau mesajele din cozi pentru a le procesa. RabbitMQ este cunoscut pentru flexibilitatea sa în rutarea mesajelor și pentru suportul robust pentru tranzacții și livrarea fiabilă a mesajelor.

Kafka este o platformă distribuită de streaming de date dezvoltată de Apache, care funcționează ca un jurnal distribuit (distributed log). Mesajele sunt produse de producători (producers) și scrise într-un topic, care este împărțit în mai multe partiții pentru a asigura scalabilitatea sistemului. Fiecare partiție păstrează ordinea mesajelor. Consumatorii (consumers) citesc mesajele din partiții în ordinea în care au fost scrise. Kafka se remarcă prin performanța sa ridicată, capacitatea de a gestiona volume mari de date și mecanismele sale de replicare și toleranță la erori, asigurând atât livrarea "cel puțin o dată" (at least once) cât și "exact o dată" (exactly once) în anumite condiții.

# 1. Aspecte funcționale și de arhitectură

## 1.1 Logica de rutare

RabbitMQ utilizează un model bazat pe mesaje și cozi, unde producătorii trimit mesaje către un schimb (exchange), iar schimbul decide la care coadă să trimită mesajele bazat pe reguli de rutare. Suportă diferite tipuri de schimburi: direct, topic, fanout și headers, oferind flexibilitate în rutarea mesajelor.

Kafka utilizează un model de jurnal distribuit, unde mesajele sunt scrise într-un jurnal și consumatorii citesc mesajele în ordinea în care au fost scrise. Nu are un mecanism avansat de rutare a mesajelor ca RabbitMQ, dar compensează prin simplitatea și scalabilitatea sa.

## 1.2 Garantarea livrării

RabbitMQ oferă diferite niveluri de garantare a livrării mesajelor, inclusiv livrare "at most once", "at least once" și "exactly once". Utilizează confirmările de mesaj și mecanisme de retry pentru a asigura livrarea.

Kafka este conceput pentru a asigura livrarea "at least once" prin replicarea mesajelor pe mai multe brokeri. Suportă și livrarea "exactly once" în anumite condiții, utilizând tranzacții și mecanisme care previn procesarea repetată a acelorași mesaje.

## 1.3 Ordonarea mesajelor

RabbitMQ asigură ordonarea mesajelor în cadrul unei cozi. Aceasta înseamnă că mesajele sunt livrate și procesate de consumatori în exact aceeași ordine în care au fost trimise la coadă. Dacă o aplicație necesită ca mesajele să fie strict ordonate, toate mesajele relevante trebuie să treacă prin aceeași coadă. Deși acest lucru asigură integritatea ordinii mesajelor, poate duce la un punct unic de blocaj, limitând astfel scalabilitatea sistemului.

Pentru a gestiona volum mare de mesaje sau pentru a crește performanța, RabbitMQ poate utiliza mai multe cozi și schimburi (exchanges), dar ordonarea strictă a mesajelor poate deveni o provocare în acest context. Utilizarea de cozi multiple poate necesita logică suplimentară pentru a menține ordinea mesajelor, dacă aceasta este o cerință critică.

Kafka asigură ordonarea mesajelor la nivel de partiție. În Kafka, un topic este împărțit în mai multe partiții, iar fiecare partiție păstrează ordinea mesajelor. Astfel, ordonarea mesajelor este garantată în cadrul unei partiții. Consumatorii de mesaje din pot procesa mesajele în ordinea în care au fost scrise în partiție, asigurând consistența și predictibilitatea procesării.

Această abordare permite Kafka să scaleze mult mai eficient, deoarece partițiile pot fi distribuite pe mai multe noduri de broker, fiecare nod gestionând o parte din sarcină. Aceasta permite atât scalabilitatea orizontală, cât și păstrarea ordinii mesajelor.

Pentru cazurile în care ordonarea strictă a mesajelor este necesară la nivel de topic întreg (nu doar la nivel de partiție), Kafka necesită ca toate mesajele să fie trimise la aceeași partiție, ceea ce ar putea limita din nou scalabilitatea, dar în majoritatea scenariilor, partiționarea oferă un echilibru optim între ordonare și performanță.

## 1.4 Transmitere multicast

RabbitMQ suportă transmisia multicast prin intermediul schimburilor de tip fanout, care trimit o copie a fiecărui mesaj la toate cozile legate de acel schimb.

Kafka nu suportă nativ multicast, dar poate distribui mesaje către mai mulți consumatori prin grupuri de consumatori, unde fiecare consumator dintr-un grup primește un subset al mesajelor.

## 1.5 Modul de stocare

În cazul RabbitMQ, mesajele pot fi stocate în memorie sau pe disc, iar cozile durabile asigură păstrarea mesajelor chiar și în cazul unei căderi a serverului. După ce mesajele sunt consumate de către consumatori, acestea sunt șterse automat din cozi, eliberând astfel spațiul de stocare și prevenind acumularea inutilă a datelor vechi. Aceasta caracteristică este utilă pentru menținerea eficienței operaționale și pentru managementul resurselor de stocare. Mesajele nu mai pot fi recuperate odată ce au fost consumate.

Kafka stochează toate mesajele în memorie într-un jurnal de tip append-only, ceea ce permite o recuperare rapidă și eficientă. Jurnalul de tip append-only asigură că scrierea pe disc este secvențială, ceea ce este mult mai rapid și eficient decât scrierea aleatorie, reducând latența și crescând debitul de date. Mesajele în Kafka nu sunt șterse imediat după consumare. În schimb, ele sunt păstrate pentru o perioadă de timp, după care sunt șterse automat. Aceasta permite reluarea consumului de mesaje de către noi consumatori sau recuperarea mesajelor în cazul unor erori, oferind o mare flexibilitate și reziliență.

Politicile de retenție în Kafka pot fi configurate pe baza timpului sau pe baza dimensiunii. Aceasta permite ajustarea stocării în funcție de nevoile specifice ale aplicației și de resursele disponibile.

## 1.6 Suport tranzacțional

RabbitMQ oferă suport pentru tranzacții, permițând gruparea mai multor operațiuni de mesagerie într-o singură tranzacție atomică. Aceasta înseamnă că un set de mesaje poate fi trimis și confirmat ca un întreg, asigurând că fie toate mesajele din tranzacție sunt livrate și procesate, fie niciunul dintre ele nu este. Această caracteristică este esențială pentru aplicațiile care necesită consistență și integritate a datelor.

Kafka oferă suport pentru tranzacții complexe, asigurând livrarea "exactly once" și menținerea consistenței datelor între topic-uri multiple. Aceasta este o caracteristică care permite producătorilor să trimită mesaje într-un mod tranzacțional, astfel încât fiecare mesaj este livrat o singură dată și în aceeași ordine în care a fost trimis, chiar și în cazul unor erori de rețea sau de sistem.

Tranzacțiile în Kafka sunt utile pentru scenarii complexe de procesare a datelor, cum ar fi integrarea mai multor fluxuri de date sau actualizarea coerentă a mai multor sisteme downstream. De exemplu, într-un sistem de procesare a plăților, Kafka poate asigura că actualizările conturilor și înregistrările jurnalelor de tranzacții sunt sincronizate și consistente.

Kafka permite consumatorilor să citească mesaje în mod tranzacțional, asigurând că niciun mesaj nu este pierdut sau procesat de mai multe ori. Acest lucru este esențial pentru aplicațiile critice care necesită integritate ridicată a datelor și consistență.

## 1.7 Gradul de securizare

RabbitMQ suportă autentificare și autorizare, criptare TLS și ACL-uri (Access Control Lists) pentru a controla accesul la resurse.

Kafka oferă mecanisme similare de securitate, inclusiv autentificare prin SASL, criptare TLS și controale ACL.

# 2. Aspecte de calitate a serviciului oferit

## 2.1 Disponibilitate

Rabbit oferă suport pentru clustering și mirror queues pentru a asigura disponibilitatea și failover-ul automat al mesajelor. Clusterele RabbitMQ permit distribuirea sarcinii între mai multe noduri, iar mirror queues replică mesajele pe mai multe noduri pentru a preveni pierderea datelor în cazul unei căderi.

Kafka este proiectat pentru disponibilitate ridicată, utilizând replicarea datelor pe mai multe brokeri și algoritmi de consens pentru a asigura toleranța la erori. . În plus, Kafka folosește algoritmul de consens ZooKeeper pentru a coordona și gestiona starea clusterului, asigurând o carateristică operațională ridicată

## 2.2 Performanță

RabbitMQ este performant pentru sarcini de mesagerie tradiționale, dar performanța poate scădea la volum mare de mesaje datorită modelului său de cozi și schimburi.

Kafka este foarte performant pentru ingestia și procesarea unui volum mare de date datorită arhitecturii sale de jurnal distribuit și a mecanismelor eficiente de replicare și partiționare.

## 2.3 Scalabilitate dinamică

Scalabilitatea în RabbitMq poate fi o provocare datorită limitărilor modelului de cozi și schimburi. Clustering-ul ajută, dar configurarea și gestionarea devin complexe la scară mare.

Kafka este extrem de scalabil, permițând adăugarea și eliminarea de brokeri și partiții dinamic, fără întreruperea serviciului.

## 2.4 Latența și debitul

Prin folosirea RabbitMQ latența poate varia în funcție de configurare și de volumul de mesaje, dar este de obicei scăzută pentru sarcini de mesagerie tradiționale. Debit relativ ridicat, dar poate fi limitat de arhitectura sa.

Kafka oferă latență predictibilă și debit foarte ridicat, fiind proiectat pentru ingestia și procesarea unui volum mare de date în timp real.

# 3. Caracteristici distinctive a celor două tehnologii

RabbitMQ:

* Flexibilitate mare în rutarea mesajelor datorită tipurilor variate de schimburi.
* Suport robust pentru tranzacții și diferite niveluri de garantare a livrării.
* Implementare relativ simplă pentru scenarii de mesagerie tradițională.

Kafka:

* Scalabilitate și performanță excepționale pentru volume mari de date.
* Arhitectură de jurnal distribuit care asigură ordonarea și persistența mesajelor.
* Potrivit pentru aplicații de streaming de date și analytics în timp real.

# 4. Implementare proprie a unei aplicații de chat folosind RabbitMQ

## 4.1 Descrierea aplicație

Este o aplicație web clasică de tip Chat, care permite comunicarea (1-1, 1-m, m-n) între diverși clienți care folosesc platforma. A fost realizată folosind următoarele tehnologii: SpringBoot pentru backend, React pentru FrontEnd, PostgresSQL pentru stocarea datelor și RabbitMq pentru a gestiona mesajele. Fiecare utilizator poate sa comunice în privat cu celălalt utilizator sau poate comunica pe canalul comun de broadcast, care funcționează ca un grup foarte mare unde toată lumea poate vorbi.

Aplicația prezintă următoarele funcționalități:

* Înregistrarea de clienți multipli
* Login/Logout pentru fiecare utilizator
* Comunicare privată și publică între utilizatori
* Stocarea mesajelor într-o bază de date persistentă
* Folosirea de cozi separate pentru fiecare tip de mesaj, în RabbitMQ

În RabbitMQ am implementat un exchange și două cozi (queues). Pentru fiecare queue am ales un binding key separat. Prima coadă este folosită pentru a gestiona mesajele din canalul de broadcast. Cea de-a doua coadă poate este folosită strict pentru mesajele private, dintre oricare 2 utilizatori.

A diagram of exchange

Description automatically generated

## 4.2 Implementare

Pentru implementarea aplicației web de tip Chat, am folosit o arhitectură bazată pe mai multe tehnologii pentru a asigura funcționalitatea și performanța dorită. Platforma permite comunicarea atât privată, cât și publică între utilizatori, gestionând eficient mesajele prin intermediul RabbitMQ. Iată cum am implementat fiecare componentă.

În backend am utilizat SpringBoot pentru a construi backend-ul aplicației. SpringBoot oferă un framework robust și scalabil, care facilitează dezvoltarea rapidă și gestionarea configurărilor complexe. Tot aici se gestionează înregistrarea și autentificarea utilizatorilor, comunicarea cu baza de date și interacțiunile cu RabbitMQ (producător și consumator) pentru trimiterea, primirea și stocarea mesajelor. Backend-ul respectă principiul REST și permite manipularea datelor prin call-uri de API.

Pentru partea de fronted am folosit React pentru a dezvolta interfața utilizatorului. Aici se gestionează afișarea mesajelor, formularele de login și înregistrare, precum și interacțiunile în timp real între utilizatori. De aici se apelează backendu-ul, mai exact API-ul creat la mai sus. Mesajele sunt afișate în timp real folosind tehnologia WebSockets.

RabbitMQ este utilizat pentru a gestiona fluxul de mesaje între utilizatori. Am implementat un exchange și două cozi (queues) pentru a separa diferitele tipuri de mesaje. Prima coadă este dedicată mesajelor din canalul de broadcast, unde toți utilizatorii pot comunica într-un mediu comun. Aceasta permite trimiterea mesajelor către toți utilizatorii conectați.

A doua coadă este utilizată pentru mesajele private între doi utilizatori. Acest mecanism asigură că mesajele private sunt livrate rapid și în siguranță doar destinatarului prevăzut.

Fiecare coadă utilizează un binding key separat pentru a direcționa mesajele corespunzător. Binding keys permit configurarea flexibilă a rutării mesajelor în funcție de tipul și destinația acestora.

Mesajele consumate nu se mai pot regăsi în RabbitMQ, dar se pot regăsi în baza de date din backend. Astfel, un utilizator se poate deconecta, iar la reconectare se poate regăsi mesajele mai vechi.

A diagram of a software application

Description automatically generated

-diagrama de deployment-

A diagram of a software system

Description automatically generated

-arhitectura conceptuala-

# Concluzie

Atât RabbitMQ, cât și Kafka sunt tehnologii puternice pentru mesagerie, fiecare având propriile avantaje în funcție de cerințele specifice ale aplicațiilor. RabbitMQ este ideal pentru scenarii ce necesită flexibilitate în rutarea mesajelor și suport tranzacțional robust. Kafka, pe de altă parte, excelează în performanță și scalabilitate pentru volume mari de date și aplicații de streaming în timp real. Alegerea între cele două depinde de nevoile specifice ale proiectului și de cerințele de performanță, scalabilitate și complexitate arhitecturală.