KONKURENTNÉ PROGRAMOVANIE

Cvičenie 14 : Úvod do message brokerov a RabbitMQ

Komunikácia v distribuovanom systéme

- Oproti paralelnému systému nemáme zdieľanú pamäť
  - Vieme mať zdieľané úložisko (klasická databáza) – ďalší uzol distribuovaného systému
  - Vieme mať zdieľané úložisko, ktoré simuluje zdieľanú pamäť (key-value databázy → viď predmet NoSQL databázy)
- Všetky dáta si uzly posielajú ako **správy** cez sieťové rozhrania
  - Hoc aj v rámci jedného počítača

Distribuované systémy

- Uzly
  - Počítače, procesy
- Možné zlyhania
  - Uzly – vypnutie, reštart, zlyhanie, spomalenie, preťazenie
  - Sieť – strata spojenia, kolísavá rýchlosť

Výzvy distribuovaných systémov

- Aká architektúra?
- Akým protokolom komunikovať?
- Ako sa uzly majú vyhľadať, napojiť, skoordinovať?
- Konzistencia a replikácia
- Tolerancia k zlyhaniam
- Bezpečnosť

Message Broker = Poštový úrad

- Samostatná služba umožňujúca asynchrónnu komunikáciu
- Úlohy message brokerov
  - Smerovanie správ
    - Kto má dostať, ktoré správy
  - Topológia
    - Ako uzly môžu vzájomne komunikovať
  - Transformácia správ medzi formátmi
  - Perzistencia – uchovanie správ, ktoré ešte majú byť prijaté, aj v prípade že by došlo k reštartu
  - Potvrdenie prijatia/spracovania správ
  - Spájanie/agregácie správ

Popolárne message brokery

- Kafka
  - Výkonná platforma pre veľké prúdy dát
- AMQP/RabbitMQ
  - Univerzálne riešenia pre klasické topológie
- MQTT
  - Špeciálne pre IoT
  - Odľahčený klient pre maličké zariadenia
- JMS – Java Message Service
  - Historické enterprise systémy

Spoločné vlastnosti

- Uzly si posielajú správy
- Komunikácia je asychrónna
  - Nečaká sa na odpoveď
- Odosielatelia a príjemcovia spolu nekomunikujú priamo
- Broker spravuje rady správ (queue)
  - Producent(i) zapisujú na koniec
  - Konzument(i) čítajú zo začiatku
- Môže sa riešiť spoľahlivosť doručenia
- Perzistencia nedoručených správ

Architektúra 1: point to point

- Producenti a konzumenti nemusia byť pripojení súčasne
- Viacero producentov zapisuje do radu
- Správu prijme **práve jeden konzument**
  - Ak je viac konzumentov počkajú si na ďalšie správy
  - Delenie práce
- Potvrdzovanie
  - Ak nie: po skonzumovaní sa správa vymaže z radu
  - Ak áno: konzument potvrdzuje (**ack**) spracovanie správy
    - Správa sa zmaže až po potvrdení
    - Ak konzument pri spracovaní spadne, po čase môže byť správa zaslaná inému konzumentovi

Architektúra 2: publish - subscribe

- Nástenka (zoznam) správ, ktoré boli prijaté
- Producenti vytvoria topic a píšu doňho
- Konzumenti sa prihlásia na odber (subscribe) na odber správ z topicu
- Každý konzument dostane kópiu doručenej správy
  - Po tom, ako každý konzument správu prečíta, sa voliteľne môže správa zmazať
- Konzument si môže vyberať či chce čítať historické správy, alebo len nové

Architektúry komplexné

- Rad je základ každého brokera
- Jednotliví brokeri poskytujú vlastné varianty
  - Ako koordinovať prácu konzumentov
  - Kedy mazať staršie správy
- RabbitMQ: pouličné schánky → poštové pobočky → bytové schránky
- Kafka: topic → partícia → črieda konzumentov

Formáty správ

- Binárne protokoly
- Textové protokoly
  - XML, JSON, STOMP

Spoľahlivosť doručenia

- At most once
  - Správy sa môžu stratiť (neskonzumovať)
- At least once
  - Konzument musí potvrdzovať správy
  - Ak klient nepotvrdí, pošleme správu znova
  - Klient sa musí vedieť vysporiadať s duplicitami
- Exactly once
  - Veľmi ťažké

RabbitMQ

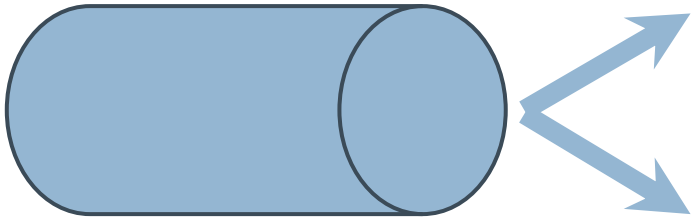
- Opensource
- Škálovateľný
- Implementácie klientov pre mnohé jazyky
- Producenti a konzumenti môžu byť naprogramovaní v iných jazykoch
- My použijeme knižnicu pre Spring boot
  - Závislosť **spring-boot-starter-amqp**

RabbitMQ pre Docker

- docker run -d --hostname rabbit --name rabbitmq -p 15672:15672 -p 5672:5672 rabbitmq:4-management
- Manažovací portál:
  - http://localhost:15672/
  - Login guest, heslo guest
    - Prípadne sa dajú vo vyššie napísanom príkaze dodať ešte
      -e RABBITMQ_DEFAULT_USER=user
      -e RABBITMQ_DEFAULT_PASS=password

Prijímanie správ

- Konzument číta správy typicky z jednej Queue
- Use case1: **Rozdeľovanie práce**
  - Na jednu queue môže byť napojených viacero konzumentov
  - Queue posiela správy čakajúcim konzumentom postupne round-robin štýlom



Potvrdzovanie správ

- Konzument po prijatí správy ju má spracovať
- Ak ju **úspešne** spracuje pošle **potvrdenie (ACK)**
  - Správa sa zmaže z radu
- Ak sa ju nepodarí spracovať alebo konzument odvisne
  - Správa sa vracia na začiatok radu a je pripravená na opätovné spracovanie
- Za normálnych okolností by konzument mal správu buď skonzumovať (ACK) alebo ju odmietnuť (NACK)
  - V oboch prípadoch sa správa zmaže z radu

Potvrdzovanie správ - Spring

- Ak metóda @RabbitListener
  - nevyhodí výnimku, Spring správu **potvrdí** po skončení metódy
  - vyhodí výnimku AmqpRejectAndDontRequeueException, spring správu **zamietne** a z radu sa vymaže
  - vyhodí inú výnimku, správa **sa vracia do radu** na opätovné spracovanie

Posielanie od producenta - vnútorný mechanizmus

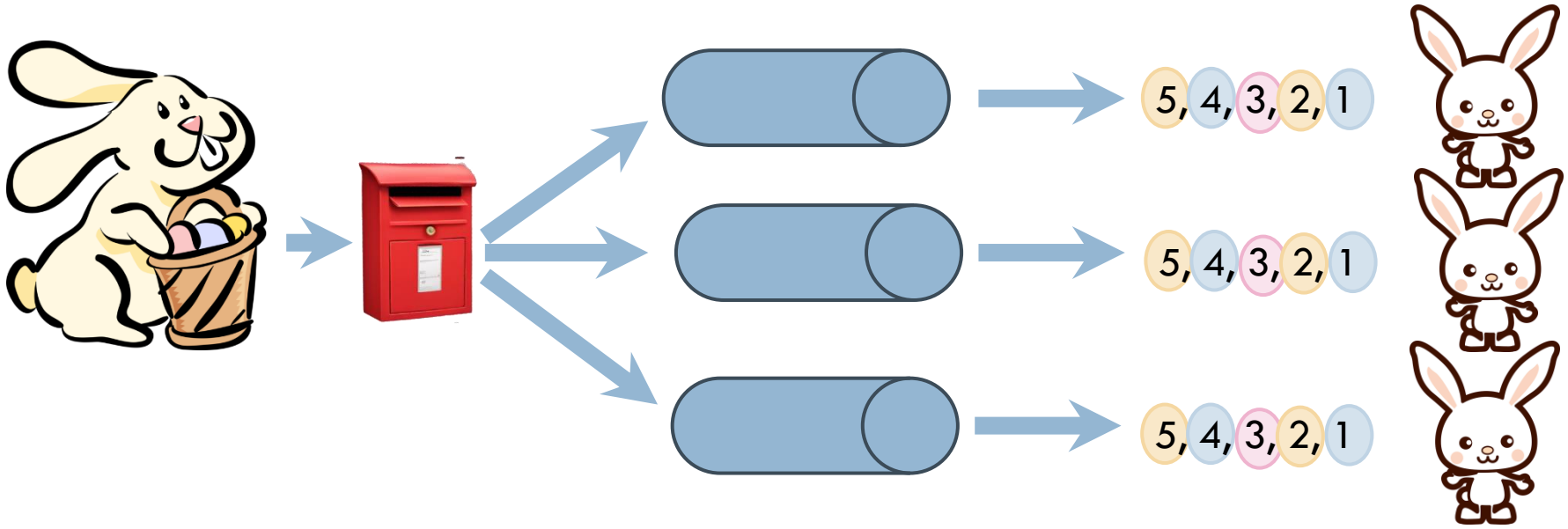
- Producent posiela správu do Exchange (schánka správ)
  - Obvykle jeden Exchange zodpovedá jednému typu dát (kapusta, nameraná hodnota, udalosť prihlásenia)
- Binding
  - Napojený na Exchange
  - Odosiela správy do príslušného radu/radov s ktorými je prepojený
- Z radu už správy odchádzajú ku konzumentom

Exchange

- Schránka správ = rozposielač správ/prvkov do radov
- Typy Exchange-ov:
  - Fanout – broadcast všetkým napojeným radom
  - Direct – zaslanie iba jednému radu na základe názvu radu pribalenému k správe
  - Topic – univerzálne riešenie podľa routovacieho kľúča

Binding pre Exchange typu Fanout

- Use case 2: **broadcastovanie správ**
- Fanout Exchange – vieme broadcastovať správy do viacerých radov
  - rozkopíruje všetky prijaté správy do všetkých radov, s ktorými je exchange prepojený cez binding
- Každý konzument si vytvorí svoj rad a binding na Exchange



Binding pre Exchange typu Fanout

- Kombinácia broadcastu a distribúcie práce:
  - Každý konzument čaká v niektorom z radov.