Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 4.9
DOI: https://doi.org/10.24867/33BE18Ninkovic

ORKES CONDUCTOR – POREĐENJE PERFORMANSI SA APACHE KAFKA

ORKES CONDUCTOR – PERFORMANCE COMPARISON WITH APACHE KAFKA

Jelena Ninković, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *U ovom radu opisani su Orkes Conductor kao primer orkestracije događajima i Apache Kafka kao primer koreografije. Urađena je analiza i poređenje performansi ova dva alata. Rad uključuje i opis implementacije oba alata, napisane u programskom jeziku Java, kao i poređenje redova koji predstavljaju osnovnu strukturu podataka na koju se alati oslanjaju.*

Ključne reči: *Conductor, zadatak, razmena poruka, Apache Kafka, red*

Abstract – *This paper describes Orkes Conductor as an example of event orchestration and Apache Kafka as an example of choreography. A performance analysis and comparison of two tools is provided. This paper also includes description of implementation of both tools, written in Java programming language, as well as a comparison of queues, which represent the fundamental data structure these tools rely on.*

Keywords: *Conductor, task, messaging, Apache Kafka, queue*

1. UVOD

Mikroservisna arhitektura (MSA) predstavlja arhitekturalni dizajn šablon koji je uveden da reši probleme oko horizontalne skalabilnosti, dostupnosti, modularnosti i agilnosti arhitekture u tradicionalnim monolitnim sistemima. Aplikacija se razvija kao skup malih servisa [1], gde je svaki nezavisno razvijan, testiran, ažuriran, skaliran i deploy-ovan i komunicira sa ostatkom preko jednostavnih mehanizama, najčešće HTTP poziva. Servisi su tako definisani da svaki ima sopstvene entitete i bazu podataka, što čini da promene u bazi podataka jednog servisa nisi vidljive u bazi drugog servisa. Ukoliko dođe do rollback-a transakcije zbog greške u jednom od servisa povratak na pređašnje stanje nije moguć jer su u pitanju distribuirane transakcije.

Da bi se rešio problem uvodi se SAGA [2] šablon. U slučaju greške okida se sekvenca rollback događaja, od jednog servisa ka drugom, u obrnutom redosledu. SAGA šablon može biti implementiran korišćenjem koreografije događaja i orkestracionim tehnikama.

U slučaju koreografije događajima, svaki mikroservis radi zasebno i kada završi lokalnu transakciju emituje događaj, koji drugi servisi osluškuju sa ciljem da započnu svoje transakcije.

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Srđan Popov, red. prof.

Proces se nastavlja sve dok poslednji servis ne emituje nijedan događaj, što predstavlja kraj transakcije.

Kod orkestracije postoji centralni orkestrator, koji se ponaša kao „roditeljski“ servis, sluša sve događaje koje emituju lokalne transakcije mikroservisa i na osnovu događaja okida sledeću lokalnu transakciju u drugom mikroservisu ili servisima.

2. ORKES CONDUCTOR

Orkes Conductor [3] je radni okvir koji je razvijen povrh Netflix Conductor-a. Pored osnovnih funkcionalnosti koje je nudila platforma dodate su funkcionalnosti i poboljšanja i cilju lakšeg deploy-ovanja i upravljanja tokovima u produkcionom okruženju.

2.1. Osnovni pojmovi

Proces orkestriranja korišćenjem Conductor-a obuhvata korišćenje tri osnovna koncepta: zadatke, radnike i tokove.

Zadatak predstavlja jedinicu posla ili korak u toku, poput kreiranja HTTP poziva, slanja mejla, procesiranja podataka ili izvršavanja poslovne logike. Predstavlja osnovnu gradivnu jedinicu toka i dodatno može biti podeljen na operatore, sistemske zadatke ili radnike za sopstveni kod.

Radnici su kod zadužen za izvršavanje zadataka. Za sistemske zadatke i operatore je zadužen Conductor server, dok zadacima koje je definisao korisnik upravljaju aplikacije koje ih implementiraju. Nakon što radnik kontaktira server da primi i izvrši zakazane zadatke Conductor će proslediti ulazne parametre zadatka radniku i preuzeti izlazne podatke nakon završetka.

Tok se definiše kao kolekcija zadataka i operatora, koji specificiraju redosled i izvršenje definisanih zadataka. Ova orkestracija se dešava u hibridnom ekosistemu koji objedinjuje serverless funkcije, mikroservise i monolitske aplikacije. Kako je Conductor jezički agnostičan orkestracija može biti izvršena u bilo kom programskom jeziku. Workflow scheduler omogućava tokovima da budu pokrenuti po određenom rasporedu. Ovo daje mogućnost da tokovi budu konfigurisani da se pokreću u željenoj učestalosti i da se prilikom kreiranja rasporeda bira verzija toka.

2.2. AI zadaci

Orkes Conductor omogućava kreiranje aplikacija koje koriste generativne AI modele i vektorske baze podataka. Generativni AI je tip veštačke inteligencije sposoban za kreiranje novog „čovekolikog“ sadržaja na osnovu pre-treniranih modela koji su bili izloženi velikim količinama

sličnog sadržaja. Ljudska interakcija je i dalje potrebna da se modeli usmere šta treba da bude generisano - usmeravaju se slanjem promptova (tekstualnih instrukcija). Odgovor Gen-AI modela je isto tekstualan, i ovi modeli se nazivaju LLM. LLM su deep learning algoritmi obučeni na velikoj količini podataka. Mogu da obavljaju razne NLP zadatke, poput generisanja, prevođenja, chatbot-ova, AI asistenata i sl.

2.3. Alert zadaci

Alert zadaci su poseban vid zadataka koji imaju ulogu da šalju notifikacije ili upozorenja na osnovu određenih uslova ili događaja u sistemu.

2.4. Rukovalac događajima

Rukovalac događajima procesira dolazeće poruke i izvršava akcije na osnovu njihovih detalja. Okidač može biti okinut od strane narednih akcija – complete task, terminate workflow, update variables, fail task i start workflow.

2.5. Kontrola pristupa

Orkes Conductor nudi kontrolu pristupa baziranu na ulogama (RBAC) korisnicima Orkes platformi, kao i aplikacijama koje koriste Conductor API. RBAC obezbeđuje pristup metapodacima tokova, zadataka, tajni, promenljivama okruženja, integracijama, promptovima, korisničkim formama, rukovaocima događaja, rasporedima, webhook-ovima i domenima.

Korisnik predstavlja čoveka koji interaguje sa Conductor-om preko Orkes platforme, i autentifikovan je korišćenjem SSO provajdera ili mejla/lozinke. Svaki korisnik može imati jednu ili više dodeljenih uloga.

Grupa predstavlja set korisnika, i predstavlja brz način da se dodele permisije većem broju korisnika. Svaka grupa može biti povezana sa jednom ili više uloga. Takođe može imati dodeljen set permisija, koje obezbeđuju pristup određenim resursima. Kada je korisnik dodat u grupu automatski nasleđuje sve uloge i permisije grupe, a kada je uklonjen iz nje gubi sve uloge i permisije koje je nasledio iz nje.

Aplikacija predstavlja aplikaciju koja interaguje sa Conductor serverom putem API-ja ili SDK-ja. Aplikaciji mogu biti dodeljene permisije, koje će obezbediti pristup određenim Conductor resursima. Svaka aplikacija može imati jedan ili više ključ/tajna parova, koji se koriste za dobijanje pristupa.

Tag predstavlja par ključ/vrednost koji može biti pridružen metapodacima resursa. Služi kao prečica za deljenje permisija brojnih resursa ili korisnika.

Uloga predstavlja set opštih podrazumevanih permisija resursima. Može biti dodeljena korisniku, grupi ili aplikaciji. Ako je dodeljeno više uloga biće dodeljene permisije za svaku od njih.

Pored permisija baziranih na ulogama moguće je dodeliti i granularne permisije grupama ili aplikacijama. Granularne permisije pružaju dodatni pristup povrh korisnikovih ili aplikacijskih permisija baziranih na ulozi.

Domen se koristi da se dodeli pristup svim zadacima u određenom domenu. Koristan je za masovno dodeljivanje prava radnik aplikaciji da izvrši sve zadatke, bez da se mora dodati svaki pojedinačni zadatak i da se definiše njegov domen.

2.6. Rutiranje zadataka

Za svaki konfigurisani tip zadatka Conductor server će održavati red i distribuiraće zadatke svim radnicima konektovanim na server. Postoji i opcija gde isti zadatak može biti rutiran različitom setu radnika na osnovu koncepta zvanog Task to Domain. Ova vrednost se prosleđuje toku kada je pokrenut, i ako je prisutna znači da se zadaci rutiraju drugačije od podrazumevanog načina.

2.7. Nadgledanje redova zadataka

Red zadataka sadrži zadatke koji čekaju da se izvrše. Nadgledanje ovih redova obezbeđuje optimalni performans i efikasnost u procesiranju zadataka, identifikovanje potencijalnih problema i održavanje pouzdanosti sistema.

2.8. Metrika i skaliranje radnika

Skaliranje i podešavanje performansi radnika zavisi od sledećih metrika: broja zadataka na čekanju, propusnosti pojedinačnog radnika i ukupnog broja pokrenutih radnika.

2.9. Rukovanje greškama

Conductor je napravljen da nudi najmanje jednu garanciju isporuke – sve poruke mogu da se čuvaju, trajne su i biće isporučene radnicima najmanje jednom. Ovakav model osigurava dve stvari: kada tok započne biće kompletiran ako su svi zadaci kompletirani i ako izvršavanje toka bude neuspešno zbog ponovnih pokušaja i sl., poruke će biti isporučene sledećem čvoru koji je živ i responsivan.

Timeout se može dogoditi ako nema radnika za zadati tip zadatka, radnik primi poruku ali prekine svoj rad pre nego što završi zadatak i zadatak nikada ne pređe u complete status ili je radnik završio procesiranje, ali ne može da komunicira sa Conductor serverom usled greške u mreži, ili srušenog Conductor servera.

3. APACHE KAFKA

Apache Kafka [4] je distribuirana platforma za striming podataka koja može da objavljuje, prima, čuva i procesira strimove u realnom vremenu. Dizajnirana je da rukuje velikim brojem real-time informacija i rutira ih mnogostrukim consumer-ima.

Osnovna jedinica unutar Kafke je poruka – niz bajtova. Poruke mogu imati opcione metapodatke – ključ. Ključ je takođe niz bajtova i koristi se kada se poruke upisuju u particije. Zbog bolje efikasnosti poruke se u Kafku upisuju kao batch – kolekcija poruka, gde se poruke kategorišu u teme, koje su dodatno razbijene na particije. Poruke se upisuju na kraj teme i čitaju sa početka. Ukoliko tema ima više particija ne može biti garantovano da će u celoj temi biti ispoštovan redosled kako su poruke stizale, samo u pojedinačnoj particiji. Particije su način na koji Kafka obezbeđuje skalabilnost, jer svaka particija

može biti na drugom serveru, što znači da jedna tema može biti horizontalno skalirana na više servera.

3.1. Producer

Producer-i [5] su aplikacije koje kreiraju poruke i šalju ih Kafka brokeru za dalje konzumiranje. Producer ne upisuje poruke u particije, već kreira zahteve za poruke i šalje ih vođi brokera. Kafka producer-i mogu biti sinhroni i asinhroni. Sinhroni producer-i nakon slanja poruke čekaju potvrdu od brokera, dok asinhroni nastavljaju sa daljim radom bez čekanja. Prednost sinhronih producer-a je pouzdanost, međutim čekanje na potvrdu može dovesti do zastoja u sistemu i ograničavanja broja poruka koje mogu biti poslate odjednom. Kod asinhronih producer-a ovi problemi su rešeni, ali su dosta komplikovaniji za implementiranje, naročito rukovanje greškama i ako nisu pažljivo implementirani može doći do gubitka podataka.

3.2. Consumer

Consumer-i [5] su aplikacije koje konzumiraju poruke. Consumer se pretplaćuje na jednu ili više tema i čita poruke u redosledu kojim su stigle. Consumer vodi računa koje poruke je već pročitao tako što vodi računa o offset-u poruka. Offset je metapodatak, integer brojač koji se stalno uvećava, na koji Kafka daje svaku poruku kako je objavljena. Čuvanjem offset-a poslednje konzumirane poruke za svaku particiju, u Zookeeper-u ili u samoj Kafki, consumer može biti zaustavljen i ponovo pokrenut bez gubljenja podataka. Consumer-i se nalaze unutar consumer grupe – jedan ili više consumer-a koji rade zajedno i konzumiraju temu. Grupa osigurava da je svaka particija konzumirana od samo jednog člana. Mapiranje consumer-a i particije se naziva vlasništvo particije od strane consumer-a. Na ovaj način consumer-i mogu biti horizontalno skalirani da konzumiraju teme sa velikim brojem poruka. Ukoliko jedan consumer ima grešku ostali članovi grupe će rebalansirati particije tako da preuzmu posao neuspešnog člana.

3.3. Brokeri i klasteri

Jedan Kafka server se naziva broker. Broker prima poruke od producer-a, dodeljuje im offset, i commit-uje poruke na disku. Takođe servisira consumer-e, odgovarajući na fetch zahteve porukama koje je prethodno commit-ovao. Kafka brokeri funkcionišu kao deo klastera. Unutar klastera, jedan broker služi kao kontroler, izabran automatski iz skupa živih članova klastera. Kontroler je zadužen za administrativne operacije, uključujući dodeljivanje particija brokerima i nadgledanje brokera u slučaju neuspeha. Particija je u vlasništvu samo jednog brokera unutar klastera, i taj broker se zove vođa particije. Međutim, particija može biti dodeljena više brokera, što će rezultovati repliciranjem particije.

Karakteristika Kafke je period zadržavanja. Brokeri su konfigurisani sa podrazumevanim periodom ili dok tema ne dostigne određenu veličinu. Pojedinačne teme mogu biti definisane sa sopstvenim periodom zadržavanja.

3.3. Zookeeper

Zookeeper je komponenta Kafke koja služi kao koordinator i služi za biranje kontrolera, čuvanje statusa brokera, čuvanje metapodataka tema, održavanje informacija o klijentskim normama i ACL tema.

4. POREĐENJE ORKES CONDUCTOR-A I APACHE KAFKE

Za poređenje orkestracije i koreografije uzet je primer onlajn prodavnice. Kreiran je Java projekat u kome su definisani sledeći mikroservisi: ordering servis, inventory servis, payment servis, notification servis i shipping servis.

4.1. Implementacija Conductor toka

Mikroservis ordering preuzima ulogu orkestratora i koordiniše komunikaciju između preostalih servisa. Tok je definisan u JSON formatu i dodat u Orkes server. U okviru ordering servisa definisan je REST endpoint koji pokreće tok korišćenjem WorkflowClient-a definisanog u zavisnosti io.orkes.conductor:orkes-conductor-client. Prilikom pokretanja toka zadaje se ime toka koji se pokreće, kao i imena i vrednosti ulaznih promenljivih.

Prilikom definisanja radnika potrebno ga je anotirati sa @WorkerTask. Anotacija od ulaznih parametara ima ime zadatka, broj niti koje se koriste za izvršavanje zadatka, kao i interval koliko često radnik šalje upite serveru.

4.2. Implementacija Kafke

Koristi se event-driven komunikacija. Kafka server (kafka i zookeeper) je podignut na portu 9092 u Doker kontejneru i korišćenjem Java biblioteke u mikrosrevisima se kreiraju KafkaTemplate i ConcurrentKafkaListenerContainerFactory (samo u servisima gde postoji consumer) na osnovu konfiguracije definisane u application.yml fajlu. U okviru ordering servisa definisan je endpoint koji pokreće slanje poruka – ovaj servis jedini nema consumer-a, budući da je servis od kojeg počinje slanje.

Prilikom definisanja consumer-a potrebno ga je anotirati sa @KafkaListener. Anotacija od ulaznih parametara ima identifikator consumer-a, temu koji osluškuje, identifikator grupe – u ovom slučaju servis u kojem se nalazi, kontejner fabrike – definisano Java kodom prilikom konfiguracije Kafke i rukovalac u slučaju greške.

4.3. Poređenje primena struktura podataka tipa red

Red kao strukturu podataka koriste i Conductor i Kafka. Kod Conductor-a zadaci koji čekaju da se izvrše se nalaze u redu, a kod Kafke se poruke smeštaju u redove.

Conductor koristi FIFO (First-in First-out) strukturu, realizovanu preko Redis liste kao skladišta podataka.

Kafka koristi log-baziranu distribuiranu append-only strukturu, što znači da su poruke redosledno upisane po particiji, ali redosled među različitim particijama nije zagarantovan i višestruki consumer-i mogu čitati poruku nezavisno (poruka ne nestaje nakon čitanja).

Tabela 1. *Poređenje Orkes Queues i Kafka redova*

	Orkes Queues	Kafka
Tip reda	FIFO lista	Commit log/append-only
Redosled	Globalni FIFO	Definisan particijom
Čuvanje poruka	Da	Da
Višestruku consumer-i	Ne	Da
Lokacija skladišta	U memoriji (Redis)	Disk
Performanse	Odlično za real-time zadatke	Odlično za bulk striming
Ponovno čitanje poruka	Ne	Da

4.4. Komparativna analiza performansi

Uzevši u obzir da performanse sistema mogu zavisiti od različitih parametara – poput konfiguracije, hardverskih resursa i memorije, teško ih je kvantifikovati.

Za potrebe ovog rada, projekat čita podatke o porudžbinama iz pet datoteka, sa različitim brojem porudžbina.

Tabela 2. *Rezultati izvršavanja*

Broj poruka	500	1000	2000	5000	10000
Conductor	10s	15s	26s	49s	95s
Kafka	0.41s	0.31s	0.43s	1.32s	7.24s

Razlika prilikom obrade malog broja podataka nije velika, ali može se videti da Kafka ima bolje performanse, i razlika u performansama se povećava sa brojem podataka.

Kako servisi za shipping i notification ne zavise jedan od drugog definisani su u paraleli. Ovaj paralelizam kod Conductora-a zahteva i dodatne FORK i JOIN zadatke, gde prilikom izvršavanja najviše vremena odlazi na JOIN zadatak. U verziji gde je Conductor tok definisan kao sekvencijalni (izvršavaju se shipping pa notification zadatak) dolazi do ubrzanja - u slučaju sa 10000 podataka vreme izvršavanja je palo sa 95s na 76s. Razlozi zašto je sekvencijalni tok ispao brži od paralelnog su da je paralelizam logički, a ne fizički i overhead orkestracije – svaki zadatak mora biti evidentiran u bazi, status zadatka se upisuje u skladište metapodataka, radnik ga mora preuzeti i vratiti rezultat, što znači da kod jednostavnih tokova overhead postaje veći nego korist paralelizacije.

Na primeru implementacije Kafke, vidi se da Kafka ne čeka da poruke i od notification i shipping servisa budu primljene da bi se smatralo da je proces uspešno završen. Za dobijanje informacija o tome uvodi se još jedan servis – delivery servis, koji čeka uspešnu obradu obe poruke i nakon što su obrađene, razmena poruka će biti uspešna. Sa dodatnim servisom dolazi do značajnog usporavanja obrade poruka korišćenjem Kafke, gde je za 10000 poruka sada potrebno 29.9s.

5. ZAKLJUČAK

Sam Conductor ne obrađuje podatke direktno, već deluje kao centralni koordinator koji ih delegira radnicima, koji zatim vraćaju rezultate po završetku. Ova arhitektura omogućava sistemu da ostane slabo spregnut, uz potpunu vidljivost i kontrolu nad dugotrajnim i složenim procesima. Za razliku od Kafke koja je optimizovana za brzo i pouzdano prosleđivanje poruka između producer-a i consumer-a, Conductor je usmeren na orkestraciju tokova i upravljanje stanjem kroz kompletan životni ciklus procesa.

U situacijama gde je potrebna jednostavna razmena poruka ili linearno procesiranje, Kafka je adekvatniji izbor. Međutim, u slučajevima sa kompleksnijom logikom, gde je neophodno izvršavanje u paraleli, ponovni pokušaji, rukovanje greškama, uslovno grananje ili su zadaci vezani za specifične korisnike Conductor pruža prednost u pogledu deklarativnog modeliranja, bolje preglednosti, naprednog upravljanja greškama i načina upravljanjem zavisnostima između zadataka – redosled izvršavanja zadataka zavisi samo od definicije toka, a ne od implementacije u mikroservisima, što omogućava dinamično ažuriranje poslovne logike, bez modifikacije pojedinačnih komponenti.

Sami Conductor i Kafka nisu međusobno isključivi – Kafka može funkcionisati kao centralna infrastruktura za rukovanje događajima, dok Conductor može da služi kao „mozak“ procesa, koji orkestrira kako će se ti događaji obrađivati, transformisati i povezivati u smislene rezultate. Ovakav hibridni pristup omogućava korišćenje prednosti obe platforme – Kafke za stabilno prosleđivanje poruka, a Conductor za strukturiranu orkestraciju tokova poslovanja, čime se postižu sistemi koji su istovremeno skalabilni i inteligentni.

6. LITERATURA

[1] N. Alshuqayran, N. Ali and R. Evans, “A Systematic Mapping Study in Microservice Architecture”, Proc. of the 9th International Conference on Service Oriented Computing and Applications, IEEE, 2016.

[2] R. H. Campbell and P. G. Richards, “SAGA: A system to automate the management of software roduction”, Proceedings of the May 4-7 1981. National Computer Conference on AFIPS, pp. 231-234, 1981.

[3] https://orkes.io/content (pristupljeno u septembru 2024.)

[4] https://en.wikipedia.org/wiki/Apache_Kafka (pristupljeno u novembru 2024.)

[5] N. Gard, “Apache Kafka”, PACKT Publishing, 2013.

Kratka biografija:

Jelena Ninković je rođena u Šapcu 1993. godine. Upisala je fakultet 2013. godine, smer Elektrotehnika i računarstvo. Diplomirala je 2019. godine sa temom „Implementacija korisničkog interfejsa podsistema bankarskog poslovanja korišćenjem AngularJS razvojnog okvira“.