|  |  |
| --- | --- |
| Grb PMF-a | Univerzitet u Novom Sadu, Prirodno – matematički fakultet  DEPARTMAN ZA MATEMATIKU I INFORMATIKU  Trg Dositeja Obradovića 4, 21000 Novi Sad |

**Komunikacija u distribuiranim sistemima orijentisana porukama**

Master rad

dr Danijela Boberić Krstićev Robert Sabo

Novi Sad, 2020

# Predgovor

Ovde ide predgovor

# Sadržaj

[Predgovor 2](#_Toc49896207)

[Uvod 5](#_Toc49896208)

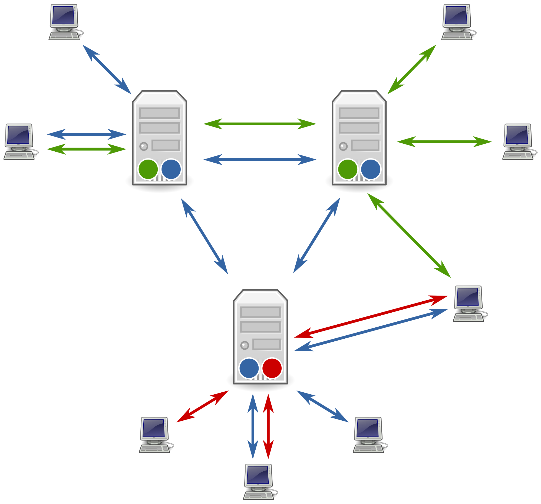
[Ovde pisemo sve I svasta **Error! Bookmark not defined.**](#_Toc49896209)

[Zaključak 39](#_Toc49896210)

Poglavlje 1

# Uvod

Svedoci smo svakodnevnih inovacija. Iz dana u dan pojavljuju se nove ideje pa time i novi koraci u razvoju tehnologije. Upotreba pametnih uređaja eksponencijalno raste. Time, podataka je sve više i više. Rastom prikupljanja podataka raste i potreban potencijal da se podaci obrade, pretvore u korisne informacije te ne kraju u usvojivo znanje.



Slika 1.1 Ilustracija komunikacije

Istorijski razvoj procesa obrade podataka je dugačak i obiman za jedan uvod u rad. Značajno je nepomenuti da već godinama obrada podataka najčešće biva decentralizovana. Ako ne i fizička, odvojenost u arhitekturi je prisutna u različitim delovima celinama obrade. Povezanost celina jeste komunikacija. Postavlja se pitanje kakva komunikacija? Kakva povezanost komponenti? Kakve i koliko poruka se razmenjuje?

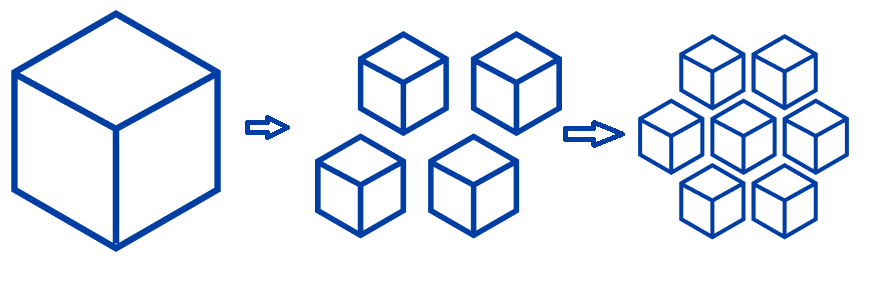
Konfuzijom pitanja vezanih za komunikaciju se nailazi na temu ovog rada: *Komunikacija u distribuiranim sistemima orijentisana porukama.* Iako sam naslov ne odgovara na sva pitanja, u radu će ipak biti razjašnjena većina njih.

## Komunikacija u distribuiranim sistemima orijentisana porukama

Radi razjašnjenja čime se to ovaj rad bavi najbolje je početi analizom samog naslova. Stoga će ovaj deo uvoda objasniti jedan po jedan pojam naslova u cilju razjašnjavanja šta

### Distribuirani sistemi

Porastom neophodnog potencijala za obradom podataka menjala se arhitektura obrađivača. Zašto? U početku je količina podataka potrebna za obradu bila značajno manjih razmera te je 1 računa samostalno uspeo da obradi sve podatke. Rast količine podataka za obradu se prati pojačavanjem hardverskih komponenti. Prekretnica se desila kada su ljudi ustanovili da je jedan jak računaj mnogo skuplji nego dva osrednja. Došlo se na ideju da se posao obrade rasporedi na 2 celine i da se svaka celina obrađuje na zasebnoj hardverskoj komponenti.



Slika 1.2 Razvoj distribuiranog sistema

Istim putem, koji je od jednog doveo do 2 hardvera, se došlo i do raspodele na *n* računara. Broj n se vremenom sve više povećava (kao što je i ilustrovano na slici 1.2). Zašto? Razlog je jednostavan: distribucija se pokazala kao dobra praksa.

Svakako se prvi cilj i prednost distribucija ogledala u iskorišćenosti i ceni hardvera. Međutim, to se pokazalo kao značajan napredak i u softveru. Naime, podeliti problem na prostije celine je u prirodi funkcionisana ljudskog mozga tokom razmišljanja. Činiti isto i u razvoju softvera predstavlja potpuno prirodan i očekivan korak. Čoveku je lakše da problem raščlani na više celine te da ih rešava pojedinačno, te se i sama softverska arhitektura sve više razvijala u manjim softverskim rešenjima koji zajedno daju sistem.

Distribuiran sistem predstavlja rešenje podeljeno na manje celine koji u korelaciji rešavaju zadati problem. Celine distribuiranog sistema mogu biti fizički i geografski odvojeni, dok je moguće, s druge strane, i da je podela samo logička. Kako god se sistem delio, manje celine predstavljaju logički udeo u rešenju problema. Jedna celina je zasebna i nezavisna i bavi se rešavanjem trivijalnog problema.

Osim uprošćavanja rešenja problema, podela na celine donosi i niz drugih prednosti. Međutim, ovaj rad se ne bavi time i distribuirani sistemi su samo oblast u kojoj se ogleda srž ovog rada. Istraživanje o mikroservis orijentisanoj arhitekturi, što je jedan vid distribucije sistema, se može pronaći u radu **Skalabilan mikroservis orijentisan sistem namenjen berzi kriptovaluta** [1]

### Komunikacija u distribuiranim sistemima

Postavlja se pitanje odnosa celina distribuiranog sistema? Bilo koja arhitektura i odnos delova distribucije da se primenjuje jedno je sigurno: delovi sistema moraju da komuniciraju. Komunikacija unutar sistema se može definisati kao razmena podataka. Dakle, svaki distribuirani sistem sadrži neki vid komunikacije između svojih sastavnih delova.

Vrste komunikacije mogu da budu različite. Podelu je moguća po raznim kriterijumima i dobijaju se sasvim jasne definicije i uočljive razlike. U ovom radu neće biti obrađeno svaka podela, štaviše samo će se neke od njih pomenuti i jedna detaljno objasniti. Do toga se dolazim uvođenjem sledećeg izraza naslov.

### Komunikacija orijentisan porukama

Prilikom razmene podataka postoje različite svrhe i upotrebljivosti razmenjenih podataka. Ovaj rad se fokusira na komunikaciju orijentisanu ka porukama. To znači da je tok podataka ključni pokretač sistema. Samim tim poruke predstavljaju prioritet:

U ovakvim sistemima svaka poruka je ključna i gubitak podataka se ne toleriše. Svaka poruka ima svoju težinu i obavezno mora biti ispostavljena svim delovima sistema koji to zahtevaju.

Do ogromnog značaja dolazi pitanje komunikacije u trenucima ekspanzije upotrebe sistema. Prethodno postavljena težina poruka predstavlja problem tek kada sistem postaje zasićen i pokazuje prve simptome zagušenja. Zbog toga je neophodno se u ranim fazama dizajniranja arhitekture komponenti sistema ima na umu mogućnost opterećenosti sistema, te se na odgovarajući način pripremiti za to. U tim fazama odabir načina i alata komunikacije predstavlja tešku odluku.

## Brokeri poruka

Slika . Put do brokera poruka

Broker poruka je alat, odnosno softver, koji ima ulogu razmene poruka. Radi lakšeg razumevanja, pre objašnjenja čemu služi i za šta se koristi, biće objašnjeno kako jedan sistem dođe do potrebe za brokerom poruka. Ilustracija na slici 1.3 je slikoviti prikaz tok i sastoji se od, grubo rečeno, koraka:

1. U prvoj fazi se ne vidi potreba za distribuiranim sistemom, te se razvije jedno softversko rešenje.
2. Nakon nekog vremena rada, dolazi do potrebe za pomoćnim alatom koji dopunjava rad prvobitnog. Ova 2 alata međusobno komuniciraju direktom vezom, jedan od njih poziva drugog po nekom protokolu.
3. Narednim zahtevima dolazi do potrebe za uvođenjem još 2 alata koji takođe direktno komuniciraju sa delovima sa kojima ima potrebe. U ovom koraku se struktura može nazvati distribuiran sistem.
4. Komplikovanjem zahteva biznis logike i daljim razvojem uvode se sledeće potrebne celine. Dodavanjem još celina i ostankom na direktnoj komunikaciji dolazi do haosa. U 4. koraku distribuirani sistem ima 7 celina i svaka celina direktno poziva neku. Postaje previše komplikovano ispratiti tok komunikacije, povezanost celina i ko s kim komunicira.
5. Kada sistem poraste do koraka 4 i postane nejasno definisati komunikaciju, uvodi se pomoćni alat koji vodi računa o komunikaciji. To je broker poruka. Svi delovi sistema komuniciraju sa brokerom i jedina direktna komunikacija svakog dela sistema je sa brokerom. Veze između delova sistema i dalje postoje, neophodna komunikacija je i dalje prisutna, samo što ona nije direktna, već se odvija preko brokera.

Broker ima zadatak da prati tok podataka. Može se zamisliti kao raskrsnica, saobraćajnica poruka. Svaka poruka stigne do njega, te on usmerava kuda sve ta poruka treba da ode. Neko bi rekao: „Da, ali veza nije direktna“; tako je, veza između celina nije direktna. A da li je neophodno da bude? Delovi sistema postaju nezavisniji, odrade svoj zahtev i odgovor šalju na broker. Ni jedna celina ne mora vodi računa kome sve odgovor treba da ispostavi. Štaviše, u dobro dizajniranim sistemima celine ne bi trebale da imaju dodirnih tačaka s drugim celinama.

Radi pojašnjenja prednosti upotrebe brokera poruka uzeće se u analizu poređenje 4. i 5. koraka sa ilustracija prikazane na slici 1.3:

* Nezavisnost delova sistema – svaka celina je priča za sebe. Radi svoj posao i ne komplikuje logiku brigom o tome kome treba dostaviti podatke. Na 4. koraku deo sistema ilustrovan s desne strane gore šalje podatke na 3 druga procesa. Neophodno je voditi računa o sva 3 procesa, dostupnosti istih i slično. Kad bi se dodao još jedan proces kojem pomenuti proces šalje podatke, bilo bi potrebno raditi izmene na istom kako bi se podaci usmerili i ka novom procesu. U slučaju korišćenja brokera poruka, pominjani proces poruke šalje na broker i ne interesuje ga ni dostupnost drugih procesa ni dodavanje novih.
* Nadzor toka podataka – na 4. koraku bi bilo izuzetno teško, gotovo nemoguće napraviti nadzor toka poruka u smislu koliko podataka vezanih za određeni deo i kada bude opterećen. Na koraku 5 bi se prostim dodavanjem još jednog procesa za nadzor moglo napraviti rešenje analize svih poruka.
* Ponovno iskorišćenost – ova prednost se nadovezuje na nezavisnost. Naime, kada se rešenja pišu univerzalno, nezavisno od sistema mnogo ih je lakše iskoristiti kasnije i kao deo nekog druge celine. Dok bi, s druge strane, jedan proces iz koraka 4. zbog direktne komunikacije s drugim delovima te usko zavisnim od njih, bio mnogo teže primenljiv u nekim drugim sistemima.
* Sigurnost isporuke – kod brokera poruka sam broker je taj koji vodi računa da poruka bude dostavljena na sve neophodne delove. Dok u slučaju prikazanom na koraku 4 svaki proces bi morao da ima svoju logiku provere dostave poruka

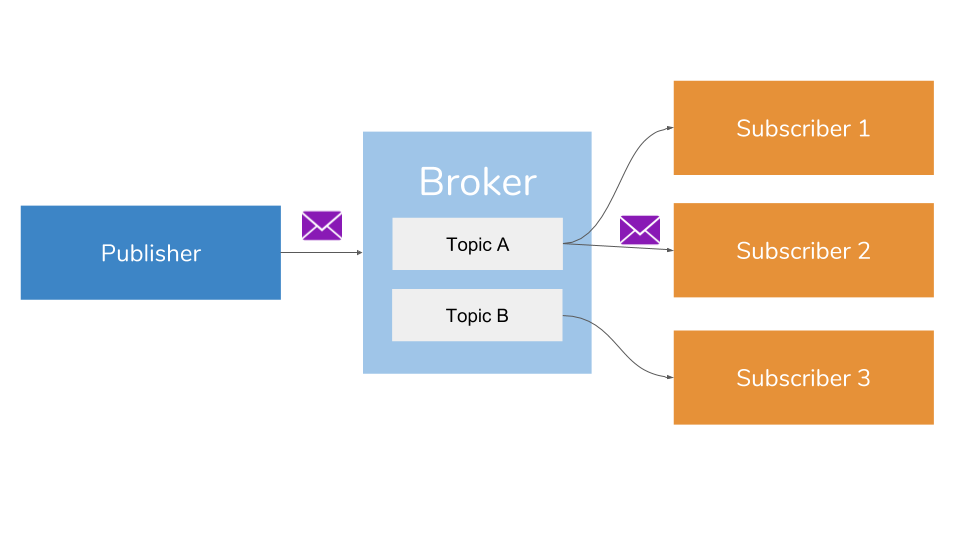
Pored prednosti, naravno, postoje i mane korišćenja brokera. Glavna mana ogleda se u potrebi za postavljanjem te i održavanjem još jednog softvera u sistemu. Ova mana povlači niz omanjih potencijalnih problema kao što su: potreba resursa brokera, mogući problemi dostupnosti, poteškoće konfigurisanja i slično. Međutim, naveden problem je konačan i zanemariv u odnosu na prednosti upotrebe brokera.

## Publish-subscribe patern

Jedna tačka zajednička kod svih brokera poruka: a to je šablon uloga u komunikaciji. Naime, svi brokeri poruka rade na tzv. publish-subscribe šablonu (od engleskog *publish-subscribe*, prevod bi bio objava-pretplata. Zbog prirodnijeg izgovora u ovom radu će se koristiti prisvojene engleske reči). Ovaj šablon predstavlja raspored uloga u komunikaciji. Prisutne su 3 uloge:

1. Publisher – subjekat koji šalje poruke
2. Broker – sam posrednik. Subjekat koji je u konekciji sa publisher-om i subscriber-om, te razmenjuje poruke.
3. Subscriber – subjekat koji se pretplatio, te prima poruke.

Na slici 1.4 su slikovito predstavljene uloge.



Slika . Uloge u pabliš-subskrajb

Pored uloga, još jedan termin je neophodno definisati: topik (eng. tema). Topik može da se predstavi kao kanal toka podataka. Podaci u različitim topicima se ne mešaju. Subscirber prilikom subsribe-a (pretplate) navodi topik na koji se subscribe-uje. To znači da će on dobijati sve poruke koje se šalju na navedeni topik. Takođe, publisher u trenutku publish-a (slanja poruke) navodi na koji topik šalje poruku. U trenutku kada publisher pošalje poruku na određeni topik, broker prima poruku, te je prosleđuju svim subscriber-ima koji su se subscribe-ovali na taj topik.

Brojnost publisher-a i subscriber-a nije ogranicena. Broker poruke raspoređuje te prosleđuje poruke po topicima. Broj publishera povezan sa brokerom ne igra ulogu. Svi subscriber-i koji su subcribe-ovani na određeni topik će dobiti publish-ovanu poruku. Dakle, ni broj subscriber-a nije bitan za broker.

Jedan hronološki ispraćen primer toka:

* Subscriber 1 uspostavlja konekciju sa broker
* Subscriber 1 se subscribe-uje na topik *Temperatura*
* Subscriber 2 uspostavlja konekciju sa broker
* Subscriber 2 se subscribe-uje na topik *Temperatura*
* Publisher šalje poruku „23.44“ na topik *Temperatura*
* Broker prima poruku i prosleđuje subscriber-ima
* Subscriber 1 i subscriber 2 dobijaju oba po jednu poruku „23.44“

Navedena pravila i ograničenja su uopštena. Kod različitih implementacija brokera se dešava da variraju neka od navedenih pravila. Primeri varijacija će biti obrađeni u nastavku ovog rada.

Poglavlje 2

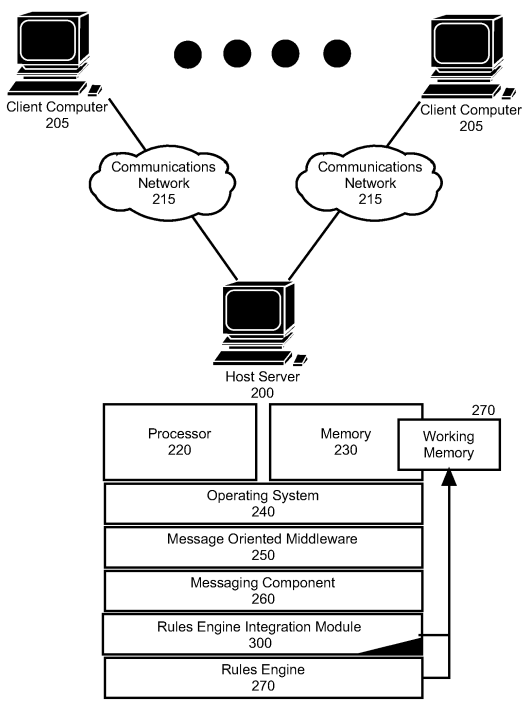
# Istraživanje na temu

Ovo poglavlje opisuje radove i publikacije sa sličnom tematikom istraženih u cilju poređenja i izučavanja problematike ovog rada. Mnogo radova postoji koji se bave skalabilnošću te ograničenjima i proširenjem ograničenja nekih sistema. Radovi su najčešće konkretnih izučavanja i istraživanja te se bave odabranim tehnologijama.

Rad na temu **Skalabilan mikroservis orijentisan sistem namenjen berzi kriptovaluta [1]** se bavi predstavljanjem primera distribuiranog sistema sa konkretnim namenama u konkretnoj tehnologiji. Tema je vezana za distribuirane sisteme i bavi se razvojem i pojašnjenjem primera jednog skalabilnog i visoko dostupnog sistema. Arhitektura komponenti sistema je napravljena tako da bude pogodna za horizontalno skaliranje [4] samih komponenti. Komunikacija između komponenti je direktna i to REST API protokolom.

U navedenom radu je korektno postavljena arhitektura u cilju skalabilnosti, ali svojom arhitekturom nalazi se na 2. ili 3. koraku sa ilustracije prikazane na slici 1.3. To znači ,da bi vremenom i daljim razvojem tog sistema, te dodavanjem komponenti i zahteva s vremenom došlo do potrebe za korišćenjem brokera. Možda ne strogo neophodno, ali svakako bi poželjno bilo prebaciti komunikaciju preko brokera.

Patent pod sa naslovom **Porukama orijentisan posrednih sa integrisanim mehanizmom pravila** [2] predstavlja rad na temu usko vezanu ovom radu. Inicijalna problematika oba rada se podudara, a to je kako uprostiti komunikaciju u distribuiranim sistemima. U navedenom radu se iznosi potreba za izdvajanjem alata koji će se baviti samo posredovanjem poruka. Time se donose značajne prednosti i olakšanja u distribuiranom sistemu.

Navedeni patent i ovaj rad imaju usku povezanost teme i inicijalne problematike. Međutim, bitna razlika je prisutna: pomenuti rad predstavlja patent, dok je ovo naučno istraživački rad. U patentu se predstavlja arhitektura implementacije posrednika poruka. Pored arhitekture, navode se ograničenja moguća upotreba hardvera. Osim toga, može se pronaći i detaljno obrađeni slučaji obrađivanja poruka u smislu dijagrama toka podataka unutar posrednika.

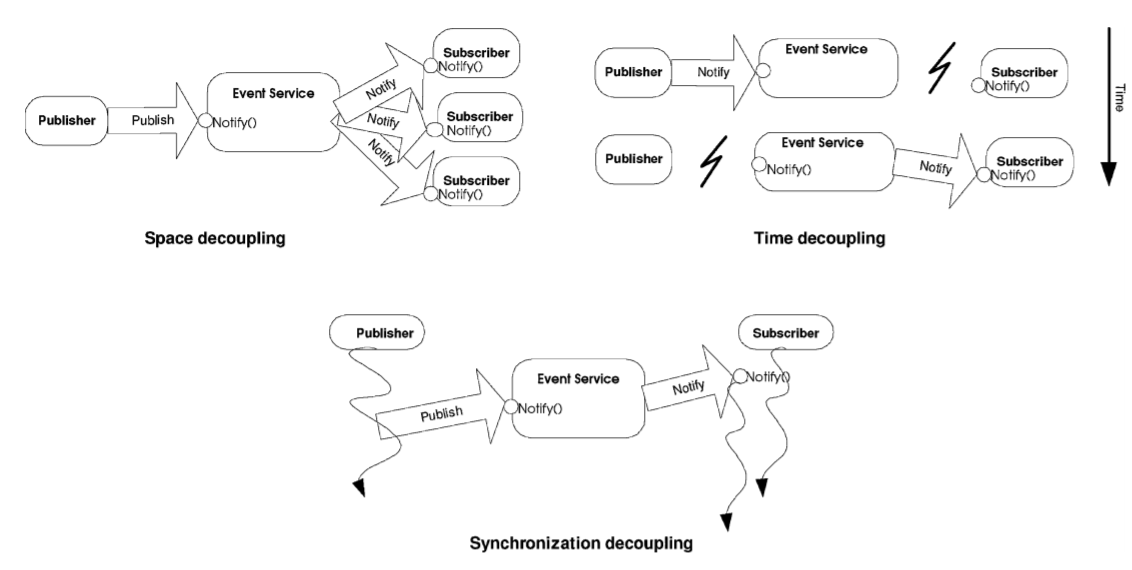
Slika . Ilustracija posrednika sa integrisanim mahanizmom pravila

Ovaj rad je fokusiran na upotrebu posrednika na višem nivou apstrakcije, ne obrađuje detalje vezane za implementaciju posrednika kao ni na moguće mehanizme pravila posrednika. Posrednik sa integrisanim mehanizmom pravila predstavlja otvorenu mogućnost za bilo koji način implementacije pravila raspoređivanja poruka. On može da bude implementiran s bilo kojim mehanizmom. Dok se, s druge strane, ovaj rad fokusira na publish-subscribe šablon implementiran u postojećim gotovim rešenjima, te poređenje istih. U ovom radu neće biti implementiran posrednik, već će se porediti postojeći posrednici u konkretnom šablonu.

Rad koji značajno detaljnije razrađuje publish-subscribe paradigmu je **Vise lica publish/subscribe-a** [3]. Naveden rad se bavi razjašnjavanjem komunikacione paradigme. Vrši poređenje sa mnogim drugim načinima komunikacije između celina distribuiranog sistema, kao što su:

* *message queuing* (red opsluživanja poruka) *-* razmena poruka kroz redove opsluživanja,
* *RPC/RMI -* RPD - Remote Procedure Call; RMI – Remote Method Invocation; oba označavaju pozivanje funkcija odnosno metoda udaljenog sofrvera
* *Shared space* (deljena memorija) – ramena podataka deljenjem iste memorijske jedinice (može biti fajl sistem, baza podataka i sl.)
* *Notifications* (obaveštenja) – prisutna upotreba *obećanja* (*promise* poznatiji u web razvoju); *observable* patern;

Kako se u samom radu navodi, ovi načini komunikacije su preteča publish-subscribe-a. Svaki od njih može da se koristi i ostvariva je komunikacija, ali su prisutne značajne prednosti upotrebe publish-subcribe-a. U radu se prednosti iskazuju u razdvojenosti sistema po 3 dimenzije: prostor, vreme i sinhronizovanost (slika 2.2).



Slika . Odvojenost u 3 dimenzije

Svaka dimenzija odvojenosti je prethodno pominjana u ovom radu, u delu gde se naznačene prednosti upotrebe brokera zaokruženo kao nezavisnost delova sistema, dok pomenuti rad detaljnije objašnjava. Odvojenost komponenti u prostoru se ogleda u ne korišćenju istog memorijskog prostora. Delovi sistema komuniciraju sa brokerom i dodirna tačka deljenja memorije im je memorija brokera, ali tim prostorom sam broker upravlja. Vremenska odvojenost se ogleda u nezavisnosti delova sistema od slanja do isporuke poruke. Postoji mogućnost da u momentu publish-a subscriber nije prisutan, te da poruku primi naknadno (ilustrovano u gornjem desnom uglu slike 2.2). Desinhronizovanost, odnosno odvojenost sinhronizacije, je jasna jer nijedan subjekat nema čekanje odgovora, dakle nakon publish-a publisher nastavlja s radom. S druge strane, subsciber takođe asinhrono i nezavisno prima poruke.

Interesantan deo rada ogleda se u varijacijama publish-subscribe-a. Naime, navikli smo, i ovaj rad obrađuje publish-subscribe baziran ta topicima. Svaka poruka se šalje na topik, subscriber-i naznačavaju koji topik ih zanima. Međutim, to je samo jedan od tipova publish-subscribe komunikacije i nazivaju ga topik bazirani. Postoje i drugi kao što su: tipski bazirani i sadržaj bazirani. Ovaj rad se zadržava na najzastupljenijem i najprirodnijem topik baziranom.

## Odabir brokera

Daljim istraživanjem nailazi se na rad slične teme, ali novijeg datuma. Rad pod nazivom **Osiguran nekasneće i skalabilno širenje podataka za aplikaciju pametnih gradova [5]** se bavi problemom komunikaciju u konkretnom distribuiranom sistemu. Domen primene je pametan grad. S obzirom na domen, očekuje se ogromna količina podataka te je neophodno odabrati odgovarajući alat za komunikaciju. Povezanost sa ovim radom se ogleda bas u odabiru paradigme jer i taj rad bira publish-subscribe kao odgovarajući zahtevima.

Pomenuti rad predstavlja paralelu ovog rada. Pre svega, u tom radu se pominju implementacije brokera koji su predmet i ovog rada, što je direktna posledica datuma kada je rađen rad. Pominju se konkretno: MQTT, Kafka i Redis uz navođenje da se sve više koriste u realnim IoT (*Internet of Things*) sistemima. Konfiguraciono parametrisanje i samo testiranje ponašanja je ekvivalent eksperimentalnog dela ovog rada. Razlikuje se što je u tom radu testira njihova interna implementacija publish-subscribe brokera koja najsličnija Kafki. Ipak ovaj rad pokriva eksperiment sa 3 brokera.

Poglavlje 3

# Brokeri

Poglavlje „brokeri“ predstavlja teorijski uvod u eksperimentalan deo ovog rada. Potrebno je navesti odabrane implementacije brokera, te razjasniti šta su osnovne karakteristike svakog od njih. Potreba za detaljnom teorijskom bazom je neophodna radi lakšeg razumevanja samog eksperimenta.

## MQTT - *mosquitto*

Prvi ujedno i najprostiji broker je MQTT. MQTT je skraćenica od *MQ Telemetry Transport* (eng. MQ telemetrijski transport). *MQ* je preuzeto od IBM MQ, gde MQ predstavlja skraćenicu od *message queue* (eng. red posluživanja). Cela skraćenica je, dakle, *message queue telemetry transport*.

MQTT predstavlja lako-kategorijski protokol razmena poruka [7]. Realizuje, prethodno objašnjen, publish-subscribe šablon komunikacije. Kao broker koristi direktnu vezu za komunikacionim uređajima. Direktna veza sa klijentima se realizuje putem mrežnog TCP/IP protokola ili, takođe moguća, websocket komunikacija. MQTT podržava i bezbednu komunikaciju na bazi SSL-a (*Secure Sockets Layer*). Najčešće se koristi, te i najznačajniju ulogu ima, u polju IoT-a.

Uz MQTT se uvek usko veže termin *lightweight*, odnosno lako kategorijski. To znači da je MQTT nužno prost protokol komunikacije. Lako kategorisanje predstavlja pogodnu upotrebu i za bežične konekcije sa mogućim kašnjenjem i gubitkom podata. Lakoća protokola se ogleda obostrane, kako klijentski tako i serverski. Klijentske biblioteke MQTT protokola su široko dostupne u većini programskih jezika i neopterećujuće su po softver zbog prirode samog protokola.

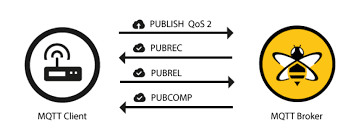
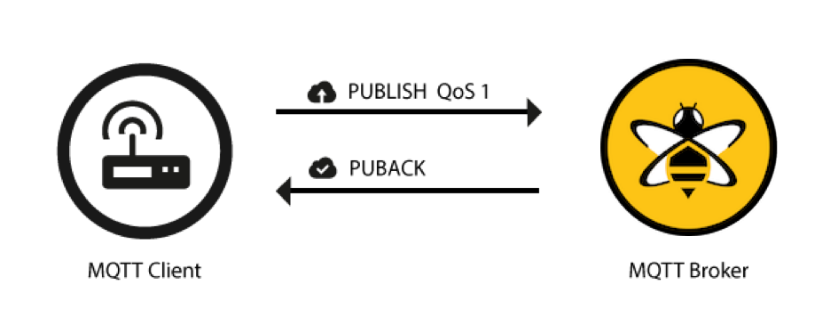
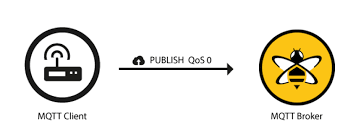
Slika . MQTT logo

MQTT broker, odnosno server, mora biti dostupan svim klijentima u komunikaciji. Najčešće se MQTT postavlja na mrežu u *cloud* kako bi bio dostupan svima. Klijent, u MQTT komunikaciji, je bitno da bude na mreži (internet ili lokalnoj) te da mu server bude vidljiv.

Publish-subscribe šablon sadrži 2 tipa klijenata. Bez obzira koji klijent je u pitanju, prvo i neophodno je uspostavljanje komunikacije sa serverom, odnosno MQTT brokerom. Zavisno od podešavanja, postoji mogućnost potpuno otvorene konekcije, obezbeđene autentifikacijom kao i bezbedne (secure) komunikacije u baziranom na SSL-u. Po uspostavljanju komunikacije klijent može da se subscribe-uje na topik ili da publish-uje neke poruke. Moguće je da jedan klijent bude istovremeno i subscriber i publisher, jer po subscribe-u može i sam da šalje poruke.

### Quality of Service

Karakteristično za MQTT je *Quality of Service*, skraćeno QoS, odnosno kvalitet usluge. Klijenti prilikom obe akcije (i publish i subscribe) definišu QoS. Numerička vrednost 0, 1 ili 2 koja označava stepen proverenosti razmene poruke između klijenta i servera. Svaka oznaka ima svoj naziv i objašnjenje:



Slika . MQTT Quality of Service

* 0 – *najviše jednom (at most once)*– nazivaju ga još i *pošalji i zaboravi*(*fire and forget*); klijent pošalje poruku i ne očekuje potvrdu. Zbog toga postoji mogućnost da usred gubitka podataka poruka nikad ne bude isporučena, a najviše je moguće jednom da se isporuči jer se ne vrši ponovno pokušavanje.
* 1 – *najmanje jednom (at least once) –* klijent pošalje i očekuje potvrdu o prijemu od servera. U slučaju da nema potvrde, poruka će se pokušati ponovo slati. Iz tog razloga, poruka će biti dostavljena bar jednom, ali postoji mogućnost i više puta u slučaju da se izgubi paket potvrde.
* 2 – *tačno jednom (exactly once)* – za razliku od prethodnog, ovde se nakon potvrde od strane servera ponovo šalje potvrda da je slanje završeno. Ovakva definicija garantuje da će poruka biti isporučena tačno jednom.

Objašnjenje razmene poruka je predstavljeno i na slici 3.2. Shodno razmenjenim porukama postoje prednosti i mane različitog QoS-a. 0 je najjednostavniji te i najbrži. 2 je najsigurniji i najprecizniji, ali zahteva mnogo dodatne komunikacije za svaku poruku, te je značajno sporiji. U okviru praktičnog dela ovog rada će se obraditi i poređenje komunikacije za različite QoS-e.

### Zamenski karakteri pretplata na topik

Konvencija MQTT protokola nalaže da se ime topika gradi po nivoima te da se nivoi razdvajaju kosom crtom: „/“ (nije pravilo, niti je neophodno, već samo preporučivo). Tako bi, na primer, topik temperature u dnevnoj sobi bio: *kuca/dnevnasoba/temperatura*; za svetlo u istoj sobi bi bilo: *kuca/dnevnasoba/svetlo.*

Osnovni model subscribe-a jeste da se navede konkretan i ceo naziv te se primaju poruke poslate na navedeni topik. Međutim, MQTT sadrži još jednu prednost, takozvani *wildcards*, odnosno zamenski karakteri.

* # - predstavlja zamenski karakter za više nivoa. To znači da će klijent koji se pretplate sa nekim prefiksom i # na kraju dobijati sve poruke sa koje imaju isti prefiks. Isto važi i za sufiks, samo se # navodi pre sufiksa. Na primer, preplatiti se na *kuca/#* označava da će stizati sve poruke sa prefiksom *kuca/* , dakle i sa topika *kuca/dnevnasoba/svetlo, kuca/kuhinja/temperatura* kao i sa *kuca/vlaznost.* Primer za sufiks bi bio *#/temperatura;* kada bi stizale sve poruke sa sufiksom na *temperatura,* dakle i *kuca/kuhinja/temperatura, kuca/dnevnasoba/temperatura* ali i *vikendica/soba/temperatura.* Naravno, ne postoji na ograničenje sufiksa i prefiksa, dakle moguće je koristiti
* + - predstavlja zamanski karakter za tačno jedan nivo. Dakle, klijenti koji se pretplate na topik sa zamenskim karakterom + umesto konkretnog nivoa, dobijaće poruka bilo koja oznaka da stoji na tom nivou ukoliko se ostali nivoi poklapaju. Na primer, pretplata na *kuca/+/temperatura* označava prijem svih poruka kojima je prvi nivo *kuca,* i treći nivo *temperatura*.

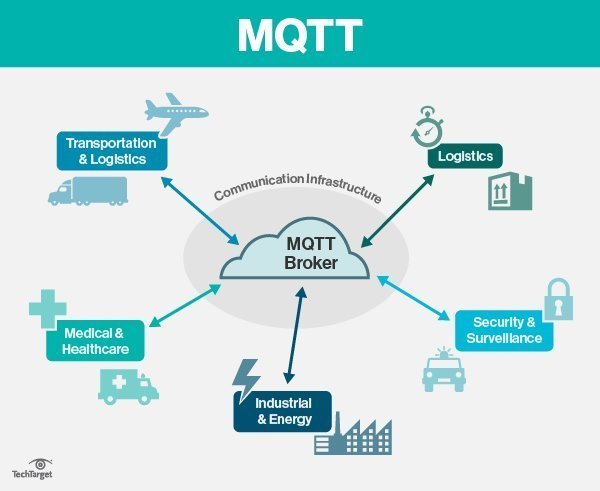
Korišćenjem nivoa u okviru topika i zamenskih karaktera prilikom pretplate MQTT protokol dobija značajno opširniju i moćniju mogućnost kombinovanja razmene poruka.

### Zadržane poruke

MQTT protokol podržava zadržavanje poruke, takozvani *retain*. Publisher prilikom slanja poruke može da pošalje sa *retain* *flag*-om. Poruke naznačene za zadržavanje broker čuva vezano za topik. Prilikom pretplate novog klijenta na topik, odmah će dobiti poslednju poruku poslatu na taj topik sa oznakom zadržavanja, bez obzira kada je poruka zaista poslata.

Dobra praksa je slati uvek poruke sa oznakom zadržavanja kada publisher šalje nekakvo stanje, odnosno ako se šalju promene stanja. Tada će svaki subscriber po pretplati dobiti uvid u poslednje prijavljeno stanje.

### Upotreba

Zbog svoje lakoće MQTT je izuzetno pogodan za upotrebu u IoT-u. Najčešće se koristi za slanje podataka sa pametnih uređaja u *cloud* u cilju prikupljanja podataka radi prikazivanje trenutnog stanja, analize i održavanja. Konkretni primeri korišćenja su ilustrovani na slici 3.3 a to su: transport i logistika, pametne kuće, industrija, medicina i zdravstvena briga.

Slika . Upotreba MQTT-a

Najveća prednost MQTT-a se ogleda u rasprostranjenosti. Dugi niz godina je prisutan i u upotrebi pa gotovo sve platforme za IoT kao većina cloud provajdera imaju podršku za MQTT, kao na primer: Amazon Web Serivces (AWS), Google Cloud, Microsoft Azure, IBM Cloud, pa i InView Web SCADA.

Osim od strane raznih providera prostranost MQTT-a se ogleda i u prisutnosti biblioteka za klijentske aplikacije. Na primer, *Paho* [8] je projekat od strane *Eclipse foundation-a* koji je razvio mqtt biblioteke za većinu danas upotrebljivanih programskih jezika.

Konkretan primer upotrebe MQTT protokola je prisutan od strane gigantske i opšte poznate korporacije Facebook. Naime, Facebook u svom proizvodu *Facebook Messenger* koristi MQTT protokol. Lakoća protokola obezbedila je ekonomičnost i nisku potrošnju baterije prilikom upotrebe na telefonu.

### Mosquitto

MQTT predstavlja protokol komunikacije, dakle skup pravila komunikacije. Da bi se obezbedila komunikacija putem MQTT-a, pored klijenata, potrebno je obezbediti MQTT server.

Mnogo *cloud* provajdera nude svoj MQTT server kao gotovo rešenje. Problem ovakvih rešenja je što se najčešće plaćaju, ili je besplatna upotreba ograničena ako uopšte postoji.

Pored gotovih rešenja već dostupnih na mreži postoje i implementacija servera. U pitanju je softver koji predstavlja MQTT server. Moguće ga je instalirati na razvojnom računaru u svrhu testiranja. Osim testiranja, isti softver je moguće koristiti i u produkciji ako se postavi na javno dostupnu adresu. Evo i primera ovakvih MQTT brokera: RabbitMQT, HiveMQT i Mosquitto. Ovo su samo neki primeri. Postoje još mnogo sličnih. Najčešće, svaki od njih po instalaciji sadrži i test klijent za publish i subscribe.

U ovom radu, u okviru eksperimenta, će se koristiti Mosquitto koji je proizvod već pominjane *Eclipse Foundation.* Broker je potpuno besplatno i slobodan za svaku vrstu upotrebe dostupan na zvaničnom sajtu [9]. Upotreba je otvorena za većinu najčešćse korišćenih operativnih sistemima. Implementira MQTT protokol u potpunosti i bez ograničenja

## Redis

Redis je izvedenica od *Remote Dictionary Server* , što bi u bukvalnom prevodu značilo udaljena serverski rečnik (rečnik u smislu mape). Redis je brza serverski alat koji služi kao keš memorija, baza podataka i broker poruka [10].

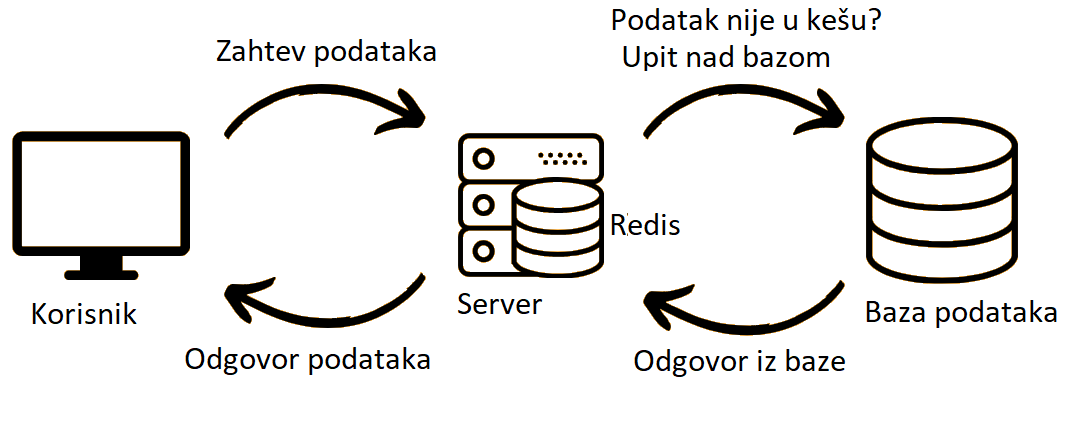
Specifičnost ovog alata je što sve podatke čuva u memoriji. Zbog toga je izuzetno brz. Takav rad softvera nosi i svoje mane u smislu ograničenja memorije, pa ga shodno tome treba i koristiti. Tešku memoriju, odnosno disk ili SSD, ne koristi prilikom opsluživanja klijenata, te operacije čitanja i pisanja može mnogo brzo da izvršava.

Slika . Redis logo

Ipak, teška memorija nije neupotrebljena kod ovog alata. Naime, Redis povremeno čuva sve podatke iz radne memorije u tešku memoriju. Jedini razlog tome je sprečavanje gubitka podataka u slučaju restarta. Zbog ovakvog ponašanja Redis je spreman u okviru milisekundi da obrađuje zahteve klijenata.

### Keš memorija

Redis se primarno i najčešće koristi kao keš memorija. Razni alate u raznim programskim jezicima nude vrlo prosto keširanje podataka iz relacionih baza u Redis. Tada, umesto čitanja iz baze, podaci se mogu dobaviti iz Redisa čime će odgovor biti mnogo brži. Osim baze podataka, keš može da se koristi i prilikom upotrebe nekih drugih resursa koji mogu da kasne, na primer, korišćenje drugih API-a.

Na lici 3.5 je prikazan primer komunikacije korišćenjem Redis keša. Dakle, na zahtev klijenta postoje 2 mogućnosti:

Slika . Ilustracija komunikacije koristeći keš

1. Traženi podatak je prisutan u keš memoriji. Tada server pročitati podatak iz keš-a i odgovoriti. Ovaj slučaj će biti ekstremno brzo izvršen.
2. Traženi podatak nije prisutan u keš memoriji. Tada server zahteva upitom nad bazom podataka. Po dobijanju odgovora iz baze, server će sačuvati traženi rezultat u keš, kako bi sledeći zahtev išao po prvom navedenom postupku, a potom i odgovoriti klijentu na zahtev.

Ključ vrednost osnovna upotreba keš memorije Redisa. Kao što i skraćenica govori: serverski rečnik, savršena terminologija za ključ-vrednost arhitekturu. Redis svaku vrednosti koju drži u memoriji čuva direktno kao ključ vrednost par. Ovakvo pravilo asocira na to da je vrednost prostog tekstualnog tipa, međutim, prednost Redisa se ogleda baš u mogućnosti čuvanja različitih tipova podataka u keš. Zbog toga se Redis predstavlja i kao baza podataka, a ne samo keš memorija. Vrednosti mogu biti tipova:

* String – tekstualni ili binarni zapisi memorije do 512MB
* List – kolekcija stringova redosleda kojeg je dodat
* Skup – sa podržanim operacijama nad skupovima (presek, unija, i razlika u odnosu na drugi skup)
* Sortiran skup
* Heš mape – mogu se čuvati osobine i vrednosti. Ovim tipom vrednosti je moguće čuvati dele objekte (entitete)
* Bitmape

### Redis pub/sub

Pub/sub, skraćenica od publish-subscribe, predstavlja Redisovu implementaciju publish-subscribe paradigme. Kao što je prethodno opisano, uloge klijenata su pusblisher i subscriber. Takođe je moguće da isti klijent bude oba od navedenog.

Zamenski znakovi postoje i ovde, kao i kod MQTT-a. Međutim, ovde ne postoje nivoi u okviru topika, već je sve kao jedan string. Zamenski karakter je \* i predstavlja više karaktera. Pa tako, ukoliko se klijent pretplati na topic *novosti\**  dobijaće poruke i sa topika *novostisrbije, novosti.zabava-dana* a i *novosti-danas-sutra*.

Iako je Redis jako brz, postoji jedna mana koja mu je neoprostiva. Naime, za razliku od MQTT-a, Redis se koristi unutar zatvorene mrežne ahritekture, dakle lokalno. Redis nije preporučljivo i ne koristi se u distribuciji fizički udaljenih sistema . Razlog je prost: ovaj alat nije namenjen za to, ne podražava ni bezbednu razmenu podataka, niti je optimizovan za komunikaciju putem intereta, niti podrđava autentifikaciju. Postoji mogućnost zaštite lozinkom, ali to je sve, ne podržava ni višestruke korisnike.

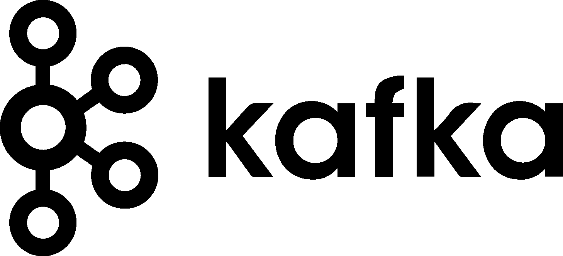
### Klaster

Redis podržava klastere. Dakle, moguće je podići više redis servera. Po dokumentaciji [11] ukoliko je povezano više servera u okviru klastera, oni će obezbediti da svaka poruka poslata na bilo koji od servera, bude poslata svim pretplatnicima na svim serverima. Trenutna implementacija samo šalje svima dalje podatke, bez optimizacija i usavršavanja komunikacije. Na tome se tek planira raditi.

Prednosti upotrebe klastera kod Redisa su uočljive kada se koristi kao keš memorija. Naime, tada će svaki server moći da koristi svu memoriju za keš, te se sistem keširanja ne ograničava na količinu memorije jednog hardvera, već je moguće povezati ih u klaster te zajedno deljeno koristiti memoriju. Međutim, tema ovog rada se fokusira na pub/sub deo Redisa.

## Kafka

Kafka predstavlja distribuiranu striming arhitekturu za obradu podataka u realnom vremenu [12]. Razvijena je od strane LinkedIn kompanije 2011 godine za interne potrebe. Deo je *Apache Software Foundation* [13].



Slika . Kafka logo

U srži, Kafka predstavlja publish-subscribe broker poruka. Međutim, razvijan je namenski da reši sledeće probleme: brzina, skalabilnost, trajnost i toleranciju greške. Karakteriše ga visok protok podataka, pouzdanost i replikabilnost. Spajanjem navedenih karakteristika, Kafka je postao alat za striming događaja.

Za razliku od Redisa i MQTT-a, Kafka koristi tešku memoriju i mnogo radi sa istom kako bi obezbedila sve zahteve.

Terminologija kod Kafke se pomalo razlikuje: umesto publish i subscribe, koriste se termini *produce* i *consume* (proizvodi i upotrebaljva). Pa tako, uloga u komunikaciji nije publisher već producer, i s druge strane, nije subscriber već consumer. Radi manje konfuzije u okviru ovog rada će biti korišćeni isti termini kao do sad, dakle publish-subscribe.

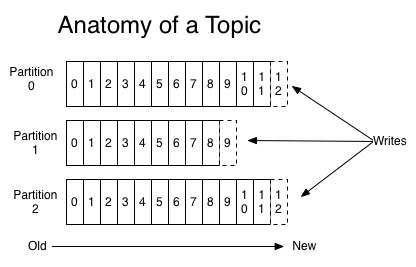
### Particije, ofseti i komit

Ključna razlika između drugih brokera i Kafke se ogleda u tretiranju poruka. Dok ostali brokeri imaju kratko memorisanje ili nemaju uopšte, Kafka dugotrajno vodi računa o svim porukama i obezbeđuje da svaka poruka sigurno stigne na svaku lokaciju, bez gubitka podataka. Mogućnost čuvanja podataka je dugotrajna, ograničava se u kapacitetima teške memorije računara (podrazumevano čuvanje poruka je 7dana, ali postoji mogućnost čuvanja podatak večno).

Da bi se objasnilo tačno kako funkcioniše, neophodno je prethodno uvesti termine specifične za ovaj alat. Prilikom kreiranja topika, neophodno je navesti koliko particija ima. Kada publisher poruke šalje na topik ona se rasporedi u jednu od particija. Subscriber, prilikom pretplate, osim topika navodi i particiju koju prati. Dakle, Kafka radi po automatici raspoređivanje poruka u cilju rasterećenja. Za korišćenje osnovnog publish-subscribe modela dovoljno je napraviti topik sa 1 particijom i svi subscriber-i će dobijati sve poruke.

Unutar svake particije topika, za svakog subscriber-a se čuva ofset. Ofset predstavlja pokazivač na poslednju poruku koju je subscriber obradio. Subscriber po inicijalnoj pretplati prima redom sve poruke (osim ako navede drugačije); po prijemu poruke šalje komit na Kafku kao naznaku da je poruka obrađena te da Kafka pomeri ofset. Jako bitno je za uočiti da unutar svake particije se čuvaju ofseti za svakog subscriber-a.

Na slici 3.7 je pokazan primer jednog topika sa 3 particije. Na taj topik je poslato ukupno 33 poruke (12 poruka na particiji 0, 9 poruka na particiji 1 I 12 poruka na particiji 2). Nove poruke se raspoređuju bez specijalnog pravila. Subscriber se pretplati na topik i određenu particiju, te ima 3 opcije:



Slika . Particije, ofseti i komit

1. *Earliest –* da čita sve poruke od početak particije
2. *Latest –* da čita samo poruke koje stignu nakon njegove pretplate
3. *Offset specific –* da prilikom pretplate navede i tačno ofset odakle želi da dobija poruke.

Ovakvim mehanizmom se dobija nešto što ni kod MQTT-a ni kod Redis-a nismo imali a to je mogućnost čitanja istorijskih poruka kao i garancija da subscriberi dobiju sve poruke, čak i one koje su publishovane za vreme kada subscriber nije bio uopšte u komunikaciji.

Čemu komplikovanje sa rasporedom topika po particijama? Kafka je dizajnirana da radi raspodelu posla, u smislu obrađivanja poruka. Do ovakve upotrebe dolazi kod vertikalnog skaliranja. Kada jedan deo sistema postaje preopterećen, podigne se još jedan isti na novoj hardverskoj jedinici. Ukoliko bi svaki od njih dobijao sve poruke za obrađivanje, izgubio bi se značaj, jer bi tada postojala 2 opterećena sistem. Naravno, postoji mogućnost aplikativnog rešavanja raspodele obrađivanja poruka, ali Kafka nudi gotovo rešenje za to. Ukoliko se 2 ista servisa pretplate na isti topik a različite particije, ravnomeran raspored poruka će biti usmeren na svakog od njih.

Kako bi se podigao nivo abstrakcije o particijama i time uprostila logika subscribera postoji grupisanje subscribera: takozvanim consumers group. Prilikom pretplaćivanja na Kafka topik nije neophodno navesti particiju, ali jeste obavezno navesti grupu subscribera. To predstavlja običan test koji služi za identifikaciju paralelnih delova sistema koji rade isti posao. Pa tako, u primeru iz prethodnog pasusa, oba servisa će se pretplatiti na topik kao ista grupa te će poruke biti ravnomerno raspoređene.

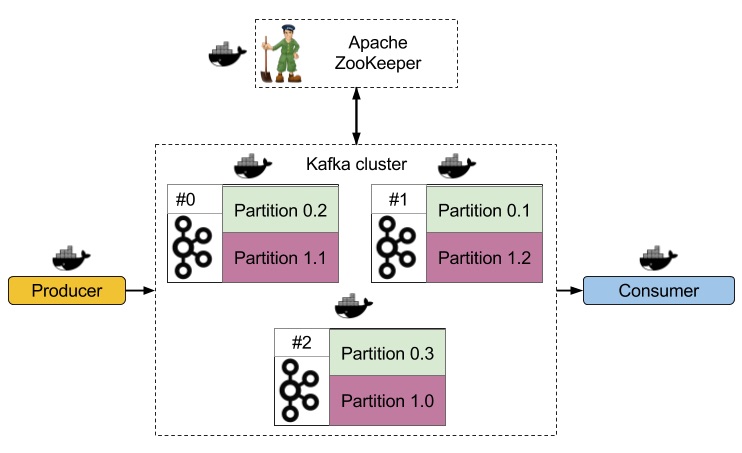
Koristeći grupe, klijenti ne vode računa o broju particija već to radi sama Kafka. Na primer, ukoliko postoje 4 particije, subscriberi bi se trebali pretplatiti na po 2 particije (mada se preporučuje da broj particija pri inicijalizaciji bude jednak broju subscribera baš iz ovog razloga). Koristeći istu grupu i pretplatom 2 servisa sa istom grupom, Kafka će automacki da ih prijavi na po 2 particije.

Slika . Ilustracija raposdele poruka po grupama subscribera

Na slici 3.8 je dat primer topika sa 4 particije i 2 grupe subscriber-a (gornji deo slike je vezan za klaster, o tome u nastavku teskta. Sada je dovoljno da se gornji deo slike posmatra kao jedan kafka broker). Na brokeru je prikazan topik sa 4 particije. Grupa subscribera A sadrži 2 subscribera, pa će po pretplati svaki subscriber primati poruke sa po 2 particije. U grupi B je predstavljeno 4 subscribera, pa će tako svaki subscriber biti zadužen za po jednu particiju. U slučajnu nedeljivih broja particija i subscribera unutar grupe neki od subscribera će dobiti više particija od drugog (razlika u broju zaduđenih particija ne sme da bude veća od 1)

### Klaster

Kafka podržava klastere. Ne samo da podržava već se i preporučuje korišćenje klastera (skoro sve jedan primer i vodič je dat bar sa 2 Kafka brokera) jer su benefiti ovog alati izraženi prilikom upotrebe klastera. Kafka serveri unutar klastera ne komuniciraju direktno. Spona između severa je pomoćni alat *Apache Zookeeper*.



Slika . Kafka klaster ilustracija

Konfigurisanje klastera je intuitivno i jednostavno. U konfiguraciji samog Kafka servera se navodi adresa do Zookeeper-a. Zookeeper je taj koji održava konzistentnost između klastera. Klijenti prilikom konekcije sa klasterom navode jednu ili više adresa kafka servera. To znači da je moguće kačiti se uvek na jedan server. Po konvenciji implementacije kafka klijenata uvek je moguće navesti adrese svih kafka servera. S druge strane, Kafka klasterima se može pristupati i putem zookeeper-a. Moguće je prosto umesto adresa kafka servera navesti adresu zookeepera. Na slici 3.9 je predstavljena ilustracija arhitekture kafka klastera.

Unutar kafka klastera politika raspodere particija topika je ujednačena raspodela po različitim serverima. Ukoliko je topik napravljen sa više od jedne particije, particije će biti ravnomerno raspodeljene unutar klastera na različite servere.

Klaster upotreba predstavlja horizontalno skaliranje Kafka servera. Time se značajno podiže nivo dostupnost i pouzdanost, kao i kapacitet obrade podataka. Iz ovih razloga Kafka predstavlja visoko pouzdan broker sa jako velikim kapacitetom protoka poruka.

### Replikacija

Prilikom kreiranja topika, pored broja particija, neophodno je navesti i faktor replikacije. Taj broj označava na koliko Kafka servera unutar klastera će se čuvati kopije svakog od particija. Preciznije, koliko će kopija podataka postojati unutar klastera.

U cilju sigurnije dostupnosti podataka, svaka poruka se kopira na više servera. Problem se javlja kod mnogo podataka, jer višestruko čuvanje istih predstavlja problem po memoriju. Međutim, Kafka koristi tešku memoriju, tako da su ograničenja mnogo manja. Pored toga, konfigurabilna je i količina puvanja podataka kao i broj čuvanih kopija: faktor replikacije.

Definisanjem prioriteta, ko čuva originalan podatak a gde se on kopira, serveri dobijaju različite uloge: *leader* i *follower*. *Leader* jedne particije je server unutar klastera koji čuva originalni podatak i primarno vodi računa o njoj u smislu ažuriranja ofseta. *Follower*-i su serveri na kojima su kopirani podaci od *leader*-a. Svaku promenu na particiji ažurira *leader* pa potom podatak biva potvrđen i na *follower*-e.   
S obzirom da *leader-a* i *follower*-a može biti ukupno koliko unutar klastera ima brokera postoji ograničenje gornjeg granice faktora replikacije, jer nema smisla da se neki podaci 2 puta čuvaju na istom brokeru. Tako je maksimalan faktor replikacije jednak broju brokera unutra klastera.

Na slici 3.10. je ilustrovan sledeći primeri: klaster sa 4 kafka brokera; topik sa 4 particije i faktor replikacije 3. Pošto je topik sa 4 particije, a klaster je od 4 brokera, raspodela je jasna, svaki server po jednu particiju, kao što i jeste predstavljeno na ilustraciji: crnom pravougaonici. Faktor replikacije je 3, što znači da se svi podaci čuvaju na 3 servera, odnosno *leader* i 2 *follower*-a. *Follower*-i su na ilustraciji prikazani narandžastom bojom. Na particiji gde je *leader* broker 1, *follower-*i su broker 2 i 3; kada je *leader* broker 3,*follower*-i su broker 4 I broker 1, itd., kao što je ilustrovano na slici.

Organizovanje particija i raspored uloga server Kafa radi automacki. Na korisniku je da po kreiranju topika navede. Primer kreiranja topika je naveden na listing 3.1. U konkrentom primeru kreiranja su navedeni parametri sa slike 3.10.

bin/kafka-topics.sh --create \

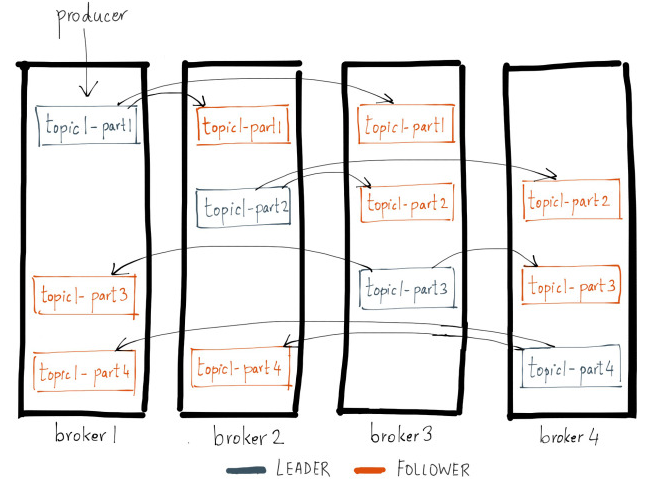
   --zookeeper localhost:2181 \

   --replication-factor 3 \

   --partitions 4 \

   --topic topic1

Listing . Kerianje topika



Slika . Leaderi i follower kod Kafka replikacija

Kada je topik kreiran postoji komanda za tzv. opis topika, odnosno prikaz rasporeda particija i replikacija unutar klastera. Na listinu 3.2 je prikazan primer opisa kreiranog topika iz prethodnog primera. Ispis Kafka opisa je ekvivalent vizualizovanom primeru na slici 3.10.[[1]](#footnote-1)

### Interfejs ka klijentima

Topic: topic1 PartitionCount:4 ReplicationFactor:3 Configs:

Topic: topic1 Partition: 1 Leader: 1 Replicas: 1,2,3 Isr: 1,2,3

Topic: topic1 Partition: 2 Leader: 2 Replicas: 2,3,4 Isr: 2,3,4

Topic: topic1 Partition: 3 Leader: 3 Replicas: 3,4,1 Isr: 3,4,1

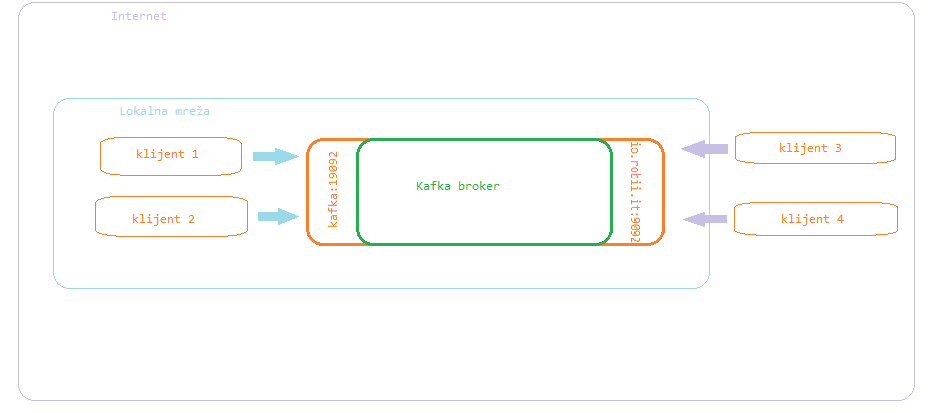
Topic: topic1 Partition: 4 Leader: 4 Replicas: 4,1,2 Isr: 4,1,2

Listing . Opis topika

Kafka pruža bezbednu komunikaciju. Razmena kroptovanih podataka na bazi SSL-a je moguće konfigurisati. Podrazumevano ponašanje ne uključuje SSL. Takođe, Autentifikacija klijenata je konfigurabilna bez ograničenja broja kombinacija korisničkih imena i lozinki (za razliku od Redis-a). Kafka podržava i autorizaciju, odnosno otvorena je mogućnost konfigurisanja koji korisnici koje topike smeju da pišu a sa kojih smeju da čitaju.

Gore navedeno podešavanje se vrši kroz konfiguraciju brokera i naziva se protokol osluškivača. Osluškivač predstavlja interfejs ka kafka klijentu. Osim različitih protokola Kafka nudi mogućnost i definisanja višestrukih osluškivača. Na taj način moguće je definisati različite osluškivače klijenata po različitim protokolima.

Različiti osluškivači su jako korisna stvar kada se Kafka broker koristi i za komunikaciju unutar mreže, lokalnu, i putem interneta, eksternu. Lokalna komunikacija može da bude prosta bez bilo kakve bezbednosti jer su podaci u lokalnoj mreži i ne preti opasnost po bezbednost. S druge strane, eksterna komunikacija je nužno obezbeđena odgovarajućom sigurnošću kako bi se zaštitilo od svih vidova napada i krađe podataka. Osluškivače je dozvoljeno imenovati prilikom konfigurisanja.



Slika . Ilustracija višesrukih osluškivača

Na slici 3.11 je dat primer Kafka brokera sa 2 osluškivača. Jedan je za lokalnu mrežu i dostupan je na adresi *kafka* i portu 19092. Drugi osluškivač je konfigurisan za spoljašnju upotrebu i dostupan je na adresu io.robii.it i portu 9092. Klijenti 1 i 2 su nuđno u istoj mreži kao i sam broker, dok su klijenti 3 i 4 bilo gde samo je uslovljeno prisustvo interneta.

## Pređenje

Nakon upoznavanja sa sva 3 ciljana brokera ovog rada, a pre eksperimentalnog dela, moguće je uraditi pregled razlika brokera.

### Skladište manipulacije podataka

Redis podacima manipulišu u radnoj memoriji i povremeno vrše čuvanje na tešku memoriju radi sigurnosti u slučaju restarta. Kafka sve podatke čuva na teškoj memoriji. Zbog toga je za očekivati da će Kafka imati donekle sporiji odziv u odnosu na Redis. Očekivana veća brzina daje prednost za Redis. Kod MQTT-a nije strogo definisano gde se manipuliše podacima jer je MQTT samo protokol. *Mosquitto,* kao implementacija MQTT brokera, podacima manipuliše u radnoj memoriji.

S druge strane, način čuvanja podataka Kafka brokeru pruža prednosti kao mogućnost istorijskog konzumiranja podataka, višestruko konzumiranje istih podataka kao i sigurniji mehanizam protiv gubitka podataka.

*Mosquitto* na tešku memoriju čuva poruke naznačene sa zadržavanjem (*retain*) da u slučaju restart može da prosledi poslednje vrednosti. Osim ovog ograničenja MQTT broker nema potrebu za bilo kakvim čuvanjem podataka na tešku memoriju.

### Šabloni topika

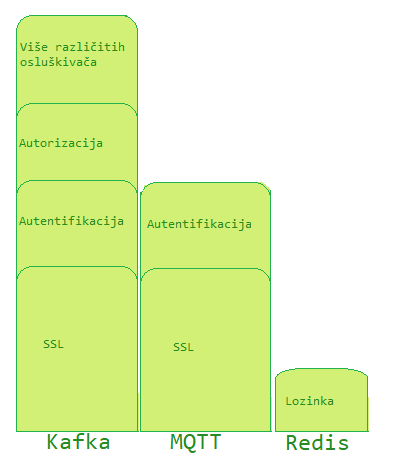
MQTT podržava najnapredniji i najintuitivniji način šablonskih topika. Nivoima i mogućim zamenskim karakterima za nivoe MQTT pruža široke mogućnosti upotrebe. Redis, slično, podržava pretplatu na topik po šablonu samo bez mogućnosti uvođenja niva, već se prepušta mašti korisnika da organizuje topike po volji.

Kafka, s druge strane, ne podržava pretplatu na topik po šablonima, ali omogućava značajne prednosti uravnoteženog raspoređivanja poruka po particijama topika.

Različite mogućnosti upotrebe su smernica korisnicima da odaberu odgovarajući alat shodno potrebama distribuiranog sistema. Svaki sistem ima svoje karakteristike i specifičnosti te će različite mogućnosti upotrebe više ili manje odgovarati.

### Bezbednost

Po pitanju bezbednosti kod ova 3 brokera postoji jednoznačno linearno poređenje:



Slika . Nivoi bezbednosti

* Kafka nudi najširi spektar mogućnosti konfigurisanja bezbednosti i visoko napredne mogućnosti
  + SSL
  + Autentifikacija
  + Autorizacija
  + Mogućnost višestrukih osluškivača s različitom definicijom
* MQTT nudi manju širinu ali zadovoljavajući nivo bezbednosti:
  + SSL
  + Autentifikacija
* Redis ne nosi atribut mogućnosti bezbedne komunikacije. Jedino što nudi je:
  + Lozinka

### Skalabinost

Na polju vertikalne skalabilnosti Kafka nudi najjaču podršku. Razlog je što je Kafka pravljena sa ciljem da predstavi skalabilno rešenje brokera poruka.

Redis podržava klaster upotrebu, ali nije nije mu to primarna upotreba i nije optimizovan za to.

MQTT ponovo zavisi od implementacije samog servera. *Mosquitto* ne podržava skalabilnost. Neke druge impelementacije pak nude mogućnost, ali nije u opisu MQTT-a i ne poklapa se sa glavnim atributom MQTT-a: *lightweight*.

Odabir se ponovo svodi na pitanje potreba distribuiranog sistema. Za distribuirane sisteme sa visokim prometom i potrebama za obradom velike količine podataka se preporučuje upotreba Kafke, dok za neko lakše i brže rešenje alata za komunikaciju će potpuno korektno poslužiti i Redis ili MQTT.

### Mehanizam pretplate podataka

Bitno za napomenu jeste mehanizam subscribera kojim dobija poruke. Naime, za Kafku može da se kaže da je maskiran publish-subscribe šablon. Za razliku MQTT-a i Redisa gde se subscriber dobija obaveštenje od servera o novoj poruci, kod Kafke se radi *pulling* mehanizam, odnosno mehanizam povlačenja poruka.

Ključna razlika se ogleda u tome što Redis i MQTT uspostave konekciju sa brokerom, pretplate se na topik i čekaju. Server je taj koji pokreće akciju slanjem poruke. S druge strane, Kafka klijent se pretplati na topik i nužno ciklično proziva server sa pitanjem: „da li ima novih poruka?“

U zavisnosti od biblioteke koja se koristi, moguće i vrlo je verovatno da se i kod MQTT-a i Redisa u okviru implementacije klijenta takođe vrši *pulling* mehanizam, ali se to nužno sakriva od korisnika. Kod Kafke je drugačija politika i očekuje se da korisnik prilagodi implementaciju tako da uzimanje poruka s Kafka bude po ovom mehanizmu.

### Gubitak podataka

MQTT pomoću različitih *QoS*-a ostavlja otvoreno za odabir korisniku nivo potrebe za osiguravanjem poruka, što je najbolje pokrivanje slučajeva.

Redis je dizajniran za upotrebu u lokalnoj mreži. Kada je u pitanju lokalna komunikacija očekivana je stabilnost veze i minimalne mogućnosti gubitka podataka. Iz tih razloga, Redis nema specifične mehanizme zaštite od gubitka podataka.

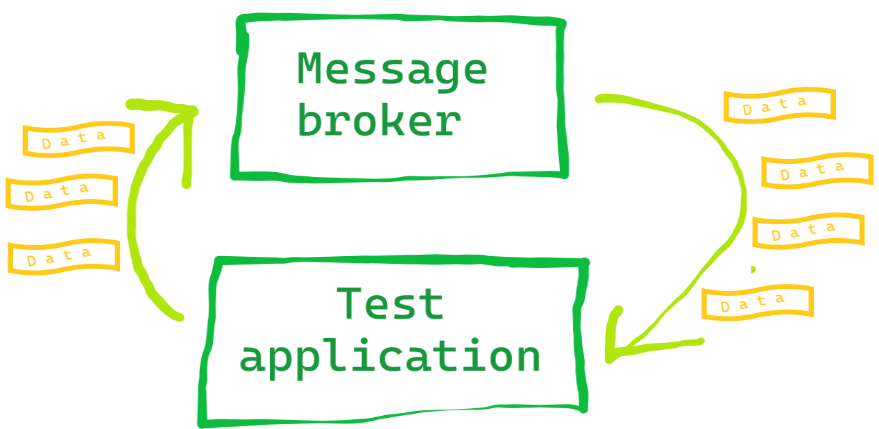
Kafka radi po principu da svaka preuzeta poruka mora da se potvrdi komitom. Ofset se pomera tek po komitu od strane klijenta. Na taj način, gotovo ne postoji mogućnost gubitka podataka.

Poglavlje 4

# Arhitektura i alati eksperimenta

Poređenje brokera urađeno u poglavlju 3 je na teorijskoj osnovi i predstavlja razlike u brokerima po dokumentaciji. Srž ovog rada je eksperimentalno poređenje brokera, odnosno odmeravanje performansi u što je realnije mogućoj simulaciji realnog okruženja. Ovo poglavlje opisuje sam eksperimentalni deo, kako je zamišljen, kako struktuiran, kako implementiran i koji u su pomoćni alati korišćeni.

Praktični deo ovog rada obuhvatao je pisanje alata za testiranje performansi. Kako bi se razmenjivali podaci kroz broker potrebno je implementirati 2 uloge: publisher i subscriber. Pošto je u pitanju simulacija i testiranje, nema prepreke da se obe uloge implementiraju u okviru iste aplikacije. Naime, posmatraju se karakteristike brokera, a ni na jedan od njih neće oticati ukoliko publisher i subscriber bude ista aplikacija. Odnosno, bez obzira što je polazna i odredišna tačka svakog podataka ista, podaci moraju da prođu putanju do brokera, procesiranje od strane brokera i tek onda do odredišta, kao što je ilustrovano na slici 4.1.

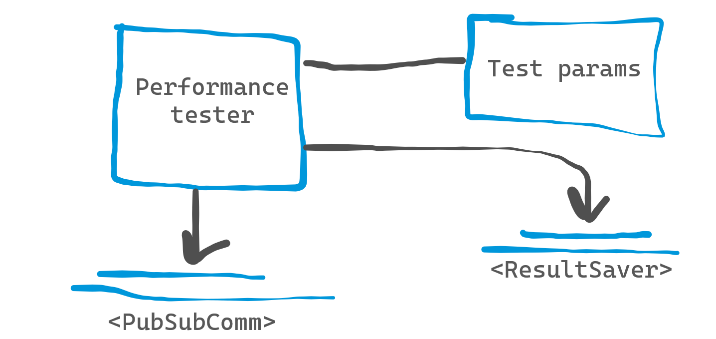


Slika . Tok podataka testiranja brokera

Kako bi se otklonile mogućnosti razlika u kodu, te i upotrebljiviji i produktivniji kod napisao, aplikacija se razdvaja na različite nivoe apstrakcije. Viši nivo test aplikacije se bavi parametrisanjem okruženja, izvođenjem samog testa u smislu slanja poruka i osluškivanja i vršenje potrebnih merenja. Niži nivo aplikacije će se baviti samom implementacijom komunikacije sa svakim od brokera. Takođe, u niži nivo spada i implementacija čuvanja rezultata.

Na slici 4.2 je prikazana skica arhitekture višeg nova aplikacije. Ključ ovog dela se nalazi u modulu *PerformanceTester*. On dobija parametre testa, koristi *PubSubComm* interfejs, kao i *ResultSaver*. Na ovaj način je softver za testiranje podeljen u 3 minimalnom zavisnošću jednom od druge. Time je i odgovornost po celinama jasno razdvojena.

Slika . Viši nivo test okruženja



PubSubComm predstavlja interfejs komunikacije po publish-subscribe šablonu. Definiše obe uloge šablona čime je zahtevana implementacija samo komunikacije takva da ista aplikacija bude i publisher i subscriber. Na listingu 4.1 je prikazan PubSubComm interfejs

Listing . PubSubComm intefejst

public interface PubSubComm extends AutoCloseable {  
 public boolean isConnected();  
 public boolean connect();  
 public void publish(String topic, String message);  
 public void subscribe(String topic, Consumer<String> onMessage);  
}

U listi metoda nije navedena metoda *disconnect* iz razloga što se prekid komunikacije radi u *close* metodi definisanom u okviru *AutoCloseable* interfejsa iz paketa *java.lang*. Nasleđivanjem *AutoCloseable* interfejsa je omogućeno da se koristi *try-with-resources* funkcionalnost [14].

## Implementacija komunikacije

Sva 3 tipa komunikacije imaju svoju klasu koja implementira *PubSubComm* interfejs. Kreiranje instanci vrši putem *Factory method* dizajn paterna.

Sve 3 implementacije konekcije se oslanjaju na neku biblioteku. Aplikacija koristi *maven*[15] te su sve korišćene biblioteke uvezane pomoću njega.

### MQTT

*MQTTPubSubComm* je klasa koja implementira *PubSubComm* interfejs kao MQTT klijent. Koristi se biblioteka iz projecta *Paho* o kojoj je pisano u ovom radu u okviru opisivanje MQTT-a (sekija 3.1.4).

Upotreba biblioteke je vrlo jednostavna i intuitivna. Koristi se *MqttClient* klasa.

Karakteristično za MQTT jeste potreba za definisanjem QoS-a (sekcija 3.1.1). S obzirom da je to specifično za MQTT a ne i za ostale brokere, taj parametar je vezan samo za ovu klasu.

### Redis

*RedisPubSubComm* klasa predstavlja *PubSubComm* kao Redis klijent. Korišćena biblioteka je takozvana *Jedis.* Ova biblioteka je preporučena sa zvaničnog sajta Redis projekta [16].

Interesantno za Redis je što pretplata zahteva pokretanje u odvojenoj niti (*thread*) jer će u suprotnom trenutna nit ostati zaglavljena i osluškivati poruke.

### Kafka

*KafkaPubSubComm* je klasa Kafka klijenta koja takođe implementira *PubSubComm*. Biblioteka koja se koristi je izdata od strane fundacije Apahe [13] koja je ujedno i potpisnik autor samog Kafka brokera.

Implementacija Kafka klijenta je složenija od prethodna 2 opisana. Naime, kafka klijent zahteva precizno definisanje raznih konfiguracionih parametara kao što su npr. serialajzer, deserialajzer, identifikacija klijenta, identifikacija grupe pretplatnika i sl. Osim više parametara, Kafka je karakteristična i po mehanizmu pretplate. Neophodno je implementirati prethodno opisan *pulling* mehanizam.

## *ResultSaver* interfejs

*ResultSaver* predstavlja deo aplikacije koji je zadužen da pamti rezultat tokom samog testa te da ga kasnije sačuva radi analize. On vodi računa kada je test pokrenut kako bi evidentirao hronološki tok testa. Svaki rezultat prijavljen od strane testera performansi pamti.

Interfejs definiše *done* metod, koji označava da je test završen i rezultat može da se sačuva po pravilu implementacije. Tokom vršenja testa se rezultati dodaju samo u memoriju. Ovakva organizacija je postavljena u cilju minimalnog opterećenja resursa za vreme trajanja testa.

### JSONResultSaver

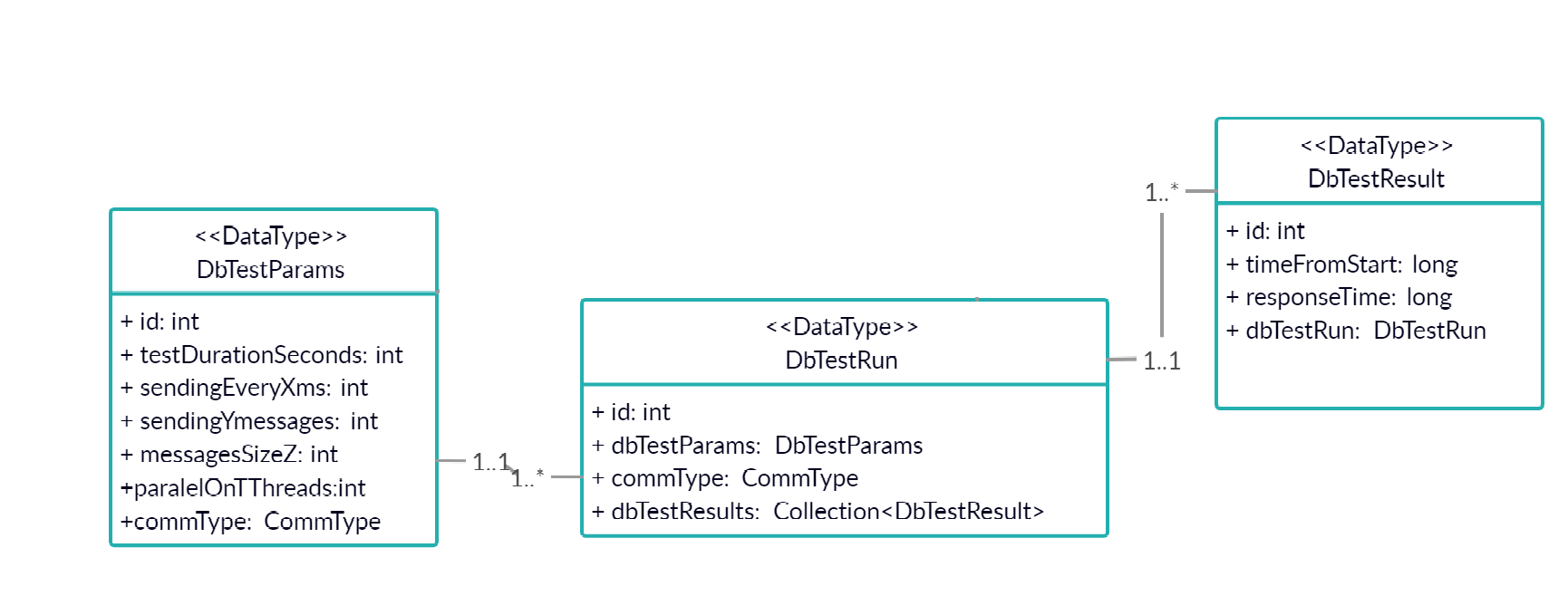
*JSONResultSaver* je klasa koja implementira *ResultSaver.* Na kraju testa sve podatke čuva u JSON file.

### DbResultSaver

*DbResultSaver* je takođe klasa koja implementira *ResultSaver*. Ona je zadužen da rezultat sačuva u relacionu bazu podataka. Ovaj rad koristi MySql bazu podataka. Za konekciju sa bazom i mapiranje entiteta korišćen je *Hibernate*.

Rezultati se čuvaju u okviru 3 entiteta. Dijagram klasa koji se čuvaju u bazu su prikazani na dijagramu.

*DbTestParams* predstavlja parametre testa. Za svaki *DbTestParam* se vežu tačno 3 *DbTestRun*-a jer ono predstavlja izvršenje teste na jednom tipu brokera. Za *DbTestRun*, odnosno za primenu parametara testa na tip brokera se veže lista *DbTestResulta.*



Dijagram . Dijagram klasa entite za čuvanje rezulata

## Tester performansi

Tester perfomasni predstavlja samu srž ove aplikacije. Razvijen je algoritam za ovaj rad koji na osnovu parametara vrši simulaciju poruka realnog okruženja. Sledeće ulazne parametre koristi ovaj tester:

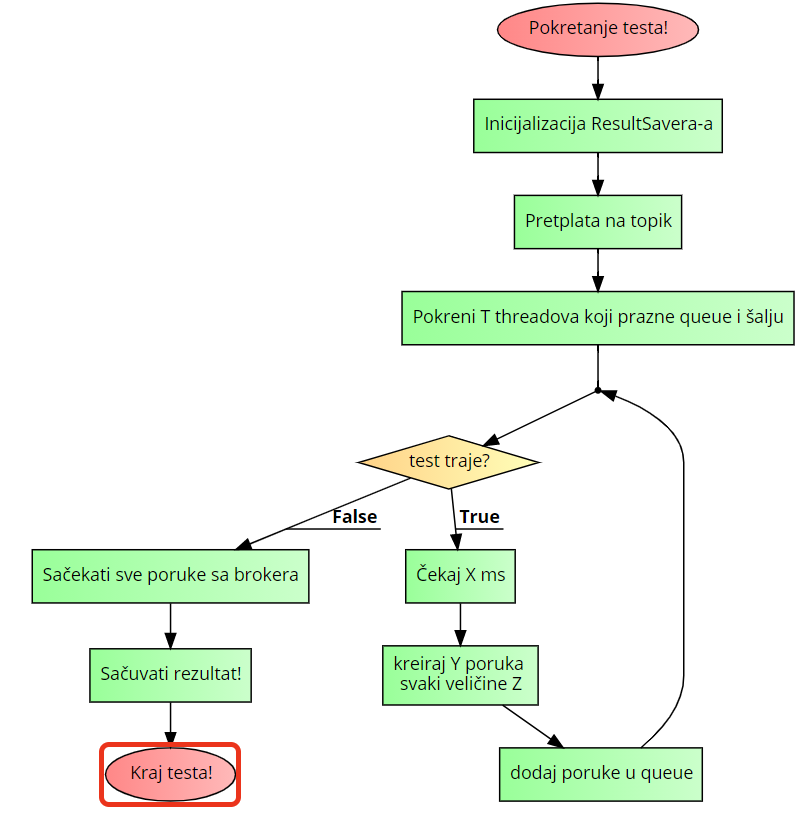
* S - dužina trajanja testa (u sekundama) -koliko dugo će se slati poruke na broker
* X - Interval između 2 slanja (milisekunde) – dužina pauze između 2 slanja
* Y – broj poruka koji će se poslati odjednom
* Z – veličina svake poruke (u bajtima)
* T – broj niti sa koliko će se istovremeno slati

Parametri su imenovani kako bi se jedan test mogao sažeti u jedno rečenicu:

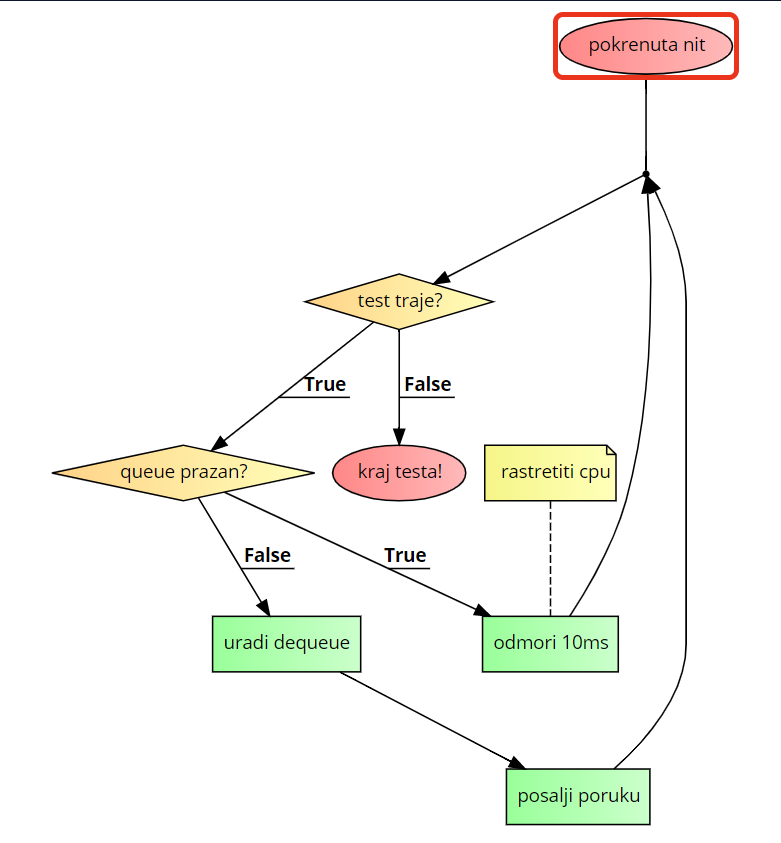
*Narednih* ***s*** *sekundi šalji svakih* ***x*** *milisekundi* ***y*** *poruka veličine* ***z*** *B paralelno na* ***t*** *niti.*

Radi realizacije paralelnog slanja algoritam se izvršava na više od 1 niti, i to tačno **t+1** niti. Glavna nit testa služi za kreiranje poruka, ostale niti rade slanje na broker. Niti imaju deljeni resurs u vidu reda opsluživanja, *queue-*a. Glavna nit testa dodaje poruke shodno ostalim parametrima u *queue*. Pomoćne niti rade konstantno i prazne *queue* slanjem poruka na broker.

Dijagram . Algoritam glavne niti testa



Algoritam glavne niti testa je prikazan na dijagramu 4.1. Dakle po pokretanju poziva se inicijalizacija nad *ResultSaver*. Zatim se uradi pretplata na topik kako bi se obrađivale dobijene poruke sa brokera. Potom se pokreću **t** komada pomoćnih niti. Sledećim korakom započinje test. Dok ne istekne **s** sekundi u petlji se rade sledeća 3 koraka: nit odmara **x** milisekundi; zatim kreira **y** poruka svaki veličine poruka svaki veličine **z** bajtova, te se te poruke dodaju u *queue*. Svaka iteracija uključuje ova 3 koraka do isteka testa. Po isteku test se sačeka da stignu sve poruke poslate na broker. Potom se sačuva rezultat.

U prethodnom algoritmu je naznačeno kako glavna nit testa radi. Na dijagramu 4.3 je predstavljen tok koda na pomoćnim nitima koje rade *dequeue* i slanje na broker.

Dijagram . Tok koda pomoćnih niti

Nit vrti petlju dok god traje test. Svaka iteracija proverava da li se nešto nalazi u *queue*-u. Kada se desi da je *queue* prazan, nit će napraviti pauzu od 10 milisekundi. Ovo je stavljeno samo radi rasterećenja procesora, da se ne bi oduzimali procesorski resursi na neprestano proveravanje da li je *queue* prazan. Međutim, dokle god ima poruka u *queue*-u, algoritam neprestano radi na pražnjenju i slanju na broker.

Kombinacijom opisane 2 paralelne radnje postignuta je maksimalna efikasnost iskorišćenosti procesora po zadatim parametrima. Prema procenama, svaki drugačiji pristup bi rezultirao nepotrebnim čekanjem ili neproduktivnim korišćenjem procesorskih resursa.

## Parametri i merenja

Postavkom problema ciljano je maksimalna širina parametrisanja. Razlog je ciljanje pokrića što je šire moguće različitih primena u sistemu. Naime, neki realni sistemi će imati jako puno manjih poruka. Neki sistemi će imati puno poruka iz različitih delova distribuiranog sistema. Treća krajnost upotrebe se ugleda u veličini poruka, neki sistem će zahtevati razmenu povećih poruka. Cilj aplikacije je može da pokrije što veći broj slučaja različitim parametrima.

Nakon postavljenih parametara postavlja se pitanje šta se tačno meri? Analiza šta bi se sve moglo meriti je bila duga i šarena. Problem je u tome što je većina podataka koje se u praksi pokazuju kao merenja protoka, u ovoj aplikaciji je postavljeno kao ulazni parametar. Na primer: količina obrađenih poruka u sekundi (izrađena u bajtovima) ili broj obrađenih poruka u sekundi. Ova aplikacija takve postavke parametriše na ulazu, a test proverava da li će broker obraditi zadato. Dakle, jedna od izuzetno bitnih rezultata je da li će broker uopšte stići da obradi podatke za zadate parametre. Rezultat ovog merenja se svode na nekoliko mogućnosti: broker obradi sve podatke, broker napravi propust podataka ili uopšte ne izdrži da radi pod navedenim opterećenjem (*app crash, fatal error*).

Osim količine protoka podataka, ključni parametar je brzina obrade brokera. Ovaj parametar je jednoznačno meriv. Naime, moguće je za svaku poruku utvrditi dužinu vremenskog intervala od trenutka slanja ka brokeru do trenutka prijema. Za ovako merenje potrebno je identifikovati poruke te je iz tog razloga na svaku poruku dodat GUID formata 8-4-4-4-12. S obzirom da je neophodno više od jednog polja, poruka je stavljena u jednu struktura sa 2 polja: guid i data, gde je guid opisana identifikacija i data neki nasumičan tekst. Na listingu 4.2 je prikazan primer jedne test poruke.

{  
 “guid”: “2767753e-eaf0-4f25-a947-dafff7ada8ba”,

“data”:

“qmcqixfpceybopmzobllhmkfcvmcyzpzovzfvpanosxfqrmpxnspwbgq  
 vjcuohksznpyjizajfvkqdqlstvkxbentvmijuwjbfhsanszbbxnoxabxvlc hqbptfomdqd”

}

Listing . Primer test poruke dužine 128B

Merenje brzine protoka poruka je trivijalno zahvaljujući postavljenoj arhitekturi. Pošto su obe uloge u komunikaciji (publisher i suscriber) jedna te ista aplikacija, one dele memoriju. Dovoljno je u jednu strukturu podataka u memoriji zabeležiti po id-u vremenski trenutak slanja poruke. Po prijemu poruke, proverava se u istoj toj strukturi kada je poruka poslata. Vremenska razlika između momenta prijema poruke (trenutak kada poruka pristigne) i momenta zabeleženog u strukturi daje brzinu protoka poruke od publishera do subscribera.

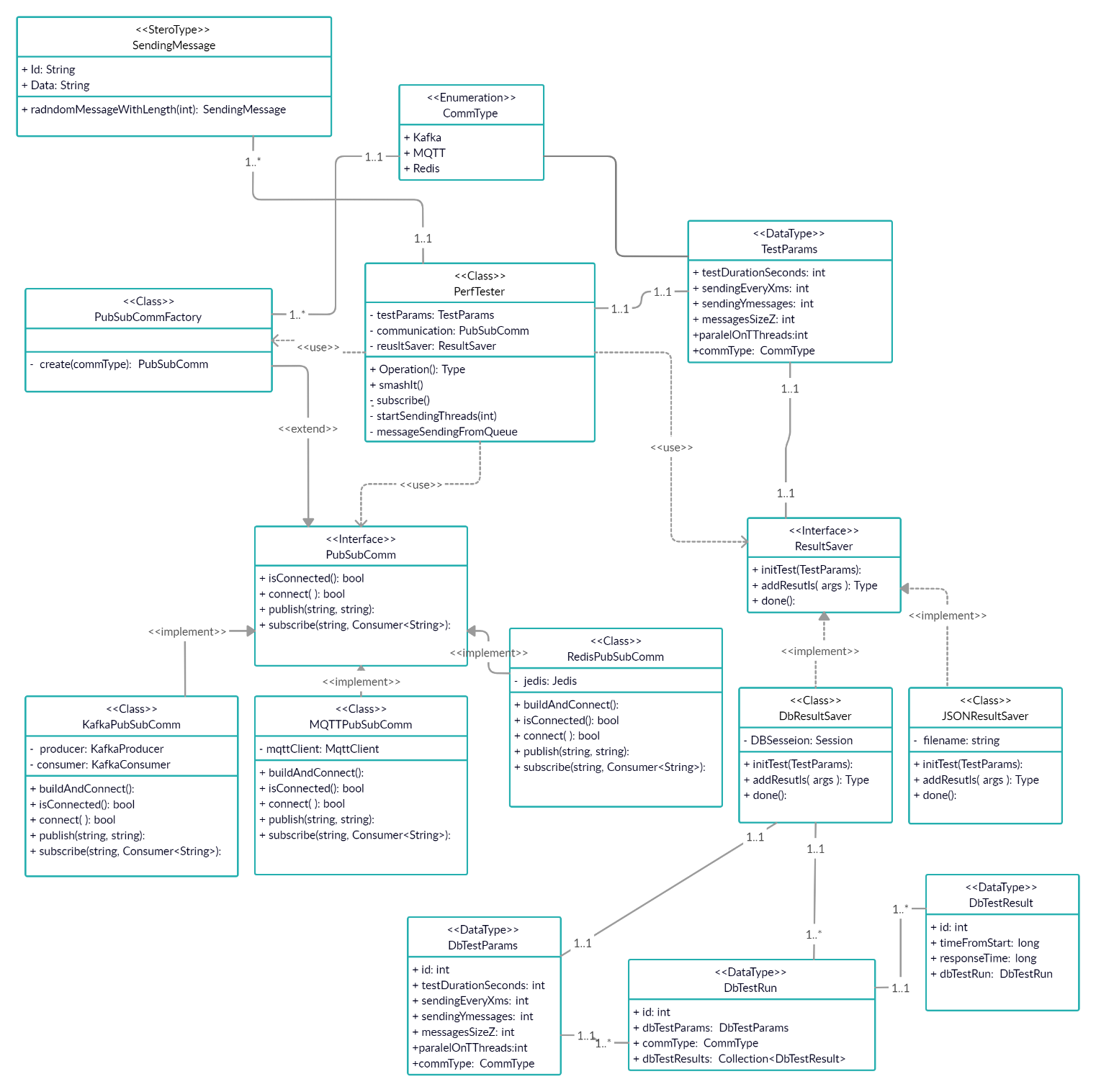
Prethodni pasus opisuje implementaciju uloge subscribera ovog testa. Subscriber se pokreće po prijemu poruke i radi sledeće

* Deserializuj poruku
* Proveri po id kada je poruka poslata
* Oduzmi trenutno vreme od vremena slanja
* Evidentiraj izračunatu razliku na *ResultSaver*

Očigledno je da u vreme potrebno za tok poruke od publishera do subscribera ulaze i neke dodatne radnje. Vreme se povećava za, na primer deserializaciju(kod prijema poruke), vreme potrebno za dodavanje u *queue* i uzimanje iz *queue*-a, i slično. Ova dodatna vremena su prisutna kod sve 2 vrste brokera, pa se mogu posmatrati kao konstante (nisu fiksne, ali se mogu razlikovati minimalno) koje ne utiču značajno na merenje u krajnjem poretku.

## Dijagram klasa

Na dijagramu 4.1 je predstavljen dijagram klasa. Gornja polovina dijagrama predstavlja, prethodno opisan, viši nivo apstrakcije testiranja. On je koncentrisan na klasu koja izvršava sam test: *PerfTester*. On koristi 2 glavne pomoćne klase *SendingMessage*, reprezent poruke koja se šalje na broker, a primer serializovane u JSON format je dat na listing 4.2, i *TestParams* koji predstavlja objekat grupisanja ulaznih parametara testa. Osim navedenih, sadrži i zavisnosti po interfejsima *PubSubComm* i *ResultSaver-a*, na koje se oslanja prilikom samog vršenja testa.



Dijagram . Dijagram klasa test aplikacije za testiranje brokera

Na levoj strani dijagrama je uočljiv dizajn patern *Factory method* gde se jedna metoda koristi kreiranje instance. U konkretnom slučaju, ovaj dizajn patern je iskorišćen za kreiranje komunikacije po *PubSubComm* intefejsu u zavisnosti od odabranog tipa brokera, enum parametar tipa *CommType*. U nižem centralnom delu se nalaze impelemntacije interfejsa *PubSubComm,* za svaki tip komunikacije jedna implementacija, kao i *ResultSavera* implementacije čuvanja rezultata u JSON fajl i u relacionu bazu podataka.

Skroz dole (desna polovina), 3 klase predstavljaju entitete koje *DbResultSaver* mapira na tabele u bazi podataka, te ih pod istima čuva. Isečak sa klas dijagram posvećen analizi konkretno tog dela se nalazi na dijagramu 4.1 u okviru ovog poglavlja.

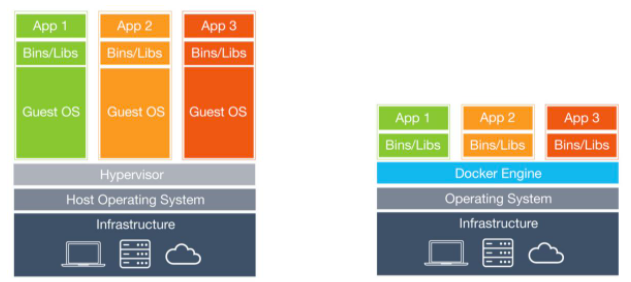
## Pomoćni alat

Osim brokera i aplikacije za testiranje korišćeni su još neki pomoćni alati raznih svrha u cilju olakšavanja testiranja. Koristeći pomoćne alate u smislu gotovih rešenja prepušta se više vremena fokusiranja na problematiku zadatka dok se za sporedne neophodne delove rešenja koriste već gotovi alati.

### Docker

*Docker* je softverska platforma za građenje aplikacija baziran na takozvanim kontejnerima. Kontejner je mala i minimalizovana slika operativnog sistema sa zasebnim okruženjem no ipak izolovanim od ostatka sistema.

Ideja kontejnerizacije je izgradnja softverske jedinice sa svojim okruženjem a maksimalno nezavisnim od ostatka sistema. U okviru okruženja, odnosno unutar jednog kontejnera, se nalazi aplikacija kao i sve potrebne zavisnosti. Na taj način se pokretnje aplikacije ograđuje od problema različitih okruženja i različitog hardvera. Preteča kontejnera je bila upotreba *Virtualnih mašina*. Na taj način se uspeo zadržati izolovan jedan deo sistema od drugog. Međutim, *virutalne mašine* su memorijski teške, svaka sadrži potpun operativni sistem te pokretanje zauzima mnogo nepotrebnih resursa. *Docker* takođe radi na principu *virtualne mašine* samo je operativni sistem iste sveden na minimalan da zadovoljava potrebe. Time se rešio problem teškoće mašine. Na slici 4.3 je ilustrovana razlika arhitekture između *virualne mašine* i *docker*-a.



Slika . Virtualne mašine i dokeri

Kontejneri se grade definisanjem i izgradnjom. Jedan *docker* kontejner se naziva *docker image,*odnosno docker slika. Definisanje načina gradnje docker slika se definiše *dockerfile*-om, što prestavlja tekstualni zapis po propisanom sintaksom. Docker slike se nadograđuju. Definicija unutar *dockerfile-*a uvek počinje navođenjem neke već postojeće docker slike. Na postojeću sliku se doda sloj neophodan konkretnom zahtevu i dobija se nova docker slika.

U ovom radu docker slike nisu građene. Iskorišćena je glavna prednost docker-a a to je jednostavno pokretanje gotovih alata. Na primer, svaki od navedenih brokera može da se instalira na radnu mašinu, konfiguriše i tako koristi. Ali, većina alata novijeg datuma, sadrži i mogućnost korišćenja samo pokretanjem dockera. Pa tako, sva 3 brokera koja se testiraju ovim radom imaju već gotove i pripremljene docker slike. To znači, da je dovoljno pokrenuti jednu komandu i broker je dostupan i spreman za upotrebu, bez instalacije, bez konfigurisanja bilo kakvog. Na primer, za pokretanje *mosquitto*-a kao MQTT servera bilo je potrebno izvršiti komandu navedenu u listingu 4.3

Listing . Pokretanje docker slike mosquito servera

docker run -it

-p 1883:1883

-p 9001:9001

-v %cd%\ dockers\mosquittoConfig:/mosquitto/config

eclipse-mosquitto

Ključno iz komande sa listinga 4.3. su **docker run** što označava da se pokreće docker kontainer kao i **eclipse-mosqutto** u smislu identifikacije docker slike koja se pokreće. Po pokretanju komande docker sistem proveri da li ima već skinutu navedenu sliku, po prvom pokrenju nema i potrebno je da je skine. *Docker hub* predstavlja internet biblioteku *docker* slika. Docker sistem sa *docker hub-*a preuzme traženu sliku te je pokreće.

### MySql

MySql je dobro poznat servis relacione baze podataka. U ovoj aplikaciji se koristi za skladištenje rezultata testova. Pogodan je za skladištenje podataka jer nudi naprednu pretragu koristeći SQL sintaksu.

Ovaj rad koristi MySql zbog mogućnosti skladištenja podataka, naprednog pretraživanja kao i mogućnost integracije sa Grafanom.

### Grafana



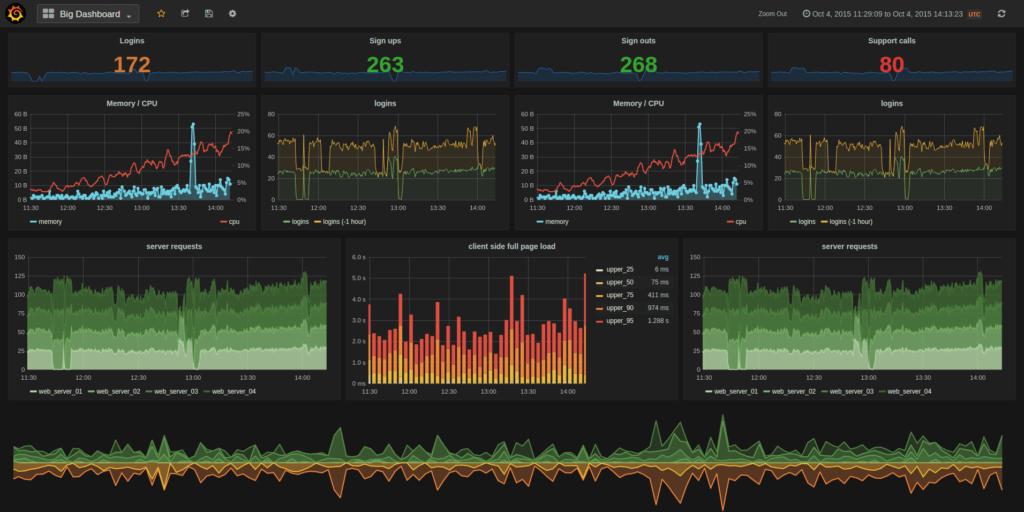
Slika . Grafana logo

Grafana predstavlja web servis za vizualizaciju podataka. Sadrži napredne mogućnosti konfigurisanja i crtanja grafika baziranih na podacima.

Sadrži integraciju sa mnogim tipovima skladištenja podataka. Između ostalog, sadrži integraciju sa MySql serverom. To znači, da Grafana ume da prikaže grafički podatke koji se nalaze u MySql bazama podataka. Pa tako, rezultati sačuvani tokom izvršavanja testova mogu biti prikazani ovim alatom.

Grafana najčešće prikazuje podatke u formi vremenskih serija, odnosno kretanje vrednosti kroz vreme. Podaci se iscrtavaju u koordinatnom sistemu gde je x osa najčešće vreme, a y osa predstavlja vrednost u zadatom trenutku. Pored prikaza vremenskih serija, grafana podržava i mnoge druge načine vizualizacije uz mnogo naprednije funkcije.

*Dashboard* predstavlja vid stranice koja prikazuje skup grafikona. Na dashboardu je moguće nacrtati i iskonfigurisati proizvoljno grafikona različitih vrsta. Cilj dashboarda je da smisleno vizualno predstavi podatke. Na lici 4.5 je dat proizvoljan primer jednog dashboard-a



Slika . Grafana primer dashboarda

Grafana dashboard je korišćen u ovom radu kao alat za prikaz rezultata testova. Naime, aplikacija za testiranje brokera se bavi samim merenjem ponašanja broker i kao rezultata daje rezultate testova. Rečeno je da se ti rezultati, između ostalog, čuvaju u MySql bazi podataka. Grafana se podešava da gađa bazu podataka gde su rezultati testova, te se iscrtan jedan dashboard namenjan za vizualizaciju eksperimentalnog dela ovog rada.

Poglavlje 3

# Zaključak

Ovo nista ne valja

# Literatura

1. Robert Sabo i mentor dr Srđan Škrbić,   
   Skalabilan mikroservis orijentisan sistem namenjen berzi kriptovaluta,   
   Prirodno-matematički fakultet, Univerzitete u Novom Sadu, 2018.  
   Pristupljeno: 22.08.2020. Dostupno na:  
   <https://github.com/robertsabo0/AWS_CryptOffer/blob/master/DiplomskiRad_RobertSabo_v2.4.pdf>
2. Geoffrey Winn i Neil G. Young,  
   Message oriented middleware with integrated rules engine  
   Sjedinjene Američke države, 07. mart 2019  
   Pistupljeno: 08.09.2020. Dostupno na:  
   <https://patentimages.storage.googleapis.com/04/57/f1/e30873a39f15d8/US10592312.pdf>
3. Patrick Th, Eugster, Pascal A. Felber, Rachid Geurraoui i Anne’Marie Kermarrec  
   The many faces of Publish/Subscribe  
   Švajcarski savezni tehnološki institut, 2003  
   Pristupljeno: 09.09.2020. Dostupno na:  
   <http://128.232.0.20/research/srg/netos/papers/2007_YONEKI_MSWIM.pdf>
4. Vaquar Khan,  
   Difference between scaling horizontally and vertically  
   github.com 30.01.2017,  
   Pristupljeno: 09.09.2020. Dosupno na:  
   <https://github.com/vaquarkhan/vaquarkhan/wiki/Difference-between-scaling-horizontally-and-vertically>
5. Shweta Khare, Hongyang Sun,, Kaiwen Zhang, Julien Gascon-Samson i Aniruddha Gokhale  
   Ensuring Low-Latency and Scalable Data Dissemination for Smart-City Applications  
   Univezitet u Britanskoj Kolumbiji, Vankuver, 20.04.2018  
   Pristupljeno: 09.09.2020. Dostupno na:   
   <http://www.dre.vanderbilt.edu/~gokhale/WWW/papers/IoTDI18_Poster.pdf>
6. Sefi Itzkovich  
   Reids, Kafka or RabbitMQ: Which MicroSerivces Message Broker to choose?  
   otonomo 29.05.2019  
   Pristupljeno: 09.09.2020. Dostupno na:  
   <https://otonomo.io/blog/redis-kafka-or-rabbitmq-which-microservices-message-broker-to-choose/>
7. Margarete Rouse  
   MQTT (MQ Telemetry Transport)  
   internetofthingsagenda.techtarget.com, januar 2020  
   Prisutpljeno: 14.09.2020. Dostupno na:   
   <https://internetofthingsagenda.techtarget.com/definition/MQTT-MQ-Telemetry-Transport>
8. Paho projekat,  
   Eclipse foundation  
   Pristupljeno: 17.09.2020, dostupno na:  
   <https://www.eclipse.org/paho/>
9. Mosquitto  
   Eclipse foundation  
   Pristupljeno: 18.09.2020, dostupno na:  
   <https://mosquitto.org/>
10. Tiago Macedo i Fred Oliveira  
    Redis Cookbook,  
    O’reilly, Avgust 2011
11. Redis specifikacija klastera  
    redis.io  
    Pristupljeno: 18.09.2020, dostupno na:  
    <https://redis.io/topics/cluster-spec>
12. Neha Nerkhede, Gwen SHapira i Todd Palino  
    Kafka, Vodič definicije  
    O’reilly, Jul 2017
13. The Apache Software Foundation  
    Pristpuljeno: 18.09.2020, dostupno na:  
    <https://www.apache.org/>
14. Java dokumentacija, „The try-with-resources Statement“  
    Oracle, 2020.  
    Pristupljeno: 21.09.2020. dostupno na:  
    <https://docs.oracle.com/javase/tutorial/essential/exceptions/tryResourceClose.html>
15. Apache Maven  
    Apache, septembar 2020.  
    Pristupljeno 21.09.2020. dostupno na:  
    <https://maven.apache.org/>
16. Redis   
    redis.io  
    Pristpuljeno 21.09.2020. dostupno na:  
    <https://redis.io/>

# Biografija

Robert Sabo

Doža Đerđa 4/40

Novi Sad, 21000

email: robert.sabo0@gmail.com

Kontakt: +381631164045

Zanimanje: student master studija– Prirodno-matematički fakultet u Novom Sadu; departman za matematiku i informatiku

Robert Sabo, rođen je 22. Septembra 1995. godine u Somboru, Srbija. Osnovnu školu “Petar Kočić” u Riđici završio je 2010. godine, iste godine upisuje Srednju ekonomsku školu u Somboru. Srednju školu uspešno završava 2014. godine. Nakon završetka odlučuje da se upiše na akademske studije. 2014. godine upisuje Prirodno-matematički fakultet Univerziteta u Novom Sadu, na departmanu za matematiku i informatiku. Po diplomiranju 2018. godine upisuje master studije na istom departmanu.

**UNIVERZITET U NOVOM SADU**

**PRIRODNO MATEMATIČKI FAKULTET**

**DEPARTMAN ZA MATEMATIKU I INFORMATIKU**

|  |
| --- |
| **KLJUČNA DOKUMENTACIJA** |

Redni broj:

**RBR**

Indentifikacioni broj:

**IBR**

Tip dokumentacije: Monografska dokumentacija

**TD**

Tip zapisa: Tekstualni

**TZ** štampani materijal

Vrsta rada: Master rad

**VR**

Autor: Robert Sabo

**AU**

Mentor: dr Danijela Boberić Krstićev

**MN**

Naslov rada: Komunikacija u distribuiranim sistemima orijentisana porukama

**NS**

Jezik publikacije: Srpski/latinica

**JP**

Jezik izvoda: Srpski

**JI**

Zemlja publikacije: Srbija

**ZP**

Uže geografsko područje: Vojvodina

**UGP**

Godina: 2020

**GО**

Izdavač: Autorski reprint

**IZ**

Mesto i adresa: Prirodno – matematički fakultet, Trg

**MA** Dositeja Obradovića 4, Novi Sad

Fizički opis rada: 8 poglavlja, 79 strana, TODO….!

**FO** 17 slika, 26listinga

Naučna oblast: Informatika

**NO**

Naučna disciplina: Distribuirani sistemi

**ND**

Predmetna odrednica/ Kafka, Redis, MQTT

**PO**

Ključne reči: Komunikacija, brokeri poruka, MQTT, Kafka, Redis

**UDK:**

Čuva se: Biblioteka Departmana za matematiku

**ČU** i informatiku, Novi Sad

Važna napomena: Nema

Izvod Opis skalabinlog mikroservis orijentisanog sistema namenjenog berzi kriptovaluta

**IZ**:

Datum prihvatanja teme:

**DТ**

Datum odbrane:

**DO**

Članovi komisiji:

**1.** dr Miloš Savić, vanredni profesor PMF-a, predsednik

**2.** dr Danijela Boberić Krstićev, vanredni profesor PMF-a, mentor

**3.** dr Jovana Vidaković, vanredni profesor PMF-a, član

**UNIVERSITY OF NOVI SAD**

**FACULTY OF SCIENCES**

**DEPARTMENT OF MATHEMATICS AND INFORMATICS**

|  |
| --- |
| **KЕY WORDS DOCUMENTATION** |

Accession number:

**ANO**

Identification number:

**INO**

Document type: Monograph type

**DT**

Type of record: Printed text

**TR**

Contents code: Master thesis

**CC**

Autor: Robert Sabo

**AU**

Mentor: PhD Danijela Boberić Krtićev

**MN**

Title:Message oriented communication

In distributed systems

**XI**

Language of text: Serbian/Latinica

**LT**

Language of abstract: Serbian

**LA**

Country of publication: Serbia

**CP**

Locality of publication: Vojvodina

**LP**

Publication year: 2020.

**PY**

Publisher: Author's reprint

**PU**

Publication place: PMF, Trg Dositeja

**PP** Obradovića 4, Novi Sad

Physical description: 8 chapters, 79 pages,

**PD** 17 photos, 26 codes ‘’’’’’ ‘TODO

Scientific field: Informatics

**SF**

Scientific discipline: Distributed systems

**SD**

Key words: Communication, message broker, Redis, Kafka MQTT,

**KW**

Holding data: Library of Department of mathematics

**SD** and informatics, Novi Sad

Note: None

Abstract: Message oriented communication in distributed systems

**AB**

Accepted by the Scientific Board on:

Defended on:

Thesys defend board:

**1.** PhD Miloš Savić, associate professor, Faculty of Sciences, Novi Sad, president

**2.** PhD Danijela Boberić Krstićev, associate professor, Faculty of Sciences, Novi Sad, menthor

**3.** PhD Jovana Vidaković, associate professor, Faculty of Sciences, Novi Sad, member

1. Listing 3.2 je korigovan da bude identičan sa slikom 3.10. Realan ispis je nula indeksiran. Umesto brokera 1,2,3 i 4 realno piše 0,1,2 i 3. Ova korekcija je urađena radi izbegavanje konfuzije prilagođavanjem slici 3.10. [↑](#footnote-ref-1)