

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 52

**ANALIZA DNEVNIČKIH ZAPISA RASPODIJELJENIH
SUSTAVA U STVARNOM VREMENU**

Matej Čubek

Zagreb, lipanj 2023.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 52

**ANALIZA DNEVNIČKIH ZAPISA RASPODIJELJENIH
SUSTAVA U STVARNOM VREMENU**

Matej Čubek

Zagreb, lipanj 2023.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 10. ožujka 2023.

DIPLOMSKI ZADATAK br. 52

Pristupnik: **Matej Čubek (0036516398)**

Studij: Računarstvo

Profil: Programsко инженерство и информациони системи

Mentor: prof. dr. sc. Damir Pintar

Zadatak: **Analiza dnevničkih zapisa raspodijeljenih sustava u stvarnom vremenu**

Opis zadatka:

Složeni informacijski sustavi obično zahtijevaju kvalitetnu podršku u vidu učinkovitog nadzora i održavanja visoke razine pouzdanosti. U raspodijeljenim okruženjima, u kojima aplikacije i servisi rade na više različitim čvorova i instanci, dnevnički zapisi postaju ključni izvor informacija za praćenje stanja sustava i detekciju problema u stvarnom vremenu. Analiza dnevničkih zapisa omogućuje otkrivanje anomalija, negativnih učinaka na performanse i drugih problema koji mogu utjecati na kvalitetu usluge za krajnje korisnike. Zadatak diplomskega rada je istražiti mogućnosti Apache Kafka platforme za potrebe obrade dnevničkih zapisa u stvarnom vremenu te njezine primjene za detekciju i nadgledanje anomalija u raspodijeljenim i skalabilnim sustavima. Potrebno je proučiti i obrazložiti prednosti i nedostatke ove tehnologije te na studijskom primjeru provesti eksperiment koji će pokazati učinkovitost korištenja Kafka platforme u kombinaciji s drugim alatima za nadzor i analizu podataka u stvarnom vremenu. Također je potrebno napraviti osvrt na mogućnosti povećanja pouzdanosti i brzine obrade podataka te poboljšanja sposobnosti sustava za reakciju na probleme u stvarnom vremenu.

Rok za predaju rada: 23. lipnja 2023.

SADRŽAJ

Indeks slika	vi
Indeks tablica	vii
1. Uvod	1
2. Arhitektura aplikacija	3
2.1. Monolitne aplikacije	3
2.2. Distribuirane aplikacije	4
2.2.1. Mikro-servisne aplikacije	5
2.3. Pokretanje direktno na hardveru računala	5
2.4. Virtualni strojevi	6
2.5. Kontejnerizacija i Docker	8
2.6. Računarstvo u Oblaku	9
2.6.1. Grozdovi računala	9
2.6.2. Lambda funkcije	10
3. Distribuirana aplikacija — CloudVane	12
3.1. Orkestrator kontejnera — Kubernetes	15
3.1.1. Implementacija aplikacije CloudVane na Kubernetesu	15
3.1.2. Komponente sustava Kubernetes	16
3.1.3. Minikube	18
3.2. Sustav za obradu podataka u stvarnom vremenu — Apache Kafka	19
3.2.1. Komponente sustava Apache Kafka	20
3.2.2. Operator za Apache Kafku na Kubernetesu — Strimzi	21
3.2.3. Apache Zookeeper	22
3.2.4. Kafka Streams	23
3.2.5. ksqlDB	27
3.2.6. Kafka Connect	27

3.2.7. Avro serializacija i Schema Registry	29
3.2.8. Kafka UI	29
3.3. Arhitektura aplikacije CloudVane	30
3.4. CloudVane kao izvor podataka	32
4. Nadziranje aplikacija	33
4.1. Dnevnički zapisi	34
4.1.1. Dnevnički zapisi u programskom jeziku Java	35
4.2. Elastic, Logstash i Kibana (ELK)	35
4.2.1. ElasticSearch	35
4.2.2. Logstash	36
4.2.3. Kibana	37
4.2.4. Filebeat	37
4.2.5. Implementacija ELK-a na Kubernetes grozdu	39
5. Nadziranje stanja distribuiranog sustava u stvarnom vremenu	40
5.1. Pregled arhitekture rješenja	41
5.2. Kubernetes infrastruktura rješenja	44
5.3. Pohrana dnevničkih zapisa u Apache Kafku	45
5.4. Agregiranje dnevničkih zapisa	47
5.5. Obogaćivanje dnevničkih zapisa	50
5.5.1. OpenAI API	50
5.6. Notifikacije u stvarnom vremenu	51
5.7. Pregled kroz Kibanu	55
6. Zaključak	57
Literatura	58
A. Isječci programskog koda	60

INDEKS SLIKA

2.1. Usporedba monolitne arhitekture s mikro-servisnom arhitekturom	6
2.2. Usporedba pokretanja direktno na hardveru, virtualnim strojevima i kontejnerima	7
3.1. Arhitektura CloudVane sustava	13
3.2. Tok podataka CloudVane sustava	14
3.3. Prozori agregacija u Kafka Streams-u i ksqlDB-u	25
3.4. Hijerarhija Kafka servisa za obradu tokova podataka	26
4.1. Kibana sučelje	38
5.1. Arhitektura sustava	41
5.2. Tok podataka sustava	42
5.3. Kafka teme s dnevničkim zapisima	43
5.4. Kafka teme na lokalnom Apache Kafka grozdu kroz pogled Kafka UI servisa	46
5.5. Topologija Kafka Streams aplikacije	49
5.6. Primjer notifikacije dnevničkog zapisa greške kroz elektroničku poštu	52
5.7. Primjer notifikacije dnevničkog zapisa greške putem Slack aplikacije .	53
5.8. Primjer notifikacije dnevničkog zapisa greške putem Pushover mobilne aplikacije	54
5.9. Prikaz filtriranih dnevničkih zapisu u Kibana sučelju	55
5.10. Prikaz kreirane kontrolne ploče u Kibani	56

INDEKS TABLICA

3.1. Vrste prozora u Kafka Streams biblioteci	25
3.2. CloudVane servisi na Kubernetesu	31
3.3. Kafka servisi na CloudVane Kubernetesu	31
5.1. Kafka teme s dnevničkim zapisima	47

1. Uvod

Informacijski sustavi s vremenom postaju složeniji te su samim time izazovniji u pogledu održavanja i nadzora. U svijetu složenih informatičkih sustava dnevnički zapisi imaju ključnu ulogu u praćenju i održavanju kvalitete svih aplikacija. Međutim, u okruženju raspodijeljenih sustava, gdje su aplikacije pokrenute na različitim čvorovima računalnih grozdova ili računalima, pregledavanje dnevničkih zapisa svih servisa postaje iznimno zahtjevan zadatak. Ova teza, pod nazivom „*Analiza dnevničkih zapisa raspodijeljenih sustava u stvarnom vremenu*“, proučava upotrebu *Apache Kafka* platforme, *ELK* platforme i *OpenAI API-ja* za obradu i analizu dnevničkih zapisa raspodijeljenih sustava u stvarnom vremenu.

Nadzor nad dnevničkim zapisima omogućuje otkrivanje anomalija, detektiranje negativnih učinaka na performanse i drugih problema koji mogu utjecati na kvalitetu usluge. Cilj je ove teze istražiti mogućnosti *Kafka* platforme za analizu dnevničkih zapisa, s posebnim naglaskom na detekciju i nadzor anomalija u raspodijeljenim sustavima.

Ovaj rad proučavat će *Apache Kafka* i *ELK* platforme istražujući kako se mogu koristiti za obradu i analizu dnevničkih zapisa u stvarnom vremenu. Analiza će se napraviti na stvarnom primjeru SaaS-a¹ *CloudVane* tvrtke *Neos* te će se upravo na toj distribuiranoj mikro-servisnoj aplikaciji pokazati iskoristivost rješenja rada.

Također, rad će razmatrati kako se korištenim alatima mogu poboljšati pouzdanost i brzina obrade podataka kako bi se pravovremeno moglo reagirati na probleme sustava. Očekuje se da će ovaj rad pružiti dublje razumijevanje važnosti i mogućnosti analize dnevničkih zapisa u raspodijeljenim sustavima, kao i uloge *Kafka* platforme u tom kontekstu.

Poglavlje 2 početak je ovog rada i pruža uvid u arhitekture aplikacija, uz opis monolitnih i distribuiranih aplikacija, kao i metoda njihovih implementiranja. Na-

¹Softver kao usluga (engl. *Software as a Service*)

dalje, poglavlje 3 posvećeno je CloudVane aplikaciji, njenim komponentama kao što su Apache Kafka i Kubernetes. U poglavlju 4 izloženo je kako se vrši nadzor putem dnevničkih zapisa i objašnjena je upotreba ELK platforme. Poglavlje 5 opisuje proces obrade dnevničkih zapisa u stvarnom vremenu. Završno, poglavlje 6 donosi rezime i osvrt na sve što je istraženo i predstavljeno u ovom radu.

2. Arhitektura aplikacija

Prvobitno su se računalne aplikacije pokretale na izoliranim serverima, svaki posvećen isključivo posluživanju jedne aplikacije. Ova je arhitektura omogućavala da svaka aplikacija ima svoje računalo na kojem radi, često smješteno u posebnoj server-sali ili odvojenoj sobi. Ovaj pristup bio je jednostavan i lako razumljiv, te je omogućavao izravno mapiranje između aplikacija i odgovarajućih resursa.

No, ovaj model nije bio bez ograničenja. Prvo, to je stvorilo nedostatak fleksibilnosti u pogledu alokacije resursa jer su se resursi često prekomjerno koristili za jednu aplikaciju, dok su ostale aplikacije patile od nedostatka resursa. Budući da svaka aplikacija zahtijeva svoj vlastiti fizički server, ovaj pristup bio je neučinkovit u pogledu prostora i energije.

Uz to, održavanje takvih izoliranih sustava predstavljalo je izazov. Svaki server morao se održavati odvojeno, što je uključivalo nadogradnje hardvera, instalacije softvera, popravke i slično. Također, ovaj model nije pružao adekvatnu otpornost na kvarove — ako bi server prestao raditi, cijela bi aplikacija bila nedostupna sve dok se server ne popravi ili zamjeni, što može rezultirati značajnim prekidima u radu. Stoga su se s vremenom razvile nove arhitekture, poput distribuiranih sustava i virtualizacije, koje su omogućile fleksibilniju, učinkovitiju i otporniju upotrebu računalnih resursa.

U ovom će se poglavlju razmotriti različite arhitekture aplikacija, uključujući distribuirane i monolitne sustave te različite metode njihove implementacije.

2.1. Monolitne aplikacije

Monolitne aplikacije karakterizira integracija svih funkcionalnosti unutar jedinstvene operativne jedinice (Harris, 2023). U ovakvom sustavu sve komponente, od korisničkog sučelja do pozadinskog sustava i baze podataka, tvore koherentnu cjelinu iste aplikacije.

Glavna prednost monolitne arhitekture leži u njenom centraliziranom pristupu koji pojednostavljuje interakcije među komponentama eliminirajući potrebu za kompleksnom komunikacijom koja je česta kod distribuiranih aplikacija. Osim toga, monolitne aplikacije često su jednostavnije za testiranje i održavanje jer se ne zahtijeva upravljanje mnoštvom odvojenih servisa.

Ipak, monolitne aplikacije imaju nedostatke. Jedan je od ključnih izazova skaliranje: kako raste opterećenje na aplikaciju, monolitne aplikacije omogućavaju samo vertikalno skaliranje, što znači povećanje resursa na postojećem serveru. Horizontalno skaliranje, koje podrazumijeva dodavanje dodatnih servera, nije moguće. Također, ažuriranje ili mijenjanje bilo kojeg segmenta aplikacije često zahtijeva ponovnu implementaciju (engl. *deployment*) cijelog sustava, što može dovesti do privremene nedostupnosti usluge.

Unatoč ovim izazovima, monolitne aplikacije zadržavaju značajnu prisutnost u računalnoj industriji. U mnogim ih slučajevima njihova jednostavnost, kombinirana s potencijalnom uštem u mrežnom prometu, čini atraktivnom opcijom za manje složene sustave ili sustave s konstantnim opterećenjem.

2.2. Distribuirane aplikacije

Distribuirane ili raspodijeljene aplikacije predstavljaju temelj suvremenih informacijskih sustava i usluga koje se svakodnevno koriste. Ove aplikacije koriste se u različitim industrijama i sektorima, od financija, zdravstva, obrazovanja, pa sve do zabave i medija. Njihova je osnovna značajka sposobnost paralelne obrade podataka na više računalnih čvorova i geografskih lokacija, čime se omogućuje brza i pouzdana usluga bez obzira na fluktuacije u potražnji. Dodatno, distribuirani sustav je skup neovisnih računala koji korisniku izgleda kao jedan cjeloviti sustav (Tanenbaum i Steen, 2007).

Pokretanje aplikacija na različitim računalnim čvorovima znači da instance aplikacije mogu podijeliti opterećenje povećavajući tako učinkovitost i performanse sustava. Ova decentralizacija također znači da kvar jednog čvora neće onemogućiti rad cijele aplikacije. Umjesto toga, rad će biti preusmjeren na ostale čvorove osiguravajući tako kontinuitet usluge. Upravo je ta visoka dostupnost, robusnost i brzina odziva ključna prednost distribuiranih aplikacija. Uz to, fleksibilnost skaliranja omogućuje im da se prilagode promjenjivim uvjetima potražnje, dodavanjem ili uklanjanjem računalnih čvorova prema potrebi.

No, unatoč brojnim prednostima, distribuirane aplikacije donose i određene izazove. Žarko et al. (2013) navode kako složena pitanja poput koordinacije i sinkronizacije među čvorovima, očuvanje konzistencije podataka kroz različite komponente sustava te upravljanje povećanim složenostima mreže zahtijevaju sofisticirane strategije i tehnike. Algoritmi i protokoli za rješavanje ovih problema moraju biti pažljivo osmišljeni kako bi se osigurao pravilan rad distribuiranog sustava.

Unatoč ovim izazovima, distribuirane aplikacije ostaju ključan alat za postizanje visoke efikasnosti, pouzdanosti i fleksibilnosti u današnjem digitalno povezanom svijetu. Budući da tehnologija napreduje, distribuirane aplikacije postaju sve sofisticirane, pružajući sve naprednija rješenja za suvremene poslovne i društvene izazove.

2.2.1. Mikro-servisne aplikacije

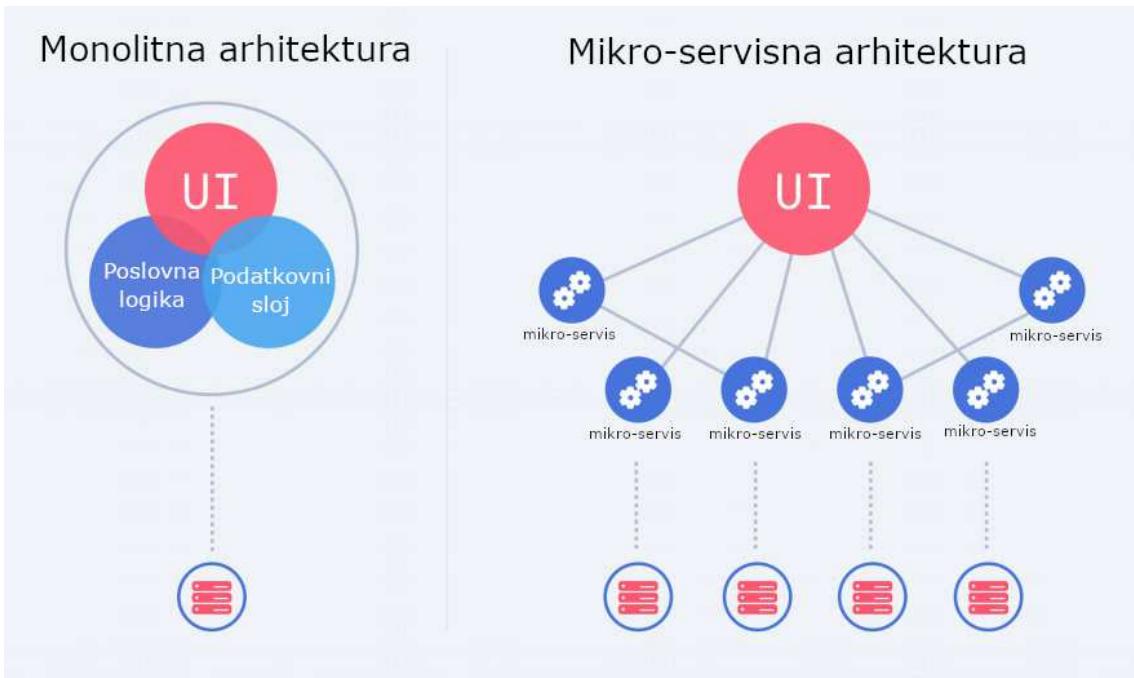
Aplikaciju **mikro-servisne** arhitekture (engl. *microservice architecture*) čini kolekcija malih autonomnih servisa. Svaki servis ima svoje funkcionalnosti, no uz međusobnu komunikaciju stvaraju kohezivnu cjelinu.

Na slici 2.1 može se vidjeti kako mikro-servisna arhitektura izgleda u usporedbi s monolitnom arhitekturom opisanom u poglavlju 2.1. Iako mikro-servisna arhitektura donosi izazove poput složenije komunikacije i održavanja sinkronizacije podataka, istovremeno pruža znatne prednosti, kao što su pojednostavljen horizontalno skaliranje i smanjeni rizik od kvarova sustava zahvaljujući autonomiji servisa. Radi efikasnosti u radu s mikro-servisima često se koristi kontejnerizacija, opisana u poglavlju 2.5, i orkestracijski alati poput Kubernetes-a opisanog u poglavlju 3.1. Upravo takav sustav koristi aplikacija CloudVane opisana u poglavlju 3 na čijim se dnevničkim zapisima bazira ovaj rad.

2.3. Pokretanje direktno na hardveru računala

Pokretanje na hardveru (engl. *bare metal*) znači da se aplikacije izvršavaju na namjenском računalu, bez upotrebe virtualizacije. Ovakva konfiguracija omogućuje korištenje svih raspoloživih resursa računala, poput procesorske snage, memorije i prostora za pohranu.

Primjena ovog pristupa donosi niz prednosti poput boljih performansi zbog izravnog pristupa hardveru, bez posredničkog sloja koji donosi dodatno opterećenje



Slika 2.1: Usporedba monolitne arhitekture s mikro-servisnom arhitekturom

Slika preuzeta sa https://dev.to/alex_barashkov/microservices-vs-monolith-architecture-411m

(engl. *overhead*). Dodatno, pokretanje na namjenskom hardveru može pružati i veću sigurnost jer se eliminiraju rizici dijeljenja resursa s drugim aplikacijama.

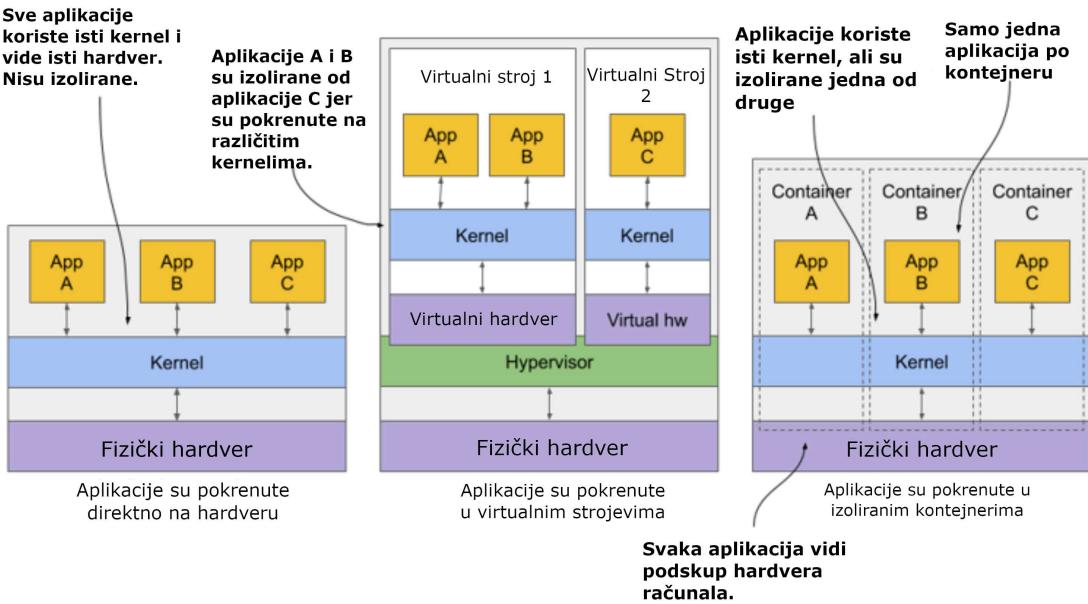
Upravljanje hardverom međutim može uzrokovati složenije i vremenski zahtjevnije procese, posebno kada je riječ o održavanju. Skaliranje aplikacija također predstavlja izazov jer zahtijeva nabavku novog hardvera i migraciju aplikacija na nove resurse.

Unatoč tim izazovima, aplikacije koje zahtijevaju visoko-intenzivne resurse mogu značajno profitirati od prednosti koje pruža ovaj pristup. Pokretanje aplikacija na samom hardveru usporedno s virtualnim strojevima i kontejnerima opisanim kasnije prikazano je na slici 2.2¹.

2.4. Virtualni strojevi

Virtualni strojevi (engl. *Virtual Machine*) su sofisticirani koncept koji simulira hardver kroz softver. Oni predstavljaju sloj apstrakcije između aplikacije i fizičkog hardvera na kojem se izvršava. Aplikacija se prividno izvršava na vlastitom, nezavisnom sustavu.

¹Kernel je glavni program operacijskog sustava, a Hypervisor je program koji upravlja virtualnim strojevima.



Slika 2.2: Usporedba pokretanja direktno na hardveru, virtualnim strojevima i kontejnerima
Slika preuzeta iz Lukša, M. (2023). *Kubernetes in Action, Second Edition.* (n.p.): Manning.

Korištenje virtualnih strojeva donosi niz značajnih prednosti. Prva među njima je mogućnost učinkovitijeg iskorištavanja računalnih resursa. Na jednom fizičkom računalu moguće je pokrenuti nekoliko virtualnih strojeva, svaki sa svojim operativnim sustavom i aplikacijama, čime se maksimizira iskorištavanje hardverskih resursa i smanjuje potreba za nabavom dodatnog hardvera.

Još jedan bitan aspekt korištenja virtualnih strojeva je izolacija. S obzirom na to da su virtualni strojevi nezavisne jedinice, moguće je pokrenuti različite aplikacije na različitim operativnim sustavima na istom fizičkom računalu. Ova izolacija smanjuje rizik od međusobnog utjecaja aplikacija i povećava sigurnost jer potencijalni problemi ili napadi na jedan virtualni stroj ne utječu na drugi.

Uz to, virtualni strojevi pružaju veliku fleksibilnost i skalabilnost. Dodavanje ili uklanjanje virtualnih strojeva, prema potrebama korisnika ili aplikacije, može se obaviti brzo i efikasno. Mogućnost preslikavanja (engl. *snapshot*) i kloniranja (engl. *clone*) virtualnih strojeva olakšava sigurnosne kopije (engl. *backup*) i oporavak sustava te omogućava jednostavno testiranje i razvoj.

Međutim, unatoč brojnim prednostima, korištenje virtualnih strojeva donosi i određeno dodatno opterećenje (engl. *overhead*). Svaki virtualni stroj zahtijeva određeni dio

računalnih resursa za rad operativnog sustava, što može rezultirati manjom efikasnošću u odnosu na pokretanje aplikacija direktno na hardveru. Upravljanje velikim brojem virtualnih strojeva također može biti složeno, ali postoje različiti alati i platforme za upravljanje virtualnim okruženjima koji mogu pomoći u tome. Usporedba pokretanja na virtualnim strojevima s pokretanjem direktno na hardveru i kontejnerima objašnjena u sljedećem poglavlju vidljiva je na slici 2.2.

2.5. Kontejnerizacija i Docker

Kontejnerizacija (engl. *containerization*) je proces pakiranja aplikacije zajedno s potrebnim okruženjem i zavisnostima (engl. *dependencies*), što omogućuje jednostavno izvršavanje i pokretanje aplikacije u različitim računalnim okruženjima (Red Hat, 2021). Kontejneri su izolirane jedinice koje sadrže sve što je potrebno za izvršavanje aplikacije uključujući programski kod, izvršno okruženje (engl. *runtime*), sistemske alate, knjižnice (engl. *libraries*) i postavke.

Docker je najpoznatiji alat za kontejnerizaciju koji je postao standard u industriji. S Dockerom programeri mogu stvoriti, pokrenuti i distribuirati aplikacije unutar kontejnera. Kontejneri stvoreni pomoću Dockera² mogu se pokrenuti bilo gdje je Docker instaliran bez obzira na osnovni operativni sustav. To znači da programeri ne moraju brinuti o kompatibilnosti operativnih sustava prilikom pokretanja svojih aplikacija.

Kontejnerizacija donosi brojne prednosti. Kontejneri su manji i brži od tradicionalnih virtualnih strojeva jer ne zahtijevaju cijeli operativni sustav. Osim toga, kontejneri osiguravaju konzistentnost okruženja kroz razne faze razvojnog ciklusa aplikacija (Gillis, 2020). Uz to, Docker omogućuje jednostavno skaliranje aplikacija dodavanjem ili uklanjanjem kontejnera po potrebi.

Kao rezultat, kontejnerizacija i Docker postali su ključni elementi modernih razvojnih ciklusa i *DevOps*³ praksi, pružajući ključnu ulogu u povećanju efikasnosti i brzine isporuke softvera.

²Kroz Docker je moguće pokrenuti kontejnere koje su kreirali različiti alati, pod uvjetom da implementiraju Open Container Initiative standard koji je postavila Linux Fundacija. Slično tome, kontejneri koji su kreirani korištenjem Dockera mogu se pokrenuti kroz bilo koje drugo izvršno okruženje kontejnera koje podržava Open Container Initiative. Ova interoperabilnost omogućava veću fleksibilnost i slobodu izbora prilikom rada s kontejnerima.

³DevOps je skup praksi, filozofija i alata koji poboljšavaju suradnju između razvoja softvera (engl. *Dev*) i IT operacija (engl. *Ops*) (Courtemanche et al., 2021).

Usporedba kontejnera, koji su zahvaljujući svojim prednostima postali temelj sustava poput *Kubernetes* iz poglavlja 3.1, s pokretanjem direktno na hardveru i virtualnim strojevima prikazana je na slici 2.2.

2.6. Računarstvo u Oblaku

Računarstvo u oblaku (engl. *Cloud computing*) koncept je pružanja IT⁴ usluga u kojem se resursi poput računalne snage, prostora za pohranu, aplikacije i drugi resursi pružaju kroz internet na zahtjev korisnika ovisno o njegovoj potrebi (Žarko et al., 2013). Oblak omogućuje korisnicima da ne moraju upravljati računalnim resursima, već njima upravlja pružatelj usluga oblaka.

Prednosti oblaka uključuju fleksibilnost, skalabilnost i troškovnu učinkovitost. Fleksibilnost dolazi od sposobnosti korisnika da brzo i jednostavno prilagode svoje resurse u skladu s potrebama. Skalabilnost se odnosi na mogućnost dodavanja ili uklanjanja resursa u skladu s promjenama u potražnji. Troškovnu učinkovitost omogućuje model u kojem korisnici plaćaju samo za resurse koje koriste.

Prema Žarko et al. (2013) postoje različiti modeli oblaka:

Javni oblik: pružatelj usluga resurse pruža svim korisnicima preko interneta.

Privatni oblik: infrastruktura posvećena samo jednoj organizaciji.

Hibridni oblik: mješavina javnog i privatnog oblaka, pruža organizacijama veću fleksibilnost prilikom upravljanja svojim resursima.

Korištenje usluga u obliku također omogućava pristup naprednim tehnologijama i alatima, poput orkestracije kontejnera, automatizacije, umjetne inteligencije i analize podataka, koje bi bile teške i skupe za implementaciju na vlastitoj infrastrukturi.

Osim toga, pružatelji usluga u obliku nude sofisticirane mehanizme sigurnosti, otpornosti na pogreške i oporavka podataka, čime dodatno povećavaju dostupnost i pouzdanost aplikacija koje se nalaze u obliku.

2.6.1. Grozdovi računala

Grozdovi računala ili klasteri (engl. *clusters*) predstavljaju skupinu povezanih računala koja rade zajedno na takav način da se u mnogim aspektima mogu smatrati

⁴Informatička Tehnologija (engl. *Information Technology*)

jedinstvenim sustavom (Žarko et al., 2013). Grozdovi računala omogućuju paralelnu obradu zadataka, visoku dostupnost usluga, ravnomjerno opterećenje i skalabilnost.

Jedna od ključnih prednosti grozdova računala je visoka dostupnost usluga zahvaljujući mogućoj distribuciji zadataka. Kao što je navedeno u poglavlju 2.2, u slučaju kvara jednog računala, ostala računala u klasteru mogu preuzeti njegove zadatke i osigurati neprekidno pružanje usluga. To je posebno korisno za aplikacije i usluge koje zahtijevaju kontinuiranu dostupnost.

Grozdovi računala, zahvaljujući distribuiranosti, također omogućuju paralelnu obradu podataka, gdje se kompleksni zadaci mogu podijeliti na manje dijelove koje svako računalo u klasteru obrađuje istovremeno. To može značajno smanjiti vrijeme potrebno za obradu velikih količina podataka.

Unatoč svim prednostima, upravljanje grozdovima računala može biti složeno i zahtijeva adekvatne alate i stručnost. Moraju se uspostaviti mehanizmi za koordinaciju između računala, ravnomjerno raspoređivanje opterećenja, otkrivanje i oporavak od kvarova te osiguravanje konzistentnosti podataka.

U kontekstu grozdova računala *računalni oblak* (engl. *cloud*) pruža ključnu infrastrukturu koja značajno olakšava upravljanje. Koristeći oblak, organizacije se mogu fokusirati na razvoj i izvršavanje svojih aplikacija bez potrebe za postavljanjem i održavanjem vlastite hardverske infrastrukture.

Pružatelji usluga u oblaku, poput *Amazon Web Services (AWS)*, *Google Cloud Platform (GCP)*, *Microsoft Azure* ili *Oracle Cloud Infrastructure (OCI)*, nude usluge za upravljanje grozdovima računala koje olakšavaju postavljanje, skaliranje i održavanje. Ti se grozdovi mogu lako prilagoditi potrebama korisnika, bilo da je riječ o povećanju ili smanjenju broja računala u grozdu.

2.6.2. Lambda funkcije

Lambda funkcije, koje su popularizirane kroz računalni oblak **AWS**⁵, predstavljaju ključni dio koncepta **bez-servernog** (engl. *serverless*) računarstva. Naziv potječe od koncepta da se programeri ne trebaju brinuti o infrastrukturi na kojoj se njihova usluga pokreće (CloudFlare, 2023).

Lambda funkcije esencijalno su blokovi koda koji su dizajnirani za obavljanje specifične zadaće i koji se mogu pokrenuti na zahtjev. One se izvršavaju samo kada su potrebne, što znači da korisnici plaćaju samo za vrijeme izvršavanja koda, a ne za

⁵Amazon Web Services

stalno prisutne poslužitelje.

Budući da AWS automatski dodjeljuje potrebne resurse za izvršavanje funkcije, bez obzira na to je li potrebno pokrenuti jednu instancu ili tisuće instanci, ove funkcije omogućuju iznimnu skalabilnost. Lambda funkcije mogu se koristiti za različite zadatke, uključujući obradu podataka u stvarnom vremenu, automatizaciju zadataka, pa čak i izgradnju cijelih mikro-servisnih aplikacija.

Unatoč prednostima koje nude mikro-servisi i bez-serverna (engl. *serverless*) arhitektura kao što su fleksibilnost i mogućnost neovisnog skaliranja komponenti, studija provedena u sklopu članka Kolny (2023) pokazuje da ove prednosti ne dolaze bez troškova. U slučaju *Amazon Prime Video* originalna se arhitektura usluge za nadzor audio/video sadržaja temeljila na distribuiranom modelu mikro-servisa, koji je uključivao AWS Lambda funkcije. Međutim, ova arhitektura pokazala se skupom i s ograničenjima skaliranja.

Kako bi riješio ove probleme, *Amazon Prime Video* je rekonstruirao uslugu u monolitnu aplikaciju, što je rezultiralo značajnim smanjenjem infrastrukturnih troškova — čak 90%. Ova promjena eliminirala je potrebu za skupim prijenosima podataka između distribuiranih komponenti i pojednostavila logiku orkestracije.

Kolny (2023) naglašava važnost pažljivog razmatranja pri odabiru arhitekture. Govori kako mikro-servisi i Lambda funkcije mogu pružiti visoku razinu fleksibilnosti, mogu također donijeti značajne troškove i ograničenja skaliranja. Stoga u nekim slučajevima, kao što je ovaj, monolitna arhitektura može biti isplativija opcija.

3. Distribuirana aplikacija — CloudVane

CloudVane je proizvod koji pomaže upravljati i optimizirati trošak te infrastrukturu u *više-oblačnom* (engl. *multicloud*) okruženju (Neos, 2023). Glavni je cilj CloudVane aplikacije pružiti vidljivost i kontrolu nad upotrebom oblaka (engl. *clouda*) u cijeloj organizaciji.

CloudVane je izgrađen na **FinOps**¹ principima koji se fokusiraju na razumijevanje i upravljanje troškovima u oblaku i resursima kako bi se donijele informirane poslovne odluke. CloudVane sadrži alatnu ploču (engl. *dashboard*) koja nudi pogled na izvještaje iz oblaka, proračune, pragove proračuna i hijerarhiju organizacije kroz jedinstveni pogled. Neke od funkcionalnosti koje CloudVane nudi jesu:

- preporuke za pravilno dimenzioniranje²
- uvidi u uporabu resursa
- otkrivanje anomalija u troškovima
- planiranje rezerviranih instanci³.
- izvještaji o uštedama oblaka i resursa
- automatizacije nad resursima⁴
- obavijesti o troškovima

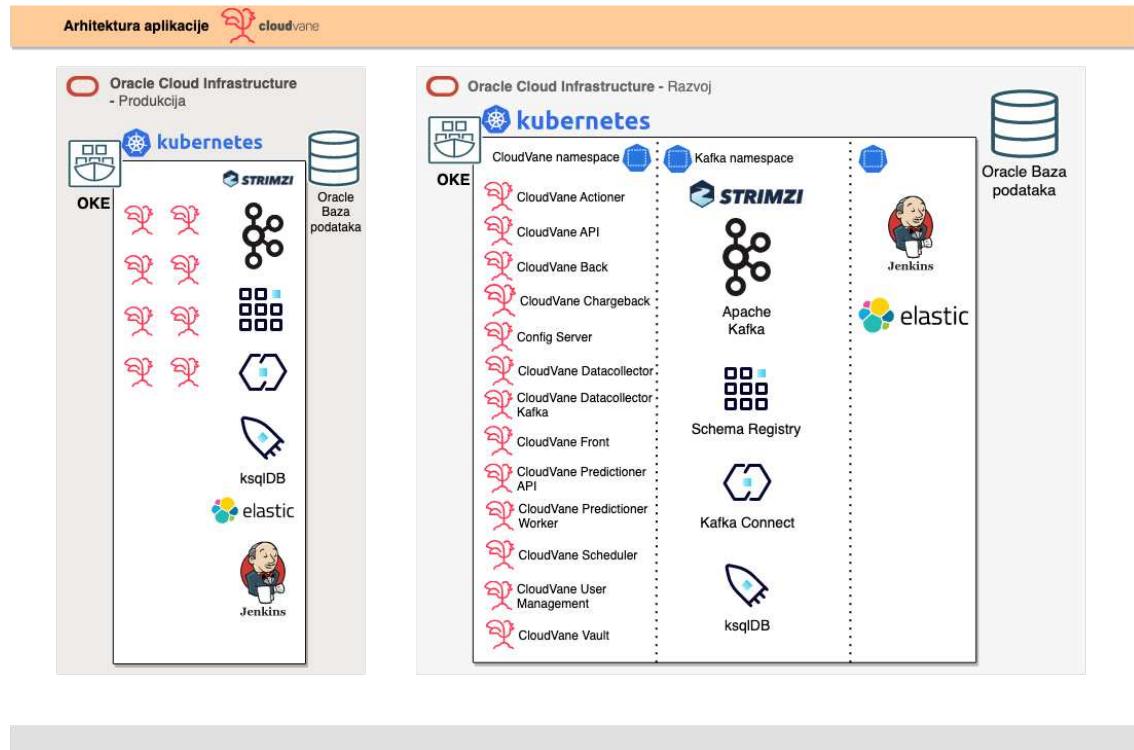
¹FinOps je udruga koja se bavi kulturom upravljanja troškova usluga u oblaku. Naziv FinOps dolazi od riječi financijsko upravljanje (engl. *Financial Operations*) i definira prakse, principe i kulturu kako bi organizacije bolje razumjele svoje troškove u oblaku i donosile informirane poslovne odluke (Finops Fundation, 2021).

²Dimenzioniranje je skaliranje resursa na oblaku s obzirom na njihovu stvarnu potrošnju kako bi se smanjila neiskorištenost.

³Rezervirane instance (engl. *Reserved instances*) jesu unaprijed plaćene usluge oblaka s popustom.

⁴Automatizacije čine gašenje resursa kada se ne koriste u svrhu uštede.

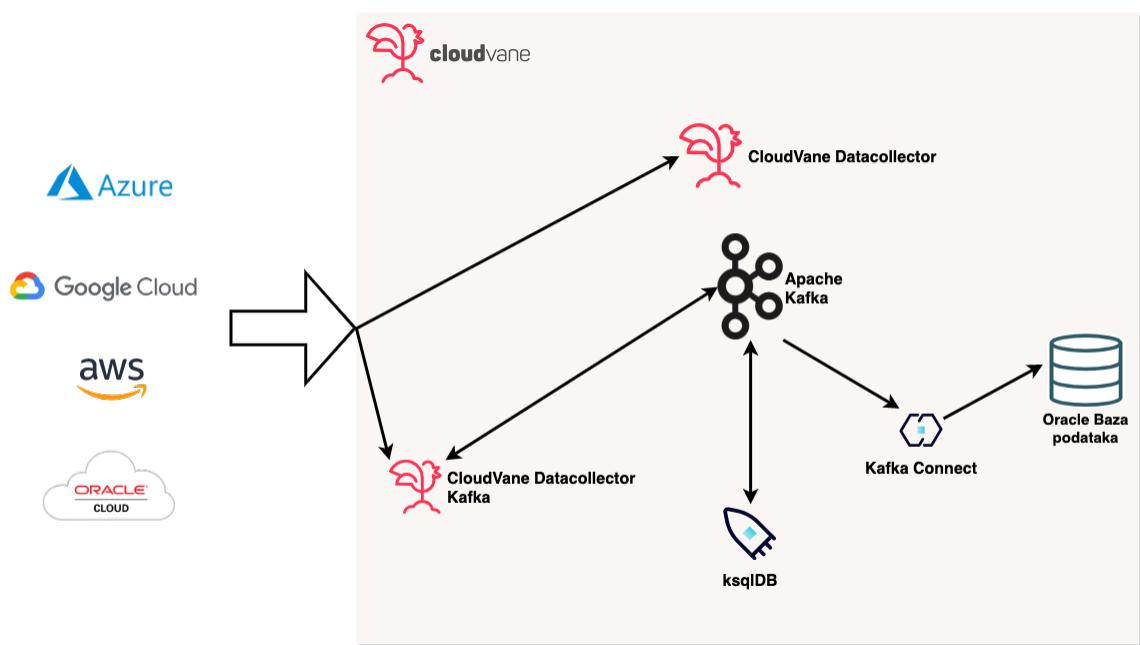
Upravo na primjeru distribuirane aplikacije CloudVane čija je arhitektura prikazana na slici 3.1, a protok podataka na slici 3.2, ovaj rad će demonstrirati analizu dnevničkih zapisa objašnjениh u poglavlju 4.1, svih dostupnih mikro-servisa aplikacije CloudVane u stvarnom vremenu (engl. *real-time*) sa svrhom kreiranja sustava nadgledanja i nadzora aplikacija (engl. *application monitoring*).



Slika 3.1: Arhitektura CloudVane sustava

Nadalje, CloudVane aplikacija je pokrenuta na infrastrukturi održavanog Kuberntes grozda (engl. *managed Kubernetes cluster*) na *Oracle Cloud Infrastructure* (OCI) oblaku. *Kubernetes*, opisan u poglavlju 3.1, pokreće sve komponente aplikacije CloudVane koji uključuju i Apache Kafku opisanu u poglavlju 3.2, osim Oracle relacijske baze podataka koja se pokreće kao servis na samom OCI oblaku. Arhitektura aplikacije opisana je u poglavlju 3.3, a transparentnost koju pruža Kubernetes omogućuje selidbu cijele infrastrukture na bilo koji *hiper-skalabilni pružatelj usluga oblaka*⁵ (engl. *hyper-scaler*).

⁵Ovaj termin odnosi se na najveće pružatelje usluge oblaka poput Amazon Web Services (AWS), Google Cloud Platform (GCP), Microsoft Azure itd. Naziv dolazi od sposobnosti brzog skaliranja računalne infrastrukture s obzirom na potražnju.



Slika 3.2: Tok podataka (engl. *data pipeline*) CloudVane sustava

3.1. Orkestrator kontejnera — Kubernetes

Lukša (2023) definira **Kubernetes** kao softverski sustav za automatizaciju implementacije (engl. *deployment*) i upravljanja složenim, velikim aplikacijskim sustavima sastavljenim od računalnih procesa koji rade u kontejnerima. Kubernetes programerima pruža sloj apstrakcije nad hardverom, što im omogućava da se usredotoče na razvoj aplikacija umjesto na upravljanje infrastrukturom.

Tako Kubernetes omogućuje implementaciju aplikacija potpuno transparentno, bez obzira na broj čvorova grozda i njihovu arhitekturu. Programeri ne moraju brinuti o detaljima kao što su odabir hardvera, operacijskog sustava za svaki čvor ili podjela resursa između različitih aplikacija. Kubernetes također može skalirati aplikacije na temelju opterećenja i automatski se oporavlja od pogrešaka, čime se osigurava visoka dostupnost i otpornost aplikacija. Zahvaljujući tome aplikacije mogu nesmetano raditi čak i u slučaju otkazivanja hardvera ili mrežnih problema.

Kubernetes se može pokrenuti direktno na hardveru računala nudeći potpunu kontrolu i fleksibilnost u konfiguraciji i upravljanju resursima. Alternativno se Kubernetes može koristiti kao *potpuno održavana usluga* (engl. *managed service*) na oblaku (engl. *cloud*). Potpuno održavan Kubernetes uvelike olakšava postavljanje i upravljanje Kubernetes grozdovima preuzimajući na sebe vremenski zahtjevne operativne zadatke poput skaliranja, nadogradnje, sigurnosti i ostalog održavanja.

Bez obzira na to kako je pokrenut, Kubernetes predstavlja robustan, fleksibilan i skalabilan okvir za upravljanje kontejnerskim aplikacijama koji omogućuje transparentno i jednostavno upravljanje distribuiranim aplikacijama.

3.1.1. Implementacija aplikacije CloudVane na Kubernetesu

Kao što je prethodno spomenuto, svi mikro-servisi koje koristi CloudVane, kao i Apache Kafka servisi kasnije opisani u poglavlju 3.2, operiraju unutar Kubernetes klastera na *Oracle Cloud Infrastructure (OCI)* platformi. DevOps prakse, koje kombiniraju elemente softverskog razvoja (engl. *Development*) i IT operacija (engl. *Operations*) kako bi se poboljšala brzina isporuke i kvaliteta softvera (Courtemanche et al., 2021), također su implementirane unutar ovog sustava.

Jenkins, moćan alat za kontinuiranu integraciju koji automatizira ključne aspekte razvojnog procesa softvera, poput izgradnje, testiranja i implementacije koda, dodatno

je integriran u ovaj Kubernetes klaster. Jenkins omogućuje korisnicima agilne DevOps procese pružajući automatizirane cikluse izgradnje za brzu i pouzdanu isporuku koda.

Jedini element infrastrukture koji se ne pokreće unutar Kubernetes klastera je *Oracle Autonomous Database for Transaction Processing and Mixed Workloads* (ATP) baza podataka koja je implementirana kao samostalna usluga unutar OCI oblaka. Ova sofisticirana relacijska baza podataka pruža robusnu, visoko dostupnu i skalabilnu platformu za upravljanje transakcijskim i mješovitim radnim opterećenjima, što dodatno poboljšava ukupnu efikasnost i performanse CloudVane aplikacije.

3.1.2. Komponente sustava Kubernetes

Kako bi se uspješno kretalo kroz kompleksni svijet Kuberntesa i učinkovito upravljalo njegovim resursima, razumijevanje temeljne terminologije je neophodno. Ovo poglavlje pruža uvid u ključne pojmove i komponente važne za razumijevanje ovog rada.

Također je bitno istaknuti da se Kubernetes objekti konfiguiraju putem deskriptivnih *YAML*⁶ ili *JSON*⁷ datoteka, omogućavajući precizno i reproducibilno definiranje stanja resursa unutar klastera.

Slijede objašnjenja ključne terminologije temeljena na knjizi autora Lukše (Lukša, 2023):

Pod: Pod je najmanja i najjednostavnija jedinica u Kubernetes modelu. To je skupina jednog ili više kontejnera koji dijele mrežu, prostor za pohranu i postavke za pokretanje.

Service: Service je apstrakcija koja definira logički set Pod-ova i pravila pristupa njima putem mreže.

Volume: Volume je mehanizam za pohranu podataka. On omogućuje da podaci ne nestanu kod ponovnog pokretanja Pod-a.

⁶YAML (engl. *YAML Ain't Markup Language*) je format za serijalizaciju podataka idealan za konfiguracijske datoteke.

⁷JSON (engl. *JavaScript Object Notation*) je format koji je lagan za čitanje i pisanje za ljude, a jednostavan za strojeve za obradu.

ConfigMap: ConfigMap omogućava spremanje konfiguracijskih informacija u obliku parova ključ/vrijednost. Ove informacije se zatim mogu koristiti unutar Pod-ova.

Secret: Secret se koristi za spremanje osjetljivih informacija, poput lozinki, OAuth tokena ili SSH ključeva. Ove informacije se zatim mogu koristiti kao i Config-Map.

Kontroler: Kontroler je komponenta koja nadzire određene aspekte klastera, reagirajući na njihove promjene.

ReplicaSet: ReplicaSet je kontroler koji upravlja brojem identičnih kopija Pod-a koje se izvode na klasteru.

Deployment: Deployment je kontroler koji omogućuje ažuriranje aplikacija na servima bez prekida rada. Deployment upravlja ReplicaSet-ovima.

DaemonSet: DaemonSet je kontroler koji osigurava da se kopija određenog Pod-a pokreće na svim čvorovima (ili na odabranim čvorovima) unutar klastera.

StatefulSet: StatefulSet je kontroler koji omogućuje upravljanje stanjem aplikacija.

Ingress: Ingress je API objekt koji upravlja vanjskim pristupima servisima u klasteru.

Kubernetes prostor imena (engl. Namespace): Prostor imena je mehanizam za izolaciju resursa u klasteru.

CRD (engl. Custom Resource Definitions): CRD-ovi su snažan okvir za stvaranje i upravljanje prilagođenim resursima.

Kubelet: Kubelet je agent koji se pokreće na svakom čvoru klastera i osigurava da su kontejneri u Pod-u pokrenuti i rade ispravno.

Kubectl: Kubectl je komandno-linijski alat za interakciju s Kubernetes klasterom.

Kubernetes operator: Operator je metoda koja omogućuje pakiranje, implementaciju i upravljanje Kubernetes aplikacijom.

3.1.3. Minikube

Za implementaciju komponenti ovog rada koristi se Kubernetes klaster različit od onoga na kojem funkcioniра CloudVane okruženje. Konkretno **Minikube**, alat dizajniran za rad s Kubernetesom na lokalnoj razini, koji se izvodi kao Docker kontejner.

Minikube je visoko prilagodljiv i lako dostupan alat koji pojednostavljuje proces postavljanja i testiranja Kubernetes aplikacija u izoliranoj i kontroliranoj okolini. Njegova sposobnost da simulira Kubernetes klaster unutar Docker kontejnera čini ga iznimno korisnim za razvoj i testiranje omogućujući programerima da eksperimentiraju, prate promjene i brzo testiraju nove funkcionalnosti bez rizika za produkcijsko okruženje (Lukša, 2023).

Ovaj pristup omogućuje sigurno testiranje i iteraciju kroz razvoj komponenata ovog rada istovremeno osiguravajući stabilnost i integritet CloudVane okoline.

Minikube zbog činjenice da je pokrenut kao Docker kontejner sadrži samo jedan računalni čvor u Kubernetes grozdu koji stvara. Međutim, svi Kubernetes objekti koji su razvijeni u sklopu ovog rada te dizajnirani i pokrenuti u Minikube-u mogu se lako migrirati u drugi Kubernetes klaster s minimalnim potrebnim preinakama.

3.2. Sustav za obradu podataka u stvarnom vremenu

— Apache Kafka

U suvremenom digitalnom dobu količina informacija i podataka koje se generiraju svakodnevno je zaista ogromna. Upravo zbog tog enormnog broja podataka sve je prisutnija potreba za izgradnjom sofisticiranih računalnih sustava koji su sposobni za brzo i efikasno procesiranje i analizu tih informacija (Čubek, 2022).

U tom se kontekstu **Apache Kafka** izdvaja se kao ključni alat. Apache Kafka je snažna, efikasna i skalabilna platforma za obradu podataka u stvarnom vremenu. Arhitektura Apache Kafke omogućuje brzu i pouzdanu obradu tokova podataka s visokom propusnošću čineći je idealnim alatom za kompleksne i zahtjevne aplikacije.

Originalno razvijena unutar LinkedIn-a, Kafka je bila rješenje za izazove povezane s upravljanjem velikim tokovima podataka. Dizajnirana kao visoko efikasan sustav poruka, Kafka omogućava dostavljanje čistih, strukturiranih informacija o korisničkim aktivnostima i metrikama sustava u stvarnom vremenu. To čini Apache Kafku ključnim resursom za sučeljavanje s izazovima modernog doba povezanih s obradom i analizom velikih količina podataka (Shapira et al., 2021).

Kao dodatak brzini i efikasnosti Kafka se ističe i svojom skalabilnošću. U mogućnosti je obraditi velike količine podataka bez gubitka performansi, što ju čini iznimno prilagodljivim rješenjem za različite potrebe korisnika. Sve to čini Kafka idealnim rješenjem za sučeljavanje s izazovima povezanim s upravljanjem velikim volumenom podataka u stvarnom vremenu.

Kao što je detaljno opisano u poglavlju 3.3, CloudVane aplikacija koristi Apache Kafka kao središnje spremište za podatke dohvaćene s poslužitelja u oblaku. Ova integrirana platforma pruža bogat ekosustav usluga koje dopunjuju temeljni Kafka sustav, uključujući *Kafka posrednike*, *Apache Zookeeper*, *Schema Registry*, *Kafka Connect*, *ksqlDB* i *Kafka UI* servise.

Svaki od ovih servisa ima ključnu ulogu u obradi i upravljanju podacima. Specifična uloga svakog servisa u kontekstu obrade podataka aplikacije CloudVane također je pažljivo razrađena u poglavlju 3.3.

Ova strukturirana arhitektura koju omogućuje Apache Kafka, s usko povezanim srodnim servisima, omogućava CloudVane aplikaciji da efikasno upravlja velikim količinama podataka pružajući robustan i pouzdan sustav za obradu podataka u stvarnom

vremenu. Detalji o funkcionalnostima i karakteristikama svakog servisa predstavljeni su u idućim poglavljima.

3.2.1. Komponente sustava Apache Kafka

Slično kao i s Kubernetesom, kretanje kroz složeni svijet Apache Kafke zahtijeva dobro razumijevanje osnovne terminologije. Cilj je ovog poglavlja pružiti uvod u ključne pojmove i komponente koji su bitni za razumijevanje rada.

Kafka posrednik (engl. *Kafka Broker*): predstavlja server u Apache Kafka ekosustavu. Kafka posrednik odgovoran je za pohranu, kategorizaciju i distribuciju poruka (ili događaja) do Kafka klijenata. Skup posrednika formira Kafka klaster, u kojem svaki posrednik efikasno skladišti i upravlja mnoštvom Kafka tema.

Kafka tema (engl. *Kafka Topic*): predstavlja logičku kategoriju ili kanal kroz koji protječe tok poruka. Zapis u temi sadrži ključ, vrijednost i vremensku oznaku. Teme u Kafki su dizajnirane tako da se mogu dalje podijeliti na manje dijelove ili particije.

Kafka particija (engl. *Kafka Partition*): predstavlja podjelu ili frakciju podataka unutar teme. Particije su ključne za paralelizaciju i omogućavaju Kafki da poboljša performanse i izdržljivost pohranjivanjem i čitanjem podataka s različitih posrednika u klasteru. Svaka particija može imati jednu ili više replika, što omogućava oporavak podataka u slučaju gubitka posrednika.

Kafka segment: predstavlja podjelu ili dio Kafka particije. Svaka particija organizirana je kao niz segmenata, gdje svaki segment predstavlja kolekciju zapisa.

U Kafka sustavu, podaci se u Kafka teme unose kroz producente i iz njih se izvlače kroz konzumente.

Producenti (engl. *Producers*): su odgovorni za slanje poruka u Kafka teme. Svaka poruka koju producent pošalje sadrži ključ i vrijednost. Ključ je obično povezan s određenom particijom teme, dok vrijednost sadrži stvarne podatke.

Konzumenti (engl. *Consumers*): su odgovorni za čitanje i obradu poruka iz Kafka teme. Svaki konzument prati svoju trenutnu poziciju (engl. *offset*) u svakoj particiji teme, što označava do kojeg je zapisa došao u svom čitanju.

Kako bi se omogućio kontinuirani tok podataka, Kafka koristi Consumer API za konzumente i Producer API za producente. Ovi API-ji omogućuju konzistentno i efikasno slanje, čitanje i obradu poruka unutar Kafka ekosustava.

Konzumenti u Kafka mogu biti konfigurirani da slijede jedan od triju modela dostave:

Najviše jednom (engl. *At most once*): Ovaj model omogućuje da neke poruke mogu biti izgubljene u slučaju greške, ali nikada neće doći do duplikata poruka. Koristi se kada je korisniku prihvatljiv određeni stupanj gubitka podataka.

Najmanje jednom (engl. *At least once*): Ovaj model osigurava da svaka poruka uvijek bude dostavljena, iako može doći do dupliciranja poruka u slučaju greške. Koristi se kada je apsolutno neophodno da nijedna poruka ne bude izgubljena.

Točno jednom (engl. *Exactly once*): Ovaj model osigurava da svaka poruka bude dostavljena točno jednom, bez gubitka ili dupliciranja. Ovaj je model najzahtjevniji za implementaciju i može imati veći utjecaj na performanse, ali koristi se kada je nužno da svaki zapis bude precizno obrađen.

3.2.2. Operator za Apache Kafka na Kubernetesu — Strimzi

Apache Kafka servisi implementirani su unutar CloudVane Kubernetes klastera korištenjem alata **Strimzi**. Strimzi je projekt otvorenog koda koji pruža operator za pokretanje Apache Kafka na Kubernetesu, znatno pojednostavljajući proces konfiguriranja, pokretanja i održavanja Apache Kafka klastera unutar Kubernetes okruženja.

Strimzi koristi Kubernetes CRD-ove kako bi stvorio potrebne Kubernetes Deployment-e i StatefulSet-ove. Ovi objekti omogućavaju definiranje i upravljanje Apache Kafka klasterima na jasan i deklarativan način, što rezultira većom učinkovitošću i manjom vjerojatnošću za greške.

Također, Strimzi omogućava jednostavno kreiranje i upravljanje Apache Kafka temama i Kafka Connect konektorima. Ovo znatno olakšava proces konfiguriranja i upravljanja ovim resursima smanjujući potrebu za ručnim intervencijama i omogućavajući brže i učinkovitije upravljanje Kafka okruženjem.

Pored toga, Strimzi dolazi s podrškom za razne sigurnosne značajke, poput TLS-a, što olakšava implementaciju sigurnosnih mjera i osigurava zaštitu Kafka klastera.

Upravo zbog navedenog jednostavnijeg, sigurnijeg i efikasnijeg upravljanja Apache Kafka okruženjem, CloudVane se koristi operatorom Strimzi za implementaciju Apache Kafka posrednika na svoj Kubernetes grozd.

3.2.3. Apache Zookeeper

Apache ZooKeeper je distribuirana usluga za koordinaciju, otvorenog je koda i njome se koriste distribuirane aplikacije. Razvila ga je Apache Fundacija i prvi put je izdan u 2008. godini.

ZooKeeper se često koristi za održavanje konfiguracija, pružanje distribuiranog sinkroniziranja, praćenje statusa čvorova u klasteru i pružanje grupnih usluga (Hunt et al., 2010). Primjerice, u slučaju Apache Kafka, ZooKeeper se koristi za upravljanje i koordinaciju brokera.

Međutim, Apache Kafka je nedavno (u vrijeme pisanja ovoga rada) prešla s korištenja ZooKeeper-a na novi protokol konsenzusa nazvan **Kafka Raft (Kraft)**.

Kako Chandrakant (2022) navodi, cilj prelaska s ZooKeeper-a na Kraft je pojednostaviti arhitekturu Kafke i poboljšati je. Kraft je varijanta Raft konsenzusnog protokola temeljena na događajima i promjena koju uvodi smanjuje prozor nedostupnosti kod pogrešaka Kafka posrednika jer može nadoknaditi propuštene događaje iz Kafka dnevnika nakon ponovnog pridruživanja.

Očekuje se da će prelazak na Kraft biti od značajne koristi zajednici Apache Kafke. Administratori sustava će lakše nadzirati, administrirati i podržavati Kafku. Novo upravljanje meta-podacima također znatno poboljšava performanse Apache Kafke omogućujući posredniku da se brže oporavi od pogrešaka i da obrađuje mnogo veći broj particija (Chandrakant, 2022).

3.2.4. Kafka Streams

Kafka Streams je lagana i snažna Java biblioteka za obogaćivanje, transformiranje i obradu tokova podataka u stvarnom vremenu (Seymour, 2021). Ova biblioteka omogućuje obradu događaja iz Kafka tema uključujući različite operacije poput agregacija (engl. *stream aggregation*), spajanja tokova (engl. *stream joining*), filtriranja poruka, itd., koje se zatim nakon definiranih operacija šalju u novu Kafka temu.

Kafka Stream pruža jednostavan API za obradu podataka omogućujući korisnicima da se usredotoče na logiku obrade podataka bez potrebe za upravljanjem infrastrukturom. Jedna od ključnih prednosti Kafka Streams biblioteke je njena sposobnost da obradi podatke u stvarnom vremenu. Za razliku od obrade u serijama (engl. *batch*) koja obrađuje podatke u određenim vremenskim intervalima, Kafka Streams omogućuje obradu podataka kako oni dolaze, što rezultira nižim latencijama i omogućuje aplikacijama da pruže ažurirane informacije u stvarnom vremenu (Seymour, 2021).

Kafka Streams također podržava „*točno-jednom*“ semantiku obrade, što znači da će svaka poruka biti obrađena točno jednom, čime se eliminira mogućnost dvostrukе obrade istih podataka (Narkhede, 2017). Osim toga, Kafka Streams biblioteka je dizajnirana tako da bude lako skalabilna. Aplikacije mogu biti horizontalno skalabilne dodavanjem više instanci aplikacije, a Kafka Streams će automatski uravnotežiti obradu podataka između njih.

Agregacije u Kafka Streams-u

Agregacije predstavljaju ključni aspekt obrade tokova podataka pružajući korisnicima mogućnost izračunavanja statistika i izvlačenja korisnih informacija iz primljenih podataka. U osnovi one omogućuju kombiniranje više ulaznih poruka u jednu izlaznu poruku (Seymour, 2021).

Kafka Streams podržava širok spektar agregacijskih operacija. Ovo uključuje standardne matematičke funkcije, poput suma, minimuma i maksimuma, ali i korisnički definirane funkcije poput *aggregate* i *reduce*.

Agregacije se u Kafka Streams biblioteci provode na temelju *ključa* Kafka poruke. Ako ključ poruke ne odgovara vrijednosti prema kojoj je potrebno agregirati, Kafka Streams nudi proces promjene ključa (engl. *rekeying*). Međutim, ovaj proces rezultira stvaranjem nove Kafka teme.

Važna komponenta Kafka Streams-a prilikom obrade agregacija jest baza poda-

taka *RocksDB*. RocksDB se koristi kao lokalna baza podataka za brzo pohranjivanje i pristup podacima koji su ključni za agregacijske operacije. Stanje Kafka Streams aplikacije se također pohranjuje trajno u specijaliziranu Kafka temu, što omogućava otpornost na greške i oporavak.

Uz mogućnost provođenja agregacija na tokovima podataka, Kafka Streams biblioteka omogućava korisnicima transformaciju velikih količina podataka u korisne informacije. Ove informacije mogu biti od presudne važnosti za donošenje informiranih odluka.

Prozori u Kafka Streams-u

Jedna od naprednih funkcionalnosti Kafka Streams biblioteke je *prozoriranje* (engl. *windows*) koje omogućuje obradu podataka unutar određenog vremenskog intervala, poznatog kao prozor (engl. *window*). Prozoriranje je posebno korisno kada se podaci trebaju agregirati na temelju vremenskih intervala, kao što su *minuta*, *sat* ili *dan*.

U biblioteci Kafka Streams postoje tri glavne vrste prozora, čije se karakteristike mogu pregledati u tablici pod referencom 3.1 i na slici 3.3. Ove vrste prozora omogućuju provođenje složenih analitičkih operacija, poput izračuna pomičnog prosjeka ili izračuna agregata unutar određenoga vremenskog intervala.

Iako se u tablici navode četiri vrste prozora, *klizeći prozor* (engl. *sliding window*) zapravo je podskup *skakućeg prozora* (engl. *hopping window*) s beskonačno malim vremenskim pomakom. Ovo znači da, za razliku od skakućeg prozora koji „preskače“ na novi vremenski interval, klizeći prozor kontinuirano „klizi“ kroz vremensku os pružajući time neprekidan pregled podataka.

Ove različite vrste prozora pružaju fleksibilnost u obradi tokova podataka s Kafkom omogućujući različite načine gledanja na podatke ovisno o specifičnim potrebama aplikacije.

Kafka Streams i ksqlDB

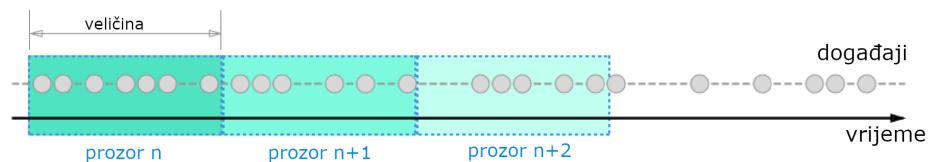
CloudVane se trenutno ne koristi bibliotekom Kafka Streams za obradu tokova podataka, već se oslanja na ksqlDB servis, koji je detaljno opisan u sljedećem poglavljju. No, važno je napomenuti da je ksqlDB zapravo izgrađen na temelju Kafka Streams biblioteke, djelujući kao sloj više razine apstrakcije kao što je prikazano na slici 3.4.

Tablica 3.1: Vrste prozora u Kafka Streams biblioteci

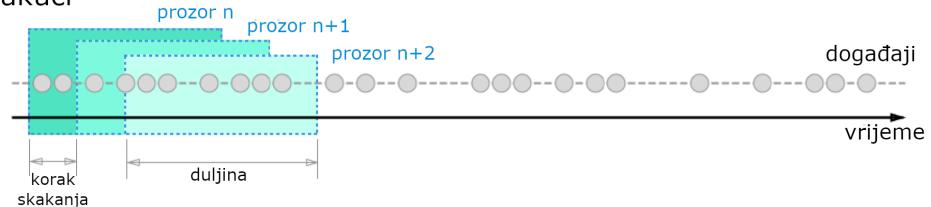
Vrsta prozora	Naziv na engl.	Karakteristika prozora
Prevrtajući prozor	Tumbling window	Ne preklapa se, imaju veličinu
Skakući prozor	Hopping window	Preklapa se, ima veličinu i iznos koliko skače
Sjednički prozor	Session window	Određen duljinom neaktivnosti
Klizeći prozor	Sliding window	Ima veličinu i klizi

Prozorne agregacije

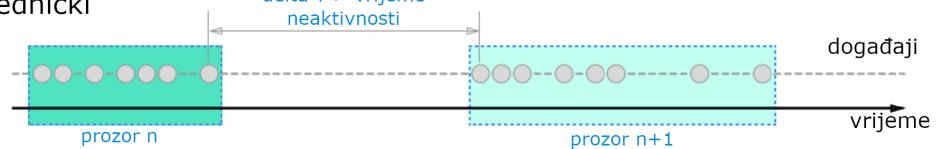
Prevrtajući



Skakući

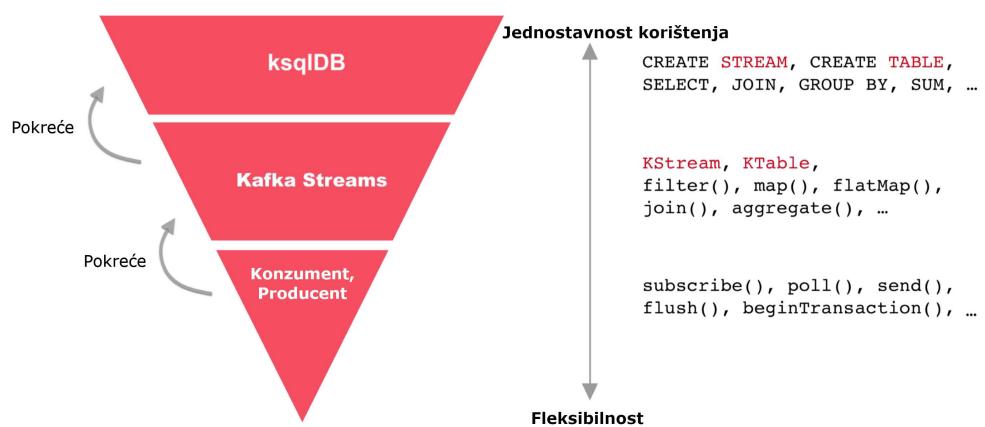


Sjednički



Slika 3.3: Prozori agregacija u Kafka Streams-u i ksqlDB-u

Slika preuzeta sa <https://docs.ksqldb.io/en/latest/concepts/time-and-windows-in-ksqldb-queries/>



Slika 3.4: Hjерархија Kafka сервиса за обраду токова података

Slika preuzeta sa <https://docs.ksqldb.io/en/latest/operate-and-deploy/how-it-works/>

3.2.5. ksqlDB

KsqlDB je specijalizirana baza podataka za obradu tokova događaja (engl. *event streaming database*). Ova platforma značajno pojednostavljuje implementaciju, pokretanje i održavanje aplikacija za obradu tokova podataka integrirajući dvije ključne komponente Kafka ekosustava: *Kafka Connect* i *Kafka Streams* (Seymour, 2021).

Kafka Connect, opisan u sljedećem poglavlju, servis je koji omogućuje jednostavno povezivanje Kafke s drugim sustavima. Integracija Kafka Connect-a u ksqlDB omogućuje korisnicima da lako povežu svoje aplikacije s različitim izvorima i odredištimi podataka.

Kafka Streams s druge strane, kao što je prikazano na slici 3.4, gradi ksqlDB te omogućuje korisnicima da se koriste funkcionalnostima obrade podataka koje ona pruža, ali s jednostavnosću upita sličnim SQL-u.

KsqlDB, kombinirajući ove dvije komponente, pruža snažnu platformu za obradu tokova podataka. Korisnici mogu jednostavno implementirati složene obrade podataka koristeći jezik sličan poznatom SQL-u, dok ksqlDB obavlja teški posao povezivanja s izvorima i odredištimi podataka i obrade podataka u stvarnom vremenu. Ova integracija omogućuje korisnicima da se usredotoče na logiku svojih aplikacija, umjesto na složenosti upravljanja infrastrukturom obrade podataka.

CloudVane se oslanja upravo na ksqlDB klaster za obradu, agregaciju i manipulaciju tokova podataka koje prikuplja od podržanih pružatelja oblaka. KsqlDB ne samo da omogućuje agregaciju podataka već i provodi filtriranje i transformaciju podataka, pružajući CloudVane aplikaciji snažan alat za efikasnu i efektivnu obradu podataka.

3.2.6. Kafka Connect

Kafka Connect je integralni dio Apache Kafka ekosustava i pruža skalabilan te pouzdan način za kopiranje podataka između Kafka i drugih skladišta podataka (Shapira et al., 2021).

Kafka Connect je dizajniran da podrži veliki broj izvora (engl. *source*) i odredišta (engl. *sink*) podataka omogućujući korisnicima da jednostavno povežu Kafku s različitim sustavima za skladištenje podataka. To uključuje tradicionalne baze podataka, datotečne sustave, usluge oblaka i mnoge druge.

Jedna od ključnih funkcionalnosti Kafka Connect-a je njegova sposobnost da se jednostavno horizontalno skalira. To znači da može podržati velike količine podataka i veliki broj veza, što ga čini idealnim za velike, podatkovno intenzivne aplikacije.

Uz to, Kafka Connect podržava neke transformacije podataka omogućujući korisnicima da prilagode podatke kako bi bolje odgovarali njihovim specifičnim potrebama pri ulazu ili izlazu iz Apache Kafke. To može uključivati promjenu formata podataka, dodavanje ili uklanjanje polja, ili nekoliko drugih vrsta manipulacije podacima.

Ukupno gledano, Kafka Connect pruža snažan i fleksibilan način za povezivanje Kafke s različitim skladištima podataka omogućujući korisnicima da maksimalno iskoriste svoje podatke.

U kontekstu CloudVane aplikacije Kafka Connect se koristi za prijenos podataka koji su agregirani putem ksqlDB-a u ciljnu bazu podataka, konkretno *Oracle ATP* bazu.

Ova funkcionalnost omogućava aplikaciji CloudVane da efikasno i pouzdano prenosi obrađene i agregirane podatke iz Kafka ekosustava u *Oracle ATP* bazu, što dalje omogućava naprednu analizu i obradu tih podataka. Korištenje Kafka Connect-a za ovu svrhu osigurava da se podaci prenose na siguran i pouzdan način, uz održavanje visoke razine performansi i mogućnosti skaliranja.

Ova integracija između Kafka Connect-a, ksqlDB-a i Oracle ATP baze podataka omogućava aplikaciji CloudVane da maksimalno iskoristi svoje podatke, pružajući joj snažne alate za obradu podataka u stvarnom vremenu i njihovo pohranjivanje za daljnju analizu i obradu.

3.2.7. Avro serializacija i Schema Registry

Avro je format serijalizacije podataka neovisan o programskom jeziku što omogućuje njegovu široku primjenu u različitim programskim okruženjima (Shapira et al., 2021). Glavne prednosti Avro-a su njegova sposobnost smanjenja mrežnog prometa kroz odvajanje sheme podataka od samih vrijednosti i korištenje binarnog formata. Ova značajka omogućuje efikasno upravljanje velikim količinama podataka, što je često ključno u sustavima za obradu podataka u stvarnom vremenu.

Avro također podržava *jaku tipizaciju* (engl. *strong typing*), što znači da svaki podatak ima strogo definiran tip. Ova značajka pomaže u sprječavanju pogrešaka i poboljšava pouzdanost obrade podataka jer se mogu izbjegći nekompatibilnosti ili nesporazumi oko formata ili strukture podataka.

Za rad s Apache Kafka sustavom Avro se često koristi u kombinaciji s **Apache Schema Registry** servisom. Schema Registry je servis koji pomaže u upravljanju shemama i provođenju pravila o shemama osiguravajući da svi podaci koji se upisuju u Kafka teme prate prije definiranu shemu. Ovo je ključno za održavanje kompatibilnosti podataka kako se sustav razvija jer omogućuje promjene u formatu podataka bez gubitka kompatibilnosti s postojećim podacima.

Sveukupno, kombinacija Avro-a i Schema Registryja pruža robustan okvir za upravljanje i obradu strukturiranih podataka unutar Apache Kafka sustava. Ove tehnologije omogućuju korisnicima da efikasno i pouzdano upravljaju velikim količinama podataka istovremeno održavajući visoku razinu pouzdanosti i kompatibilnosti. Upravo iz tih razloga CloudVane aplikacija koristi Avro format za pohranu podataka u Kafka teme i Schema Registry servis za održavanje i upravljanje shemama.

Ova praksa omogućuje aplikaciji CloudVane da maksimalno iskoristi prednosti koje Apache Kafka pruža, uključujući skalabilnost, brzinu i pouzdanost. Korištenje Avro formata osigurava da su podaci kompaktni i dobro strukturirani, što poboljšava performanse i omogućuje efikasnu obradu podataka. Istovremeno upotreba Schema Registry servisa osigurava da su sve sheme podataka dosljedno upravljane i održavane, što pomaže u održavanju kvalitete podataka i kompatibilnosti kroz vrijeme.

3.2.8. Kafka UI

Pristupanje Apache Kafka sustavu koristeći komandno-linijske alate (engl. *Command Line Interface tools*) može biti težak i spor proces, pogotovo za korisnike koji nisu iskusni s komandno-linijskim alatima. S druge strane, korištenje grafičkih korisničkih

sučelja (engl. *Graphical User Interface*) pruža nekoliko prednosti koje mogu poboljšati produktivnost i iskustvo rada s Apache Kafka temama, shemama i konektorima.

Korisnička sučelja omogućuju jednostavnije i efikasnije upravljanje jer eliminiraju potrebu za pamćenjem složenih naredbi koje su neophodne za provođenje osnovnih operacija na Apache Kafki. Upravo iz tog razloga, uz Apache Kafku, pokrenut je Kafka UI servis na Kubernetes klasteru.

3.3. Arhitektura aplikacije CloudVane

CloudVane, kako je prethodno navedeno, *distribuirana* je aplikacija koja se izvodi na *Oracle Kubernetes Engine* (OKE) platformi. Osim toga, CloudVane je primjer *mikro-servisne arhitekture*, što znači da se sastoji od brojnih servisa, svaki s određenim zadacima unutar cjelokupnog sustava. Ovi servisi su detaljno opisani u tablici 3.2 i 3.3, a cjelokupna arhitektura sustava prikazana je na slici 3.1.

Ova struktura omogućuje visoku razinu modularnosti i fleksibilnosti. Svaki servis može se razvijati, skalirati i održavati neovisno o ostalima, što omogućuje brzi razvoj i lako ažuriranje. Također, ova arhitektura omogućuje bolju iskoristivost resursa jer svaki servis može biti skaliran prema svojim specifičnim potrebama.

U kontekstu CloudVane aplikacije, ova mikro-servisna arhitektura omogućuje efikasno upravljanje i obradu velikih količina podataka. Svaki servis može se specijalizirati za određeni aspekt obrade podataka, što rezultira visokom performansom i pouzdanošću cjelokupnog sustava. Detaljan opis svakog CloudVane servisa može se pronaći u tablici 3.2.

Uzimajući u obzir da CloudVane koristi usluge Apache Kafke, te usluge su smještene u zasebnom Kubernetes prostoru imena u odnosu na glavne servise aplikacije CloudVane. Apache Kafka servisi opisani su u tablici 3.3, a protok podataka kroz CloudVane sustav prikazan je na slici 3.2.

Ova organizacijska struktura, koja uključuje odvajanje proizvodnjskog klastera od klastera za razvoj i testiranje, omogućuje jasnu segregaciju i izolaciju resursa među različitim komponentama sustava. Takva struktura ne samo da pojačava sigurnost i upravljivost sustava već i olakšava skaliranje i održavanje svake komponente neovisno jedna o drugoj. Osim toga, ovakva organizacija pruža bolju kontrolu nad resursima te olakšava praćenje i dijagnostiku problema unutar sustava. U konačnici, ovakav pristup omogućuje efikasnije upravljanje resursima i optimizaciju performansi, čime se poboljšava ukupna pouzdanost i efikasnost CloudVane aplikacije.

Tablica 3.2: CloudVane servisi na Kuberentesu

Naziv servisa	Opis servisa
CloudVane Actioner	Upravlja akcijama na oblaku
CloudVane API	REST poslužitelj i autentifikacija
CloudVane Back	Spring poslužiteljske strane
CloudVane Chargeback	Omogućuje izradu izvještaja o potrošnji
CloudVane Config	Spring Config server
CloudVane Datacollector Kafka	Dohvaća podatke iz oblaka, obrađuje ih i prosljeđuje u Apache Kafku
CloudVane Datacollector	Dohvaća podatke iz oblaka, obrađuje ih i pohranjuje direktno u bazu podataka
CloudVane Front	Angular poslužitelj klijentske strane
CloudVane Predictioner API	REST poslužitelj predikcijskog sustava
CloudVane Predictioner Worker	Sustav umjetne inteligencije koji računa predikcije
CloudVane Scheduler	Raspoređivač zadataka CloudVane aplikacije
CloudVane User Management	Sustav za upravljanje korisnicima i autorizaciju
CloudVane Vault	Sustav za sigurnu pohranu korisničkih kredencijala

Tablica 3.3: Kafka servisi na CloudVane Kuberentesu

Naziv Kafka servisa	Opis servisa
Kafka Broker	Poslužuje Apache Kafku
Kafka Connect klaster	Upis tokova u bazu podataka
ksqlDB	Obrađuje tokove podataka
Schema Registry	Upravlja shemama podataka
Kafka UI	Jednostavan pregled Kafka tema

3.4. CloudVane kao izvor podataka

U okviru ovog rada svi će dnevnički zapisи prethodno opisanih servisa biti preusmjereni iz testne okoline CloudVane klastera u Kafka teme, koje se nalaze u Apache Kafka na lokalnom Kubernetes klasteru. Takva struktura omogućit će efikasnu obradu zapisa te njihovo korištenje za generiranje rezultata istraživanja.

Metodologija i alati potrebni za praćenje aplikacije CloudVane bit će opisani u sljedećim poglavljima. Ovim pristupom osigurava se jasan i detaljan uvid u kompleksni proces obrade podataka u stvarnom vremenu koristeći kombinaciju Apache Kafka, Kubernetes i drugih alata te tehnologija.

4. Nadziranje aplikacija

Nadziranje (engl. *monitoring*), definirano kao sustavno promatranje i provjera ponašanja i izlaza sustava te njegovih komponenti kroz vrijeme (Monitorama, 2016), ključni je element upravljanja informatičkim sustavima i aplikacijama. Kroz tu praksu, koja se sastoji od stalnog nadzora i analize, omogućeno je održavanje optimalnih performansi, stabilnosti i pouzdanosti digitalnih resursa.

Nadzor aplikacija nije samo proces identifikacije i rješavanja problema. On nudi dragocjene uvide koji omogućuju temeljitiju optimizaciju sustava, poboljšanje korisničkog iskustva i osiguranje da se sve digitalne komponente funkcionalno integriraju. Sastoji se od kontinuirane provjere, identifikacije anomalija i promptnog rješavanja problema prije nego što se pretvore u kritične situacije. Ovim proaktivnim pristupom nadzor postaje ključan instrument za održavanje visoke razine usluge i postizanje operativne izvrsnosti.

U ovom poglavlju rada detaljno se istražuje nadzor aplikacija poput aplikacije CloudVane, koja je opisana u poglavlju 3, s posebnim naglaskom na kontekst *dnevničkih zapis*a prikazanih u poglavlju 4.1 iz *distribuiranih mikro-servisnih sustava*. Upotrebom alata *ELK*, čije su komponente i funkcionalnosti predstavljene u poglavlju 4.2, ilustrirat će se korištenje sofisticiranih alata za izdvajanje dnevničkih zapisova iz distribuiranih aplikacija unutar Kubernetes grozda. Detaljno će se obraditi procesi prikaza, transformacije i unosa dnevničkih zapisova u Apache Kafku.

4.1. Dnevnički zapisi

Dnevnički zapisi aplikacija (engl. *application logs*) ključna su komponenta koja pruža detaljan uvid u rad aplikacije te su ključni za otkrivanje i rješavanje problema, ali i za optimizaciju performansi aplikacije.

Dnevnički zapisi aplikacija sastoje se od sekvensijalno zapisanih događaja koji se generiraju unutar aplikacije. Ti događaji mogu biti različite vrste, uključujući informacije o radnji koju je korisnik poduzeo, statusu aplikacije, greškama, upozorenjima i drugim relevantnim informacijama. Dnevnički zapisi pružaju sveobuhvatnu sliku o tome što se događa unutar aplikacije u svakom trenutku.

U kontekstu *mikro-servisnih arhitektura*, kao što je to slučaj s CloudVane aplikacijom, dnevnički su zapisi posebno važni. Budući da je svaki mikro-servis odgovoran za svoj dio funkcionalnosti, dnevnički zapisi svakog mikro-servisa pružaju uvid u ispravnost rada cijele aplikacije. Ako dođe do problema, dnevnički zapisi mogu pomoći u identificiranju komponente koja je uzrokovala problem.

Međutim, upravljanje dnevničkim zapisima u mikro-servisnom okruženju može biti izazovno. Jedan od najizraženijih problema jest problem agregiranja i pretraživanja dnevničkih zapisa. S obzirom na to da je aplikacija raspoređena na više čvorova unutar Kubernetes grozda koji su izolirani, konsolidacija i pretraživanje informacija postaje kompleksan zadatak (Newman, 2015).

U takvom distribuiranom okruženju, svaka komponenta ili mikro-servis generira svoje dnevničke zapise. S obzirom na to da su ovi dnevnički zapisi disperzirani preko različitih čvorova, njihova centralizacija zahtijeva robusna i skalabilna rješenja. Bez odgovarajućih alata i strategija pretraživanje tih zapisa na jednom mjestu može biti pričinno izazovno i vremenski zahtjevno. Osim toga, pitanje je i kako oblikovati i strukturirati te zapise tako da pružaju korisne informacije. Različiti mikro-servisi mogu imati različite strukture dnevničkih zapisa. Zahtijeva se strategija koja ne samo da agregira ove zapise već ih i standardizira i strukturira za laku analizu i dijagnostiku.

U nastavku poglavlja fokus će se staviti na dnevničke zapise programskog jezika *Java*. Fokus na programski jezik Javu proizlazi iz činjenice da je većina servisa unutar distribuirane aplikacije CloudVane razvijena upravo ovim programskim jezikom. Iako i drugi programski jezici nude slične alate za konfiguraciju i upravljanje dnevničkim zapisima, specifičnosti Jave u ovom kontekstu su od posebne važnosti zbog potrebnih

konfiguracija. Također, posvetit će se pažnja ELK alatima, što je centralna tema poglavlja 4.2. Ova dva aspekta čine ključne komponente istraživanja i analize nadzora aplikacija unutar distribuiranih sustava.

4.1.1. Dnevnički zapisi u programskom jeziku Java

CloudVane koristi **Logback biblioteku** za svoje potrebe generiranja dnevničkih zapisu, također koristeći **SLF4J apstrakciju**¹ koju nudi biblioteka *Spring*. Logback je konfiguriran da ispisuje dnevničke zapise u JSON formatu, što olakšava njihovu obradu, posebno u distribuiranom sustavu gdje različiti servisi generiraju zapise. Primjer konfiguracije Logback-a na aplikaciji CloudVane prikazan je u isječku A.2.

Primjer jednog dnevničkog zapisa prikazan je u isječku A.3. Ovaj zapis ilustrira kako se informacije strukturiraju u JSON formatu.

Osim *Logback*, postoji i *Log4j* biblioteka, koja je također popularna opcija za dnevničke zapise u Java aplikacijama. Međutim, u kontekstu CloudVane aplikacije, Logback je odabran zbog fleksibilnosti u konfiguraciji.

4.2. Elastic, Logstash i Kibana (ELK)

ELK, akronim koji označava **ElasticSearch**, **Logstash** i **Kibana**, predstavlja snažan skup alata sposoban za prikupljanje, pohranu, analizu i vizualizaciju podataka u stvarnom vremenu. Često se koristi za upravljanje dnevničkim zapisima i praćenje događaja, što ga čini nezamjenjivim alatom za sigurnosne analize, poslovnu inteligenciju i efikasno pretraživanje podataka (Konda, 2023).

U ovom se poglavlju detaljno razmatraju funkcionalnosti svake pojedinačne komponente ELK-a te se opisuje njegova uloga u rješenju koje predlaže ovaj rad.

4.2.1. ElasticSearch

Elasticsearch je distribuirani sustav za pretragu i analizu podataka u stvarnom vremenu. Omogućava istraživanje podataka na brz i skalabilan način koji do sada nije bio ostvariv. Koristi se za pretragu cijelog teksta, strukturiranu pretragu, analitiku i kombinaciju svih triju funkcionalnosti (Gormley i Tong, 2015).

¹SLF4J nudi mogućnost generičkog pisanja dnevničkih zapisu u programskom kodu neovisno o konkretnoj implementaciji sustava generiranja dnevničkih zapisu.

Zasnovan na *Apache Lucene*², Elasticsearch predstavlja distribuirani sustav za pretragu teksta s HTTP sučeljem. Ovo sučelje omogućuje jednostavan pristup i manipulaciju podacima kroz standardizirane HTTP protokole.

U okviru ELK-a, Elasticsearch igra ključnu ulogu kao središnje mjesto za pohranu svih podataka. U kontekstu ovoga rada, to su dnevnički zapisi prikupljeni pomoću Logstash-a, komponente zadužene za njihovo prikupljanje, transformaciju i slanje u Elasticsearch, opisane u poglavlju 4.2.2.

Elasticsearch ne samo da omogućava pohranu podataka već nudi i mogućnosti za njihovu detaljnu analizu i pretragu. Ove mogućnosti, u kombinaciji sa skalabilnošću i brzinom koju pruža, čine Elasticsearch idealnim izborom za upravljanje velikim količinama podataka u stvarnom vremenu (engl. *real-time*).

Indeksi u ElasticSearch-u

Indeks (engl. *index*) u kontekstu ElasticSearch-a jest spremnik za čuvanje podataka (Konda, 2023). Predstavlja kolekciju dokumenata sa sličnim karakteristikama. Indeks je identificiran svojim imenom, koje se koristi prilikom izvršavanja operacija pretrage, ažuriranja, brisanja i indeksiranja na dokumentima. U ovom se radu indeks koristi za organiziranje i skladištenje dnevničkih zapisa omogućavajući njihovu brzu i efikasnu pretragu.

4.2.2. Logstash

Logstash je servis otvorenog koda i predstavlja snažan sustav za obradu podataka. Njegova je glavna svrha izvlačenje podataka iz različitih izvora, njihova transformacija i obogaćivanje te naknadno slanje prema odredištu (Konda, 2023). Zahvaljujući arhitekturi koja je bazirana na obradi tokova podataka, svaki događaj koji prođe kroz Logstash obrađuje se sukladno prethodno konfiguriranim pravilima i u stvarnom vremenu se prosljeđuje na odredište. To omogućava kontinuiranu obradu podataka, što je posebno primjenjivo u kritičnim sustavima, kao što su oni koji se bave analizom grešaka izvučenih iz dnevničkih zapisa.

²Apache Lucene je snažan sustav za pretraživanje teksta otvorenog koda koji se koristi u brojnim aplikacijama za pretragu i analizu podataka.

U okviru ovog rada Logstash igra ključnu ulogu prikupljajući dnevničke zapise iz različitih Apache Kafka tema. Zatim se zapisi upisuju u ElasticSearch indeks, nakon čega se mogu pretraživati i postaju lako dostupni za daljnju analizu.

4.2.3. Kibana

Gormley i Tong (2015) navode da je **Kibana** višenamjenska konzola koja nudi niz opcija, od izvršavanja upita do razvoja nadzornih ploča, grafova i vizualizacija dijagrama za detaljnu analizu i agregacije. Međutim, za komunikaciju s ElasticSearch-om i pozivanje API-ja nije nužno koristiti Kibana. Na primjer, API-ji ElasticSearch-a mogu se pozivati koristeći *cURL*, *Postman* ili klijente podržanog programskog jezika.

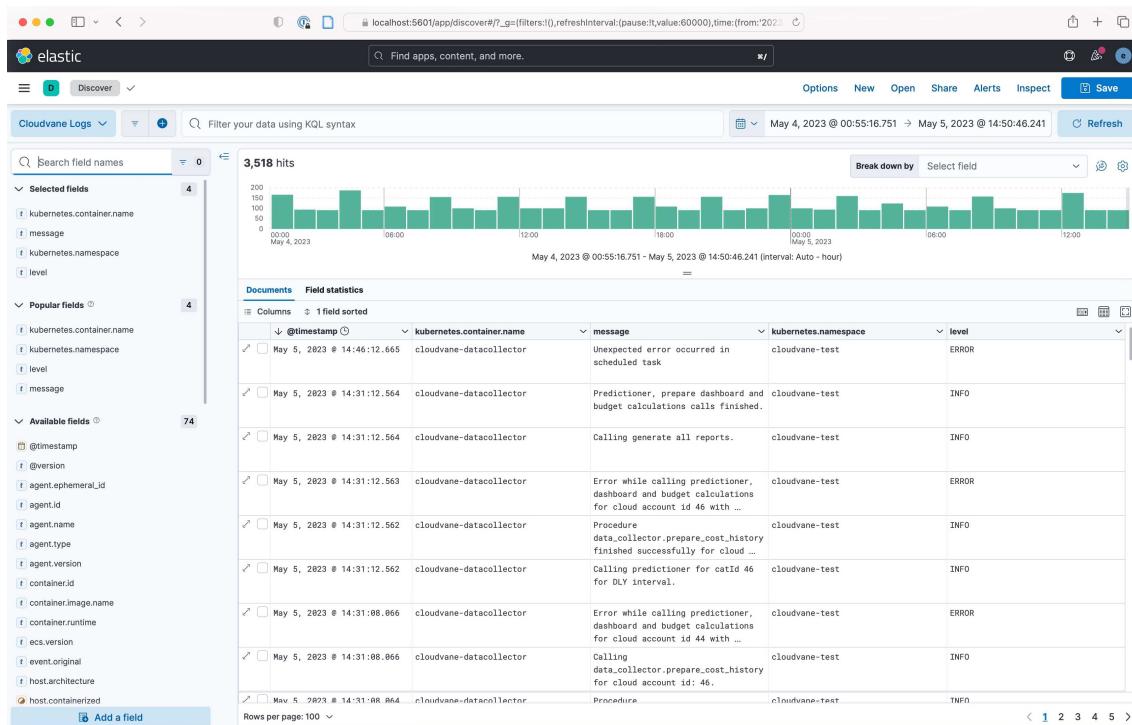
Kibana, kao alat za vizualizaciju i istraživanje podataka, u ovom radu koristi se kao sučelje na kojem se mogu kreirati intuitivni grafikoni i na kojem se mogu filtrirati te pregledavati dnevnički zapisi. Ovaj alat omogućuje interaktivno navigiranje kroz velike količine dnevničkih zapisa. S Kibanom je moguće konfigurirati nadzorne ploče koje pružaju vizualne sažetke podataka i omogućuju detaljno istraživanje specifičnih događaja. Primjer prikaza alata Kibane s dnevničkim zapisima može se vidjeti na slici 4.1.

Kibana pruža mogućnost vizualizacije podataka pohranjenih u ElasticSearch-u koristeći njene snažne mogućnosti upita i filtriranja. Time se omogućuje detaljna analiza specifičnih metrika ili trendova. Ova funkcionalnost je ključna za učinkovito analiziranje i razumijevanje obimnih podatkovnih setova, kao što su dnevnički zapisi koje obrađuje ovaj rad.

4.2.4. Filebeat

Filebeat je alat za prikupljanje podataka čiji je glavni zadatak skupljanje i centralizacija dnevničkih zapisa. Njegova je uloga u ELK-u nadziranje specifične datoteke s dnevničkim zapisima, njihovo prikupljanje i zatim prosljeđivanje izravno u ElasticSearch indeks ili u Logstash za daljnju obradu (Konda, 2023).

Filebeat je samo jedan od mnogih alata u sklopu *Beats* platforme koju je razvila tvrtka Elastic. **Beats** je platforma koja se sastoji od niza alata za prikupljanje podataka, uključujući dnevničke zapise, mrežni promet, metrike i slično. Svi alati unutar Beats platforme razvijeni su s ciljem jednostavnosti i učinkovitosti.



Slika 4.1: Kibana sučelje

Zbog svog dizajna koji naglašava jednostavnost i minimalnu potrošnju resursa, Filebeat je iznimno pogodan za distribuirane okoline poput Kubernetesa.

Iako Logstash nudi značajne mogućnosti obrade podataka, Filebeat je specijaliziran da bude lagan i brz, s naglaskom na efikasno prikupljanje dnevničkih zapisa i njihovo brzo proslijđivanje do odredišta. To znači da se Filebeat može pokrenuti, a da se to ne odrazi značajno na performanse. Filebeat i Logstash često se koriste u tandemu; *Filebeat* je odgovoran za prikupljanje i proslijđivanje dnevničkih zapisa, a *Logstash* ih zatim obrađuje i obogaćuje prije konačnog indeksiranja u ElasticSearch-u.

U okviru ovog rada, Filebeat je pokrenut na CloudVane Kubernetes klasteru u obliku Kubernetes DaemonSet-a te je konfiguriran da prikuplja dnevničke zapise s CloudVane servisa na testnoj okolini. Nakon prikupljanja ti se dnevnički zapisi prosljeđuju preko interneta na lokalni Minikube Kubernetes klaster, točnije u Apache Kafka teme što je detaljnije opisano u poglavljju 5.3.

S obzirom na to da se Filebeat pokreće na Kubernetesu kao DaemonSet, pokreće se u onoliko instanci koliko čvorova ima grozd. Jedna instanca Filebeat-a po čvoru omogućuje Filebeat-u pristup kontejnerima koji se na njemu vrte. Konfiguracija ovoga servisa implementirana je kao Kubernetes ConfigMap i prikazana je u isječku A.5.

4.2.5. Implementacija ELK-a na Kubernetes grozdu

U ovom radu su ElasticSearch, Kibana i Logstash implementirani u lokalnom Kubernetes klasteru koristeći ELK CRD-ove. Elastic, kompanija koja stoji iza razvoja ovih aplikacija, kreirala je CRD-ove za te programe kako bi se olakšala njihova implementacija na Kubernetes klasteru. To omogućava korisnicima da jednostavno postave aplikacije i upravljaju tim aplikacijama unutar Kubernetes okruženja pružajući im integrirane mogućnosti za konfiguraciju, skaliranje, sigurnost i nadzor.

Filebeat je međutim, kao što je prethodno opisano, pokrenut na CloudVane Kubernetes grozdu kao DaemonSet, a njegova konfiguracija je pružena putem ConfigMap objekta.

5. Nadziranje stanja distribuiranog sustava u stvarnom vremenu

Kao što je prethodno spomenuto u poglavlju 4, nadziranje aplikacija ključno je za održavanje optimalnih performansi, stabilnosti i pouzdanosti aplikacija. Strukturirano i sustavno nadziranje može biti presudno za otkrivanje i rješavanje problema prije nego što oni počnu utjecati na korisnike ili poslovanje.

Kako su suvremene aplikacije postale sve složenije, zadatak njihova nadzora postao je izrazito zahtjevan. Ovaj je izazov posebno istaknut kod nadziranja distribuiranih sustava, gdje svaki individualni servis može potencijalno postati točka slabosti ili čak izazvati potpuni kvar sustava.

S takvom razinom kompleksnosti nadziranje distribuiranih sustava prelazi granice jednostavnog praćenja pojedinačnih servisa i zahtijeva upotrebu sveobuhvatnih alata za nadzor. Nadzor nije samo otkrivanje grešaka ili slabosti unutar pojedinačnih servisa. Pored toga, nužno je uspostaviti sistemski pregled, identificirati anomalije u stvarnom vremenu, reagirati pravodobno i učinkovito kako bi se osigurao nesmetan rad distribuiranih sustava.

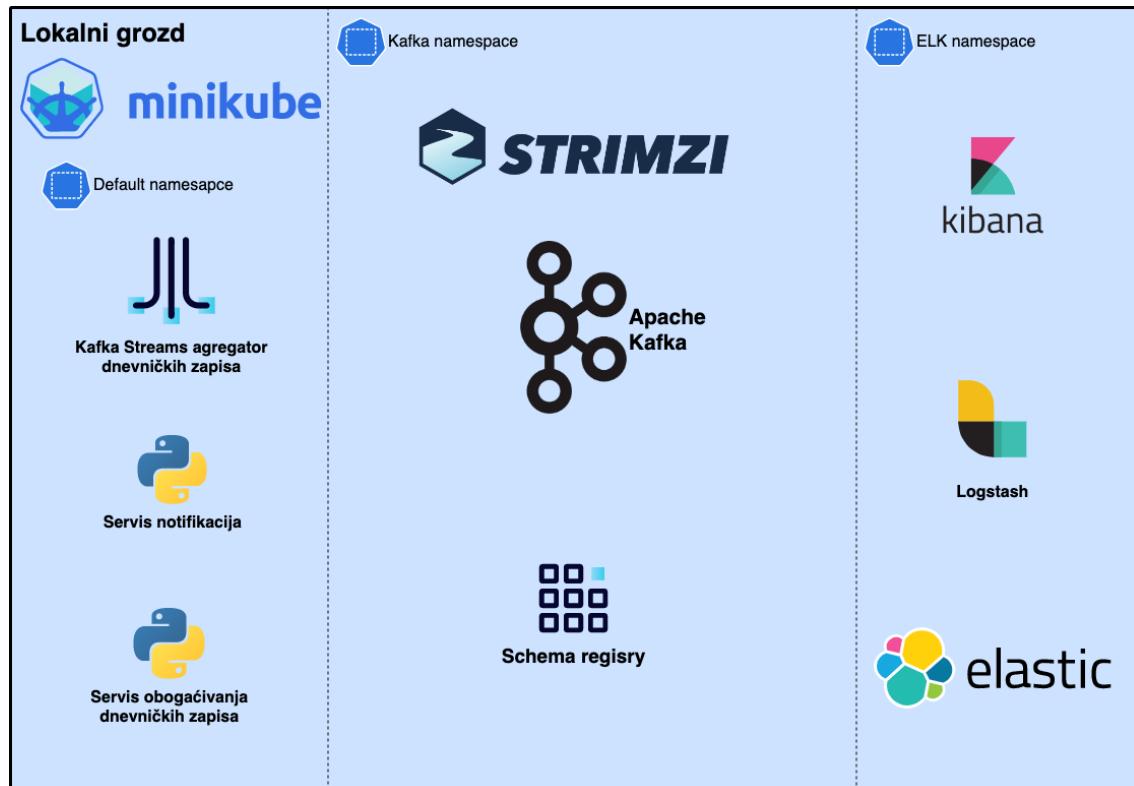
U tom je kontekstu potrebno koristiti napredne alate za nadzor koji omogućuju dubinsko praćenje performansi i stanja svih servisa, bez obzira na to gdje se nalaze. Ti alati trebaju omogućiti brzo otkrivanje problema, detaljne informacije o prirodi problema i omogućiti promptno djelovanje kako bi se osigurali optimalan rad i performanse distribuiranih sustava. Ovakav pristup osigurava visoku razinu pouzdanosti i stabilnosti omogućujući sustavima da stalno pružaju visokokvalitetnu uslugu korisnicima.

U nadolazećim poglavljima rada obrađuje se arhitektura rješenja koja omogućava agregiranje dnevničkih zapisa svih servisa distribuirane aplikacije CloudVane u stvarnom vremenu. Ovo rješenje ne samo da filtrira zapise prema njihovom statusu već koristi i umjetnu inteligenciju kako bi ponudilo moguća rješenja za prepoznate greške.

Time se problem, zajedno s predloženim rješenjima, korisniku prezentira kroz sustav notifikacija. Osim toga, rješenje uključuje vizualizaciju svih relevantnih podataka unutar Kibane, što korisnicima omogućava brzo i intuitivno razumijevanje stanja njihove aplikacije. Ovo rješenje omogućuje korisnicima da brzo reagiraju na potencijalne probleme, čime se unapređuje ukupna stabilnost i performanse distribuiranih aplikacija.

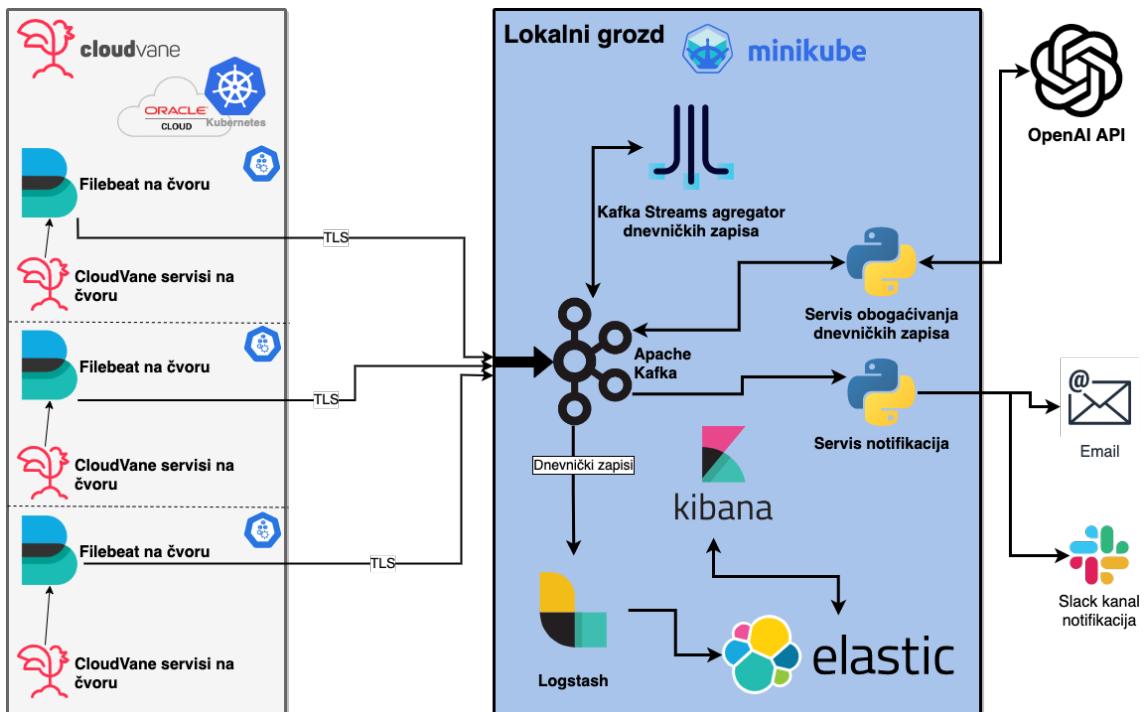
5.1. Pregled arhitekture rješenja

Kako je već spomenuto u prethodnim poglavljima, arhitektura rješenja, koje je predmet ovog rada, vidljiva na slici 5.1, sastoji se od više komponenata koje zajedno formiraju sveobuhvatni sustav za nadzor distribuiranih aplikacija. Ovaj sustav, koji je implementiran na lokalnom Minikube Kubernetes grozdu te djelomično na CloudVane Kubernetes grozdu, omogućuje kontinuirano praćenje, analizu i rješavanje problema unutar aplikacija.



Slika 5.1: Arhitektura sustava

Na CloudVane Kubernetes grozdu je pokrenut **Filebeat** servis koji je odgovoran za prikupljanje dnevničkih zapisa svih servisa aplikacije. Kako je opisano u poglavljju 4.2.4, Filebeat dnevničke zapise zatim šalje preko interneta na teme Apache Kafka koja je pokrenuta u lokalnom Minikube grozdu. Budući da se ovi podaci prenose preko interneta, Apache Kafka je zaštićena TLS-om¹ kako bi se osigurao siguran prijenos podataka. Na slici 5.2 vidljiv je protok dnevničkih zapisa kroz sustav.



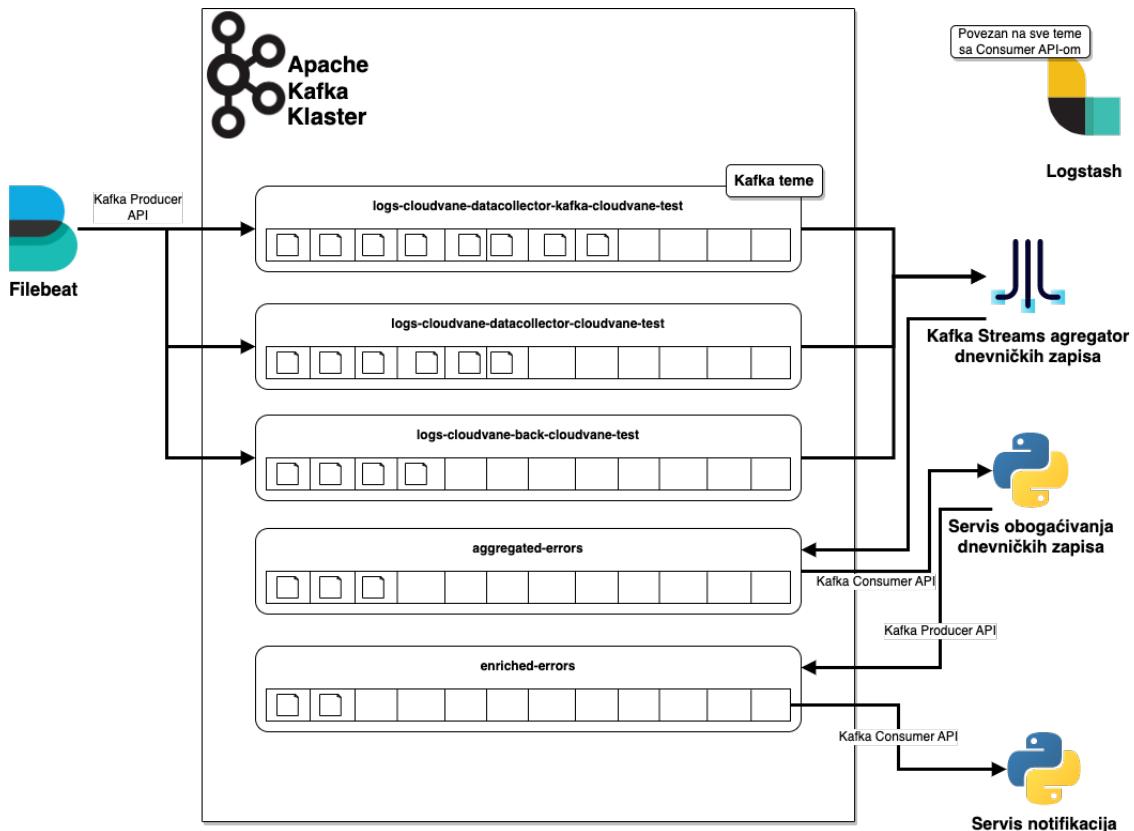
Slika 5.2: Tok podataka (engl. *data pipeline*) sustava

Na lokalnom Kubernetes Minikube grozdu pokrenuta je **Apache Kafka** pomoću alata **Strimzi**. Uz jedan Apache Kafka posrednik² (engl. *broker*) pokrenut preko alata Strimzi, pokrenut je i Schema Registry servis za upravljanje shemama podataka. U poglavljju 5.3 opisani su implementacijski detalji lokalne instance Apache Kafke i Kafka teme koje se koriste za pohranu dnevničkih zapisa.

Nakon Apache Kafke na lokalnom je Kubernetes grozdu pokrenuta i distribuirana **Kafka Streams aplikacija** koja filtrira i agregira dnevničke zapise. Ova aplikacija je detaljno opisana u poglavljju 5.4. Agregirani dnevnički zapisi zatim se šalju natrag u nove, agregirane Kafka teme, a taj je protok vidljiv na dijagramu 5.3.

¹TLS (Transport Layer Security) je protokol koji osigurava zaštitu podataka tijekom prijenosa preko interneta.

²S obzirom na to da se Minikube sastoji samo od jednoga čvora, nema razloga pokretati više replika posrednika (engl. *brokers*).



Slika 5.3: Kafka teme (engl. *topics*) s dnevničkim zapisima (engl. *logs*)

Paralelno s tim, **servis za obogaćivanje dnevničkih zapisova** pokrenut je na Kubernetes grozdu i opisan je u poglavljju 5.5. Ovaj servis je dizajniran da bude skalabilan i prilagodljiv, ovisno o potrebama sustava. Za svaku grešku koja se javlja u CloudVane aplikaciji i koja se detektira u dnevničkom zapisu, ovaj servis komunicira s API-jem usluge umjetne inteligencije kako bi pronašao njeno potencijalno rješenje. Svaka detektirana greška, zajedno sa sugeriranim rješenjem, zatim se šalje u novu Kafka temu kao što je vidljivo u dijagramu 5.3.

Na sve teme koje sadrže dnevničke zapise spojen je **Logstash servis**, opisan u poglavljju 4.2.2 i prikazan u dijagramu 5.2. Logstash skuplja dnevničke zapise iz svih tema, a zatim ih proslijeđuje u **ElasticSearch** indeks. Ova funkcionalnost omogućuje korisnicima da putem **Kibane** iz poglavља 4.2.3, imaju cijelovit uvid u dnevničke zapise, bilo da su agregirani ili ne. Implementacija pregleda dnevničkih zapisa kroz Kibanu demonstrirana je u poglavljju 5.7.

Konačno, kako bi korisnici bili u stvarnom vremenu informirani o svim greškama koje se javljaju u aplikaciji, implementiran je i **servis za notifikacije**. Ovaj servis, opisan u poglavljju 5.6 i vizualiziran u dijagramu 5.2, obavještava korisnike o svim de-

tektiranim problemima putem različitih kanala komunikacije uključujući elektroničku poštu, mobilne notifikacije i Slack kanal.

Ova visoko integrirana arhitektura omogućuje proaktivni pristup upravljanju distribuiranim aplikacijama, omogućujući korisnicima da brzo reagiraju na probleme i osiguraju maksimalnu učinkovitost i pouzdanost svojih aplikacija.

5.2. Kubernetes infrastruktura rješenja

Kao što je prethodno istaknuto, i prema dijagramu 5.1, sve komponente ovog rješenja implementirane su na lokalnom Kubernetes Minikube klasteru. Jedini izuzetak je Filebeat servis, koji je pokrenut na CloudVane Kubernetes klasteru, što je detaljnije objašnjeno u poglavlju 4.2.4.

Na Kubernetes klasteru stvorena su dva dodatna prostora imena (engl. *Namespace*). Prvi, pod imenom „*kafka*“, sadrži Kafka servise Strimzi operatora, Schema Registry, Kafka UI te prateće Service objekte za mrežnu komunikaciju s Kafkom. Drugi, imenovan „*elk*“, sadrži ELK servise, uključujući Logstash, Kibana i ElasticSearch. U *zadanom* (engl. *default*) prostoru imena nalazi se Kafka Streams aplikacija, uz koju je pokrenut i servis za obogaćivanje i servis za notifikacije.

Sve usluge koje su pokrenute na Minikube klasteru konfiguirirane su s jednom replikom, s obzirom na to da se Minikube klaster sastoji od jednog čvora, kao što je navedeno u poglavlju 3.1.3. U producijskom okruženju broj replika može se povećati jednostavnim izmjenama parametara u konfiguraciji Kubernetes objekata.

Za potrebe pristupa Apache Kafka posredniku s interneta, bilo je nužno konfigurati Strimzi Kafka objekt i omogućiti pristup preko TLS-a. Dodatno, bilo je potrebno kreirati Kafka User objekt na temelju kojeg Strimzi generira kredencijale za autentifikaciju u obliku Secret objekta. Ti se kredencijali zatim izvlače i kao novi Secret pohranjuju u CloudVane Kubernetes klaster omogućavajući Filebeat servisu slanje dnevničkih zapisa u Minikube klaster.

ELK alati, uključujući ElasticSearch, Filebeat i Kibana, također generiraju kredencijale za pristup u obliku Secret objekta. Iz Secret objekta je potrebno izvući te podatke kako bi se omogućila autentifikacija u Kibantu. Logstash primjenjuje konfiguraciju za čitanje iz Kafka tema i pisanje u ElasticSearch u obliku ConfigMap objekta, čiji je sa-

držaj prikazan u isječku A.4.

Konačno, servisi za obogaćivanje, notifikacije i Kafka Streams aplikacija pokrenuti su kao *Deployment* objekti. Servis za obogaćivanje i servis za notifikacije dobivaju konfiguraciju i tajne (API ključeve i druge kredencijale) u obliku ConfigMap i Secret objekata. Ove aplikacije imaju *Dockerfile* u svom izvornom kodu, prikazan za primjer Kafka Streams aplikacije u isječku A.6, koji opisuje postupak kreiranja kontejnera iz izvornog koda. Slike kontejnera se nakon izgradnje pakiraju i objavljuju na Docker Hub repozitoriju, što omogućuje Kubernetes klasteru da ih preuzme i pokrene prilikom kreiranja Deployment objekta.

5.3. Pohrana dnevničkih zapisa u Apache Kafku

Kada Filebeat servis prikupi dnevničke zapise s CloudVane servisa i prenese ih u Apache Kafku koja je postavljena na lokalnom Kubernetes Minikube klasteru, proces rezultira stvaranjem specifičnih tema koje su vidljive kroz Kafka UI alat, kako je prikazano na slici 5.4. Filebeat za svaku uslugu, čije dnevničke zapise prikuplja, kreira zasebnu temu. Ovaj pristup omogućava dobro strukturirano i organizirano skladištenje dnevničkih zapisa.

Kada se podaci zabilježe u Apache Kafku, svaki konzument poruka može se povezati na relevantnu temu i čitati poruke. U okviru ovog rada, konzumenti poruka izvornih dnevničkih zapisa su Logstash i Kafka Streams aplikacija.

Kasnije u procesu, prema dijagramu toka podataka 5.3, vidljivo je kako se servisi za obogaćivanje i notifikacije također povezuju na određene teme kako bi čitali poruke. Servis za obogaćivanje dodatno upisuje obogaćene podatke u svoju temu. Ovaj proces omogućava dnevničkim zapisima da prođu kroz niz transformacija prije nego što se konzumiraju, pružajući tako dodatne vrijednosti.

Sve relevantne Kafka teme detaljno su opisane u tablici 5.1, u kojoj je naveden i opis podataka koji se nalaze u svakoj od njih. Ova tablica pruža korisne informacije o tome što svaka tema predstavlja, čime olakšava razumijevanje strukture i svrhe svake teme.

UI for Apache Kafka fdd9ad9 5/11/2023, 16:02:15

	Topic Name	Partitions	Out of sync replicas	Replication Factor	Number of messages	Size	Actions
□	logs-other-kafka-test	1	0	1	233062	99 MB	...
□	logs-cloudvane-kafka-kafka-test	1	0	1	133305	83 MB	...
□	logs-cloudvane-predictioner-cloudvane-test	1	0	1	72042	32 MB	...
□	enriched-errors	1	0	1	669	4 MB	...
□	IN __consumer_offsets	50	0	1	25587	2 MB	...
□	logs-cloudvane-scheduler-cloudvane-test	1	0	1	4243	2 MB	...
□	aggregated-errors	1	0	1	372	2 MB	...
□	logs-cloudvane-vault-cloudvane-test	1	0	1	3472	2 MB	...
□	logs-cloudvane-datacollector-kafka-cloudvane-test	1	0	1	2660	2 MB	...
□	logs-cloudvane-config-cloudvane-test	1	0	1	1336	681 KB	...
□	logs-cloudvane-datacollector-cloudvane-test	1	0	1	716	513 KB	...
□	logs-cloudvane-back-cloudvane-test	1	0	1	249	312 KB	...
□	logs-other-cloudvane-test	1	0	1	547	238 KB	...
□	logs-cloudvane-actioner-cloudvane-test	1	0	1	45	27 KB	...
□	IN __strimzi_store_topic	1	0	1	32	11 KB	...
□	IN __strimzi-topic-operator-kstreams-topic-store-changelog	1	0	1	32	4 KB	...
□	IN __schemas	1	0	1	8	944 Bytes	...
□	error_aggregator_kstream-KSTREAM-AGGREGATE-STATE-STOR...	1	0	1	0	0 Bytes	...

Slika 5.4: Kafka teme (engl. *topics*) na lokalnom Apache Kafka grozdu kroz pogled Kafka UI servisa

Tablica 5.1: Kafka teme s dnevničkim zapisima

Ime Kafka teme	Sadržaj podataka u temi
logs-cloudvane-*-cloudvane-test	Dnevnički zapisi pojedinog CloudVane mikro-servisa
aggregated-errors	Agregirani dnevnički zapisi grešaka koje upisuje Kafka Streams aplikacija
enriched-errors	Agregirani dnevnički zapisi grešaka s dodatkom o potencijalnom rješenju koje upisuje Python servis koji obogaćuje poruku greške pomoću umjetne inteligencije
error_aggregator_kstream_*	Interna tema Kafka Streams aplikacije za spremanje podataka potrebnih za agregacije
_consumer_offsets	Interna tema Apache Kafke za spremanje pozicije čitanja
_schemas	Interna tema za Schema Registry servis za pohranu shema
_strimzi*	Interne teme Strimzi servisa

5.4. Agregiranje dnevničkih zapisova

Dnevnički zapisi mogu biti itekako brojni, posebno u kompleksnim sustavima. Na primjer, u CloudVane testnom okruženju može se generirati i do 15000 dnevničkih zapisa u samo jednom satu. Velik broj dnevničkih zapisa često je simptom nekog problema unutar sustava, gdje se greške javljaju i sustav započne s generiranjem obavijesti o problemima.

No, problemi se rijetko rješe sami od sebe. Sustav obično nastavlja generirati dnevničke zapise sve dok se problem ne riješi. To može rezultirati velikim količinama dnevničkih zapisa koje treba obraditi i analizirati kako bi se razumjelo što se točno događa unutar sustava.

Svaki dnevnički zapis u stanju je pružiti dragocjene informacije o različitim aspektima sustava, uključujući performanse, potencijalne greške, ponašanje korisnika i ostale vitalne elemente operativnosti sustava. No, dnevnički zapisi često sadrže poruke

o greškama koje se iznimno često ponavljaju u kratkim vremenskim intervalima stvarajući time veliku količinu redundantnih informacija.

Upravo u ovom kontekstu **Kafka Streams** biblioteka postaje iznimno korisna. Kafka Streams, opisana u poglavlju 3.2.4, agregiranjem sličnih ili identičnih dnevničkih zapisa može drastično smanjiti količinu podataka koju treba obraditi zadržavajući ključne informacije o greškama i problemima unutar sustava.

Na taj se način omogućuje obrada agregiranih informacija, što omogućuje brže i efikasnije identificiranje te rješavanje problema. Ovaj pristup ne samo da smanjuje kompleksnost i trošak analize dnevničkih zapisa već i olakšava otkrivanje dubljih uzoraka i trendova, omogućavajući proaktivno rješavanje problema prije nego što uzrokuju ozbiljnije posljedice.

Izrađena Kafka Streams aplikacija, čiji je algoritam prikazan pseudokodom 1 i čija je topologija ilustrirana na dijagramu 5.5, uključuje sljedeće korake:

1. Dnevnički zapisi se konzumiraju iz svih Kafka tema koje započinju ključnom riječi „*logs*“.
2. Dnevnički zapisi se filtriraju temeljem statusa koji određuje jesu li to informativni dnevnički zapisi ili predstavljaju greške. U ovom kontekstu izostavljaju se svi dnevnički zapisi koji ne predstavljaju greške s obzirom na to da je cilj korisniku pružiti reakciju na greške unutar aplikacije.
3. Dnevnički zapisi grupiraju se prema ključu Kafka poruke unutar kojeg se nalazi sadržaj greške. (Ključ Kafka poruke je definiran u konfiguraciji Filebeat servisa koja je prikazana u isječku A.5)
4. Agregacija dnevničkih zapisa provodi se po ključu poruke. Ovo grupiranje se izvodi korištenjem *klizećeg prozora* u trajanju od *30 sekundi*.
5. Agregacija koju izravno pruža Kafka Streams biblioteka mapira se na objekt koji sadrži broj agregiranih poruka, kao i prvu dohvaćenu vremensku oznaku.
6. Agregirani dnevnički zapisi se pohranjuju u Kafka temu „*aggregated-errors*“.

Algorithm 1 Kafka Streams aplikacija

Uzorak: *logs – .* – regularni izraz Kafka tema koje treba agregirati.*

Izlaz: *aggregated – errors Kafka tema s filtriranim i agregiranim dnevničkim zapisima.*

for (*topic* in *topics*) **do**

stream := *streamBuilder.stream(topic)*

filteredStream := *stream.filter(isValidErrorMessage())*

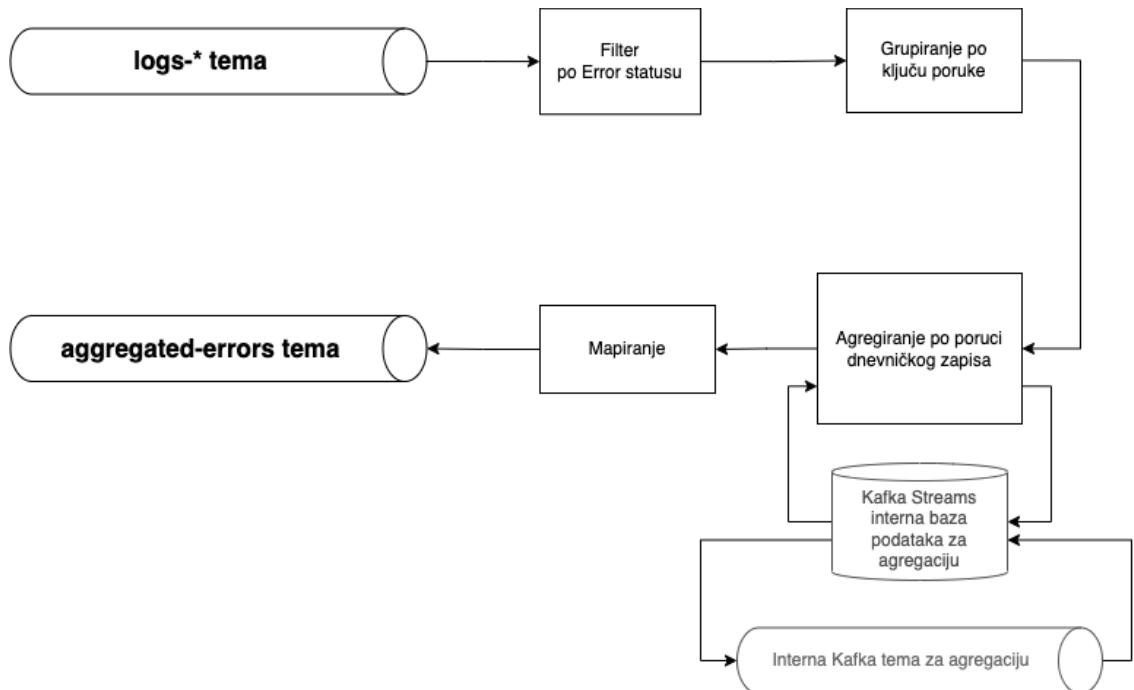
aggregatedStream := *filteredStream.groupByKey()*

.windowedBy(SlidingWindow, 30sec).aggregate()

mapped := *aggregatedStream.map(mapToResult())*

mapped.to(outputTopic)

end for



Slika 5.5: Topologija Kafka Streams aplikacije

5.5. Obogaćivanje dnevničkih zapisu

Prilikom prvog susreta s greškama unutar sustava često je nejasno u čemu je problem. Iz izravnih dnevničkih zapisu uzrok problema i njegovo potencijalno rješenje često nisu odmah očigledni.

U svrhu rješavanja ovog problema, ovaj rad istražuje poseban Kafka servis, izrađen u Pythonu, koji koristi *OpenAI API* kako bi obogatio prethodno agregirane dnevničke zapise o greškama. Cilj je ovog obogaćivanja pružati dodatne informacije korisnicima koji su zaduženi za rješavanje problema, čime se pojačava njihova sposobnost razumevanja i brzog reagiranja na probleme.

Posebna značajka ovog servisa je njegova sposobnost horizontalnog skaliranja, omogućena putem Kafka Consumer API-ja. Ova karakteristika omogućuje servisu da se prilagodi povećanju opterećenja, čime se osigurava održavanje stabilnih performansi čak i pod visokim zahtjevima.

U procesu obogaćivanja servis koristi *OpenAI API*, čiji je opis sadržan u sljedećem poglavlju, za analizu grešaka. Na temelju njegovog odgovora servis generira korisne informacije koje će korisnicima pomoći da bolje razumiju problem.

Kao što je vidljivo u dijagramu 5.3, ovaj servis čita dnevničke zapisu iz Kafka teme „*aggregated-errors*“, obogaćuje ih koristeći OpenAI API, a zatim ih šalje u novu Kafka temu pod nazivom „*enriched-errors*“.

Kroz ovaj proces servis ne samo da poboljšava korisnikovo razumijevanje i reakciju na greške već i pomaže u poboljšanju ukupne kvalitete i pouzdanosti sustava.

Primjer dnevničkog zapisu obogaćenog umjetnom inteligencijom prikazan je u isječku A.7. Polje „*instructions*“ generirano je umjetnom inteligencijom na temelju agregiranih dnevničkih zapisu.

5.5.1. OpenAI API

OpenAI API predstavlja sučelje koje omogućava pristup sofisticiranim algoritmima strojnog učenja koje je razvila organizacija *OpenAI*. Ovi algoritmi sposobni su za razumijevanje, generiranje i interpretiranje prirodnog jezika, što ih čini iznimno korisnima za širok spektar primjena. U kontekstu ovog rada ti se algoritmi koriste za obogaćivanje informacija o greškama u dnevničkim zapisima.

Za potrebe ovog rada korišten je OpenAI model pod nazivom *text-davinci-003*. No, važno je naglasiti da se prethodno opisani servis može konfigurirati za upotrebu bilo kojeg dostupnog modela OpenAI API-ja.

Model *text-davinci-003* jedan je od najmoćnijih modела za obradu prirodnog jezika koje je OpenAI razvio. Ovaj model je treniran na širokom skupu podataka, što mu omogućava da razumije i generira koherentan i kontekstualno relevantan tekst temeljen na dobivenim ulaznim podacima. Takva sposobnost modela omogućuje visokokvalitetno obogaćivanje informacija sadržanih u dnevničkim zapisima.

Komunikacija s OpenAI API-jem može se ostvariti kroz HTTP protokol ili kroz biblioteke napisane za popularne programerske jezike. U ovom radu odabrana je direktna komunikacija putem HTTP protokola s OpenAI API-jem.

5.6. Notifikacije u stvarnom vremenu

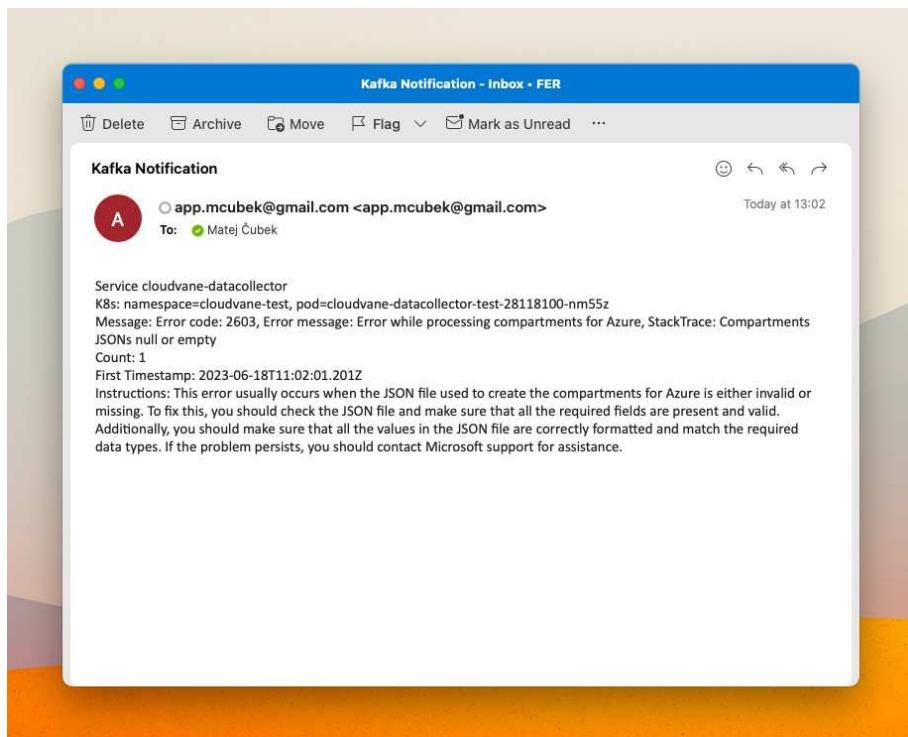
Problem s identifikacijom grešaka u sustavima može biti izazovan. Kada greška nastane, često nije odmah evidentno da određeni servis ima problem i da je njegov rad narušen. Ovaj izazov može otežati pravovremeno rješavanje problema, održavanje pouzdanosti i performansi sustava.

Kako bi se riješio ovaj problem, za potrebe rada je razvijen poseban servis. Ovaj servis također je izrađen u Pythonu i ima mogućnost horizontalnog skaliranja zahvaljujući mogućnostima koje pruža Kafka Consumer API. Ova značajka omogućuje servisu da se prilagodi promjenjivim zahtjevima za performanse, omogućujući mu da efikasno obrađuje velike količine podataka i održava visoku razinu usluge.

Ovaj servis služi kao sustav za obavještavanje, poslužujući se različitim kanalima kako bi korisnike obavijestio o nastalim greškama u sustavu. To se postiže slanjem notifikacija putem elektroničke pošte, objavama na Slack kanalu te slanjem obavijesti putem Pushover aplikacije. Ovaj višekanalni pristup obavještavanja osigurava da relevantne informacije o problemima dođu do korisnika na najbrži i najefikasniji mogući način.

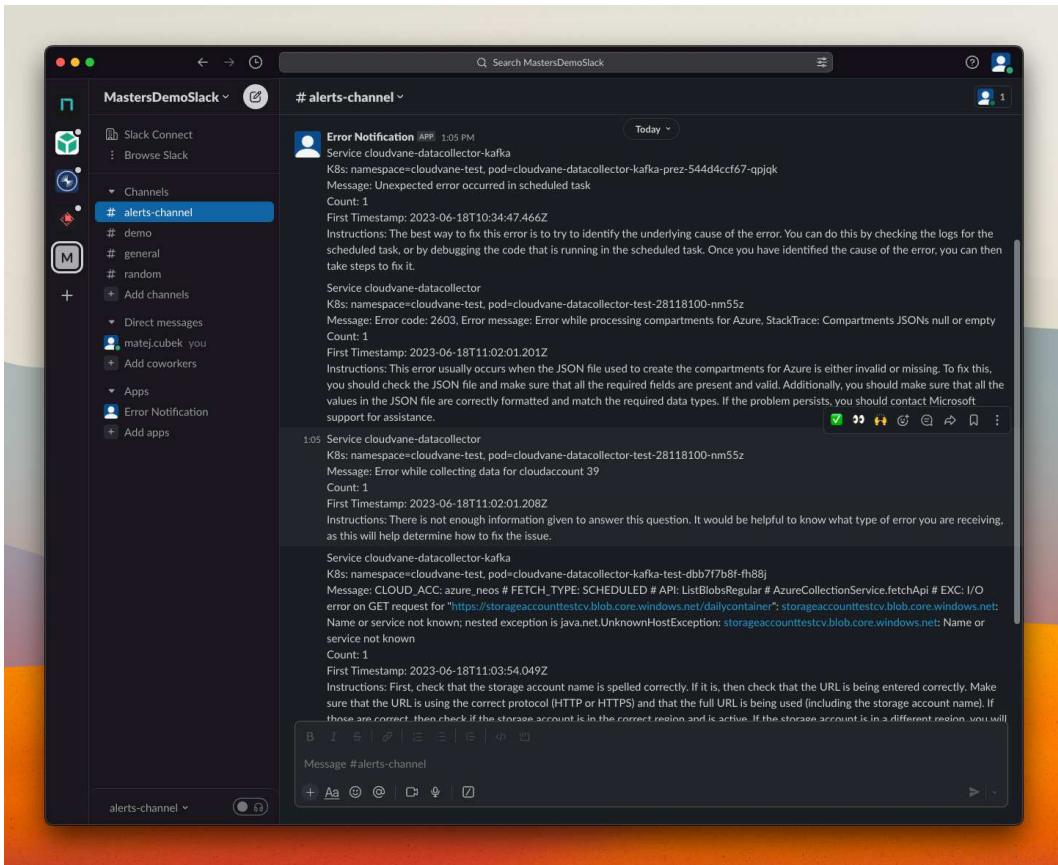
Svaka obavijest sadrži ključne informacije o grešci, uključujući vrijeme kada je greška prvi put identificirana, opis problema prema informacijama iz dnevničkih zapisu, kao i potencijalno rješenje problema dobiveno putem OpenAI API-ja. Ove informacije pomažu korisnicima da brzo shvate prirodu problema olakšavajući time proces popravka.

Dnevnički zapisi koji se isporučuju u notifikacijama čitaju se iz Kafka teme „*enriched-errors*“ kao što je vidljivo na dijagramu 5.3. Primjeri notifikacija prikazani su na sljedećim slikama: notifikacija elektroničke pošte nalazi se na slici 5.6, Slack notifikacija na slici 5.7, a Pushover mobilna notifikacija na slici 5.8.

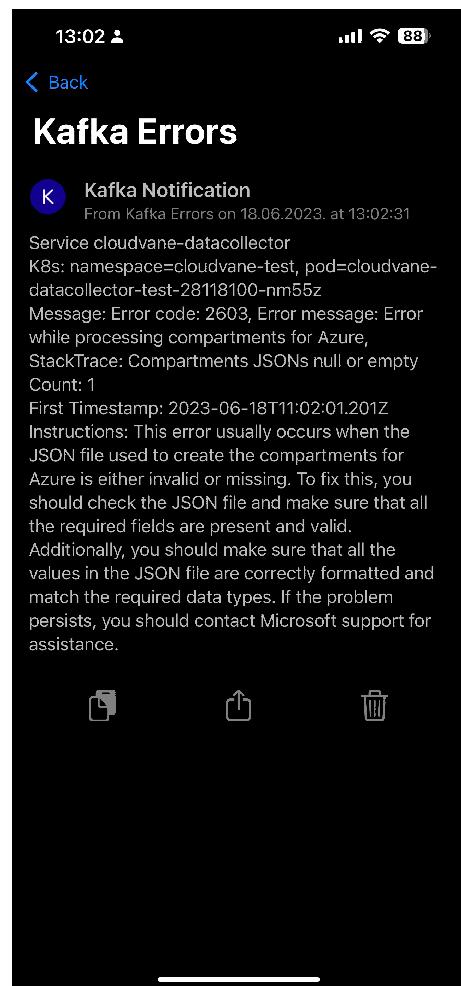


Slika 5.6: Primjer notifikacije dnevničkog zapisa greške putem elektroničke pošte

Ovaj servis je ključan za održavanje stabilnosti sustava omogućavajući pravovremeno reagiranje na greške, smanjenje vremena neaktivnosti i poboljšanje ukupne kvalitete usluge.



Slika 5.7: Primjer notifikacije dnevničkog zapisa greške putem Slack aplikacije

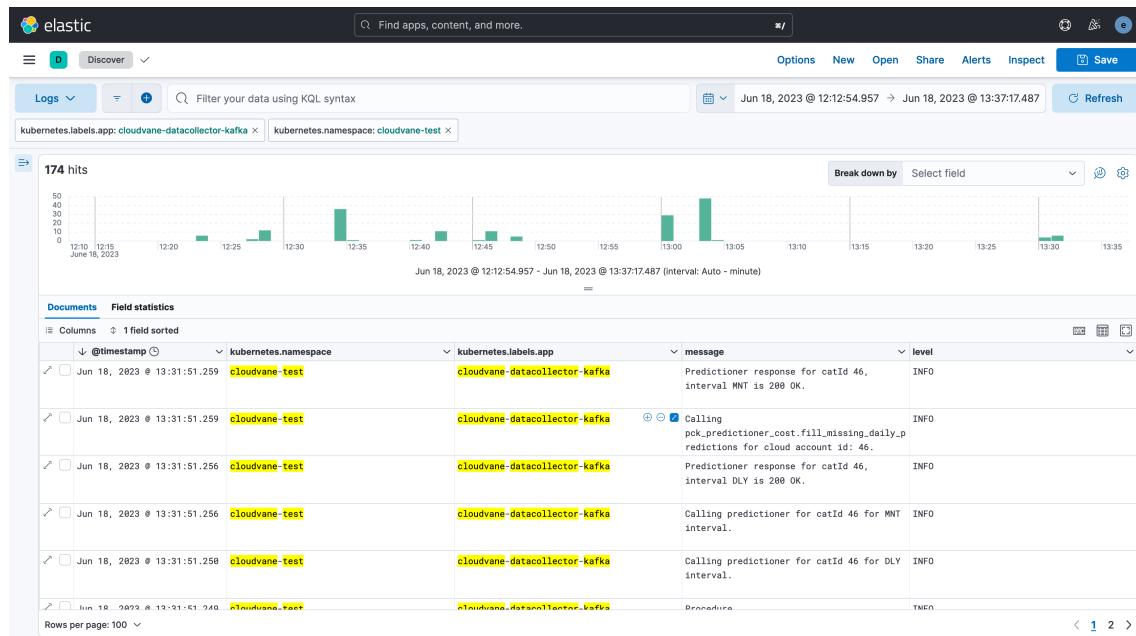


Slika 5.8: Primjer notifikacije dnevničkog zapisa greške putem Pushover mobilne aplikacije

5.7. Pregled kroz Kibantu

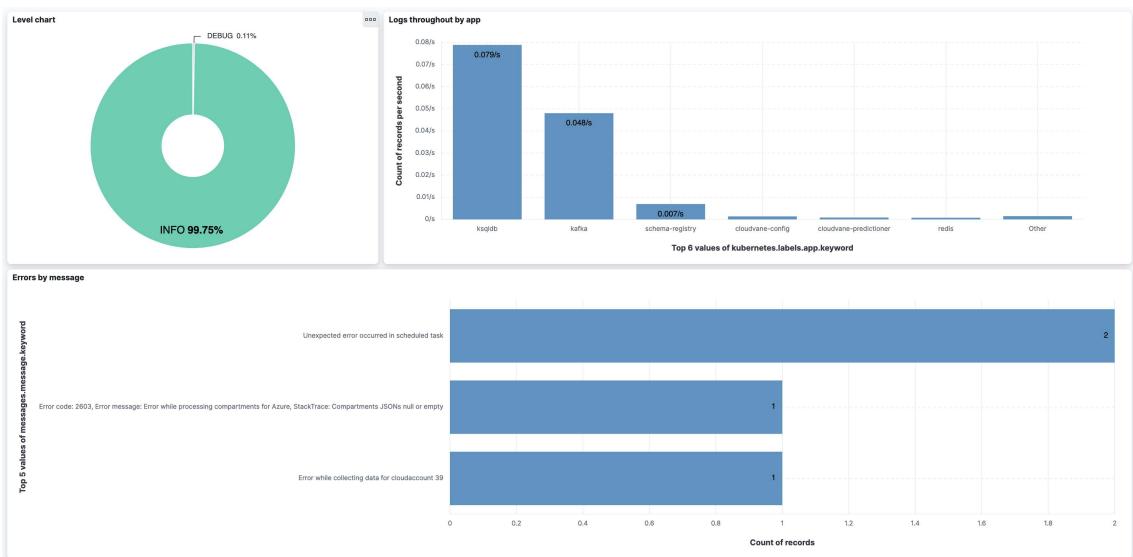
Naposljetku, ako dođe do greške u aplikaciji i treba ju popraviti, potrebno je pregledati stanja svih servisa. Pregled stanja servisa u većini slučajeva uključuje pregled dnevničkih zapisa, međutim to, kao što je prethodno opisano, u distribuiranoj okolini nije jednostavan zadatak. Zbog velikog broja servisa za potpun pregled sustava potrebno je koristiti alat poput Kibane koji agregira dnevničke zapise (engl. *logs*), a uvod u nju dan je u poglavlju 4.2.3.

Kibana, sučelje podataka ElasticSearch-a, omogućuje pregled dnevničkih zapisa svih servisa sustava kroz objedinjeno sučelje i omogućuje njihovo filtriranje po svim dostupnim atributima poput čvora grozda, imena servisa, prostora imena itd., kao što je prikazano na slici 5.9.



Slika 5.9: Prikaz filtriranih dnevničkih zapisa (engl. *logs*) u Kibantu sučelju

Kibana također omogućuje i kreiranje kontrolnih ploča s ključnim metrikama kao što je prikazano na slici 5.10. Ove kontrolne ploče pružaju brzi pregled performansi i stanja sustava te mogu pomoći u brzom identificiranju i rješavanju problema.



Slika 5.10: Prikaz kreirane kontrolne ploče u Kibani

6. Zaključak

U doba digitalizacije i sve veće ovisnosti o informacijskim sustavima efikasno i pravovremeno praćenje dnevničkih zapisa postaje presudno za pravilan rad i održavanje aplikacija. U ovom radu prikazano je istraživanje kako se platforme *Apache Kafka* i *ELK*, uz pomoć *OpenAI API-ja*, mogu iskoristiti za obradu i analizu dnevničkih zapisa raspodijeljenih sustava u stvarnom vremenu.

ELK platforma koja uključuje *ElasticSearch*, *Logstash* i *Kibana*, zajedno s *Apache Kafka* platformom koristila se za obradu, prikupljanje i prikaz dnevničkih zapisa. *Kafka Streams* biblioteka korištena je za agregaciju dnevničkih zapisa, a *OpenAI API* za obogaćivanje zapisa, što omogućuje dublju analizu i razumijevanje informacija prikupljenih iz sustava.

Na primjeru SaaS aplikacije *CloudVane* tvrtke *Neos* demonstriralo se kako je moguće primijeniti opisane metode i tehnologije za nadzor dnevničkih zapisa u složenim distribuiranim sustavima. Rezultati istraživanja pokazali su da je korištenjem navedenih alata moguće detektirati i nadzirati anomalije u radu sustava na učinkovit način.

Rad je također pokazao kako se korištenjem *Apache Kafka* platforme i *ELK* platforme mogu unaprijediti pouzdanost i brzina obrade podataka, što omogućuje pravovremeno reagiranje i pregled problema sustava.

Nadalje, kao nastavak na ovaj rad otvaraju se mogućnosti za buduća istraživanja. Primjerice, moguće je izgraditi servis koji bi, ovisno o detektiranoj grešci na koju je pretplaćen, mogao pokrenuti odgovarajuću skriptu. Moguće je integrirati servis *Grafana* za vizualizaciju sustava. Također, postoji prostor za poboljšanje postojećih procesa, unapređenje automatizacije te ugradnju modela predikcije koji bi mogao koristiti kontekst prošlih pojavljenih dnevničkih zapisa.

LITERATURA

Kumar Chandrakant. Kafka's Shift from ZooKeeper to Kraft | Baeldung, Studeni 2022. URL <https://www.baeldung.com/kafka-shift-from-zookeeper-to-kraft>.

Inc CloudFlare. Why use serverless computing? | Pros and cons of serverless, 2023. URL <https://www.cloudflare.com/learning/serverless/why-use-serverless/>.

Meredith Courtemanche, Emily Mell, i Alexander Gillis. What Is DevOps? The Ultimate Guide, Prosinac 2021. URL <https://www.techtarget.com/searchitoperations/definition/DevOps>.

Finops Finops Fundation. FinOps Foundation - What is FinOps?, Studeni 2021. URL <https://www.finops.org/introduction/what-is-finops/>.

Alexander Gillis. What Are Containers (Container-based Virtualization or Containerization)?, Ožujak 2020. URL <https://www.techtarget.com/searchitoperations/definition/container-containerization-or-container-based-virtualization>.

Clinton Gormley i Zachary Tong. *Elasticsearch: The Definitive Guide*. O'Reilly, 2015. ISBN 978-1-4493-5854-9. Google-Books-ID: 2tQBoQEACAAJ.

Chandler Harris. Microservices vs. monolithic architecture, 2023. URL <https://www.atlassian.com/microservices/microservices-architecture/microservices-vs-monolith>.

Patrick Hunt, Mahadev Konar, Flavio P Junqueira, i Benjamin Reed. ZooKeeper: Wait-free coordination for Internet-scale systems. 2010.

Marcin Kolny. Scaling up the Prime Video audio/video monitoring service and reducing costs by 90%, Ožujak 2023. URL

<https://www.primevideotech.com/video-streaming/scaling-up-the-prime-video-audio-video-monitoring-service-and-reducing-latency/>
Section: Video Streaming.

Madhusudhan Konda. *Elasticsearch in Action, Second Edition*. Manning, Svibanj 2023. ISBN 978-1-61729-985-8. Google-Books-ID: c7OyzgEACAAJ.

Marko Lukša. *Kubernetes in Action, Second Edition*. Manning, Kolovoz 2023. ISBN 978-1-61729-761-8. Google-Books-ID: _YrwzgEACAAJ.

Monitorama. Monitorama PDX 2016 - Greg Poirier - Monitoring is Dead. Long Live Monitoring, Srpanj 2016. URL <https://vimeo.com/173610062>.

Neha Narkhede. Exactly-once Semantics is Possible: Here's How Apache Kafka Does it, Lipanj 2017. URL <https://www.confluent.io/blog/exactly-once-semantics-are-possible-heres-how-apache-kafka-does-it/>

Neos. CloudVane, 2023. URL <https://www.neos.hr/solution/cloudvane/>.

Sam Newman. *Building Microservices*. O'Reilly Media, Veljača 2015. ISBN 978-1-4919-5035-7. Google-Books-ID: 1uUDoQEACAAJ.

Inc Red Hat. What is containerization?, Travanj 2021. URL <https://www.redhat.com/en/topics/cloud-native-apps/what-is-containerization>.

Mitch Seymour. *Mastering Kafka Streams and ksqlDB*. "O'Reilly Media, Inc.", Veljača 2021. ISBN 978-1-4920-6246-2.

Gwen Shapira, Todd Palino, Rajini Sivaram, i Krit Petty. *Kafka: The Definitive Guide*. O'Reilly Media, Studeni 2021. ISBN 978-1-4920-4303-4. Google-Books-ID: MstMEAAAQBAJ.

Andrew S. Tanenbaum i Maarten van Steen. *Distributed systems: principles and paradigms*. Pearson Prentice Hall, Upper Saddle River, NJ, 2nd ed izdanju, 2007. ISBN 978-0-13-239227-3. OCLC: ocm70707891.

Matej Čubek. Apache Kafka Seminar. 2022.

Ivana Podnar Žarko, Krešimir Pripužić, Ignac Lovrek, i Mario Kušek. *Raspodijeljeni Sustavi - Radna inačica udžbenika*, svezak v1.3. 2013.

Dodatak A

Isječci programskog koda

Isječak koda A.1: Primjer dnevničkog zapisa CloudVane servisa

```
{  
    "@timestamp": "2023-06-11T20:32:51.032Z",  
    "@version": "1",  
    "message": "Calling  
        data_collector.prepare_cost_history for cloud  
        account id: 76.",  
    "logger_name":  
        "hr.neos.cloudvane.datacollector.predictioner.  
        PredictionerService",  
    "thread_name": "scheduling-1",  
    "level": "INFO",  
    "level_value": 20000  
}
```

Isječak koda A.2: Konfiguracija Logback-a

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
<contextName>cloudvane-dc-kafka</contextName>

<!-- Include CONSOLE_LOG_PATTERN -->
<include
    resource="org/springframework/boot/logging/logback/defaults.xml"/>

<appender name="CONSOLE_PATTERN"
    class="ch.qos.logback.core.ConsoleAppender">
<layout class="ch.qos.logback.classic.PatternLayout">
<Pattern>${CONSOLE_LOG_PATTERN}</Pattern>
</layout>
</appender>

<appender name="CONSOLE_JSON"
    class="ch.qos.logback.core.ConsoleAppender">
<encoder class="net.logstash.logback.encoder.LogstashEncoder"
    />
</appender>

<springProfile name="local">
<root level="INFO">
<appender-ref ref="CONSOLE_PATTERN" />
</root>
</springProfile>

<springProfile name="dev,dev-test,test,prod">
<root level="INFO">
<appender-ref ref="CONSOLE_JSON" />
</root>
</springProfile>
</configuration>
```

Isječak koda A.3: Primjer dnevničkog zapisa sa CloudVane servisa s metapodacima

```
{  
    "@timestamp": "2023-05-03T17:00:04.068Z",  
    "@metadata": {  
        "beat": "filebeat",  
        "type": "_doc",  
        "version": "8.7.1"  
    },  
    "log": {  
        "offset": 294991,  
        "file": {  
            "path":  
                "/var/log/containers/cloudvane-datacollector..."  
        }  
    },  
    "level_value": 20000,  
    "logger_name":  
        "hr.neos.cloudvane.datacollector.collector.cloud.azure.  
AzureCollectionService",  
    "ecs": {  
        "version": "8.0.0"  
    },  
    "host": {  
        "ip": [  
            "10.244.1.90",  
            "fe80::8bf:80ff:fe5f:3832"  
        ],  
        "mac": [  
            "0A-BF-80-5F-38-32"  
        ],  
        "hostname": "filebeat-kafka-matej-44gl2",  
        "architecture": "x86_64",  
        "os": {},  
        "containerized": true,  
        "name": "filebeat-kafka-matej-44gl2"  
    }
```

```

} ,
"stream": "stdout",
"input": {
  "type": "container"
},
"instana.trace.id": "adf2108f84007f73",
"level": "INFO",
"container": {
  "id": "a248de8e52ca95d0591cd220cd4f
438488e4765bc1de51644c4c7621784b9e45",
  "runtime": "cri-o",
  "image": {
    "name": "fra.ocir.io/frpf7jvyycgx/cv-kafka/openjdk-17/
cloudvane-datacollector:1fc43..."
  }
},
"message": "Fetching AZURE cost file list from URL:
https://storageaccounttestcv.blob.core.windows.net/
dailycontainer?restype=container&comp=list",
"agent": {
  "ephemeral_id": "5c71c9f5-3073-4b71-90a8-04ef13492db4",
  "id": "18a62338-ce41-4a14-9d6d-e9f6887041b0",
  "name": "filebeat-kafka-matej-44gl2",
  "type": "filebeat",
  "version": "8.7.1"
},
"@version": "1",
"thread_name": "scheduling-1",
"kubernetes": {
  "labels": {
    "app": "cloudvane-datacollector-kafka",
    "client": "test",
    "pod-template-hash": "75f86b6d7c"
  }
},

```

```

"container": {
    "name": "cloudvane-datacollector"
},
"node": {
    "labels": { },
    "hostname": "10.38.1.11",
    "name": "10.38.1.11",
    "uid": "9c999e81-cafc-405c-936c-8c45183351b5"
},
"pod": {
    "ip": "10.244.1.5",
    "name": "cloudvane-datacollector-kafka-test-75f86b6d7c-g2kgb",
    "uid": "1a2ff384-ac41-47d0-815d-43c27591cff7"
},
"namespace": "cloudvane-test",
"namespace_uid":
    "4af72ed8-18a8-4d35-ad03-966d1804e9c7",
"namespace_labels": {
    "kubernetes_io/metadata_name": "cloudvane-test"
},
"replicaset": {
    "name":
        "cloudvane-datacollector-kafka-test-75f86b6d7c"
}
},
"indexName": "cloudvane-datacollector-kafka"
}

```

Isječak koda A.4: Konfiguracija Logstash-a na lokalnom Kubernetes grozdu

```
input {
    kafka {
        bootstrap_servers =>
            "masters-cluster-kafka-bootstrap.kafka.svc.cluster.local:9092"
        topics_pattern => "^logs.*"
        group_id => "logstash"
        codec => "json"
        decorate_events => "extended"
    }
}

filter {
    mutate {
        add_field => { "topic" => "%{[@metadata][kafka][topic]}" }
    }
}

output {
    elasticsearch {

        hosts =>
            ["https://masters-es-internal-http.elk.svc.cluster.local:9200"]
        index => "k8s-%{topic}"

        ssl => true
        ssl_certificate_verification => false
        user => "elastic"
        password => "ELASTIC_PASSWORD"

        ilm_enabled => false
    }
}
```

Isječak koda A.5: Konfiguacija Filebeat-a na CloudVane Kubernetes grozdu

```
filebeat.inputs:
- type: container
format: cri
paths:
- /var/log/containers/*.log
processors:
- decode_json_fields:
fields: [ "message" ]
target: ""
overwrite_keys: true
- add_kubernetes_metadata:
in_cluster: true
host: ${NODE_NAME}
matchers:
- logs_path:
logs_path: /var/log/containers/
- script:
lang: javascript
source: >
function process(event) {
    var appName=event.Get("kubernetes.labels.app");
    if (appName!==null && appName.indexOf("cloudvane") !== -1)
    {
        event.Put("indexName",event.Get("kubernetes.labels.app"));
    }
    else if (appName!==null && appName.indexOf("kafka") !== -1)
    {
        event.Put("indexName","cloudvane-".concat(event.Get("kubernetes.labels.
    }
    else
    {
        event.Put("indexName", "other");
    }
var input = event.Get("message");
input = input.replace(/RequestId:[^ ]*/g, " ");
input = input.replace(/Time:[^ ]*/g, " ");
input = input.replace(/\d{4}-\d{2}-\d{2}/
```

```

    \d{2}:\d{2}:\d{2}\.\d{3}/g, "");
input = input.replace(/\d{4}-\d{2}-\d{2}
    \d{2}:\d{2}:\d{2},\d{3}/g, "");
var kafkaKey = {
  "message": input,
  "service": event.Get("indexName"),
  "kubernetes": {
    "namespace": event.Get("kubernetes.namespace"),
    "pod": event.Get("kubernetes.pod.name")
  }
};
event.Put("kafkaKey", JSON.stringify(kafkaKey));
}

- drop_event:
when:
not:
or:
- equals:
  kubernetes.namespace: "cloudvane-test"
- equals:
  kubernetes.namespace: "kafka-test"
- type: log
paths:
- /var/log/containers/*.out
processors:
- decode_json_fields:
  fields: [ "message" ]
  target: ""
  overwrite_keys: true
- script:
  lang: javascript
  source: >
    function process(event) {
      event.Put("indexName", "usage");
      var kafkaKey = {
        "message": event.Get("message"),
        "service": event.Get("indexName"),
      };

```

```
    event.Put("kafkaKey", JSON.stringify(kafkaKey));
}

processors:
- add_host_metadata:
setup.template:
name: k8s
pattern: k8s-*
setupilm.enabled: false
output.kafka:
hosts: [ "MY_PUBLIC_IP:30096" ]
version: '2.6.0'
key: '%{[kafkaKey]}'
topic: 'logs-%{[indexName]}-%{[kubernetes.namespace]}'
partition.round_robin:
reachable_only: false
required_acks: 1
batch.max_events: 2048
compression: snappy
max_message_bytes: 1000000
ssl.enabled: true
ssl.certificateAuthorities: [ '/etc/filebeat/ssl/ca.pem' ]
ssl.certificate: '/etc/filebeat/ssl/certificate.pem'
ssl.key: '/etc/filebeat/ssl/key.pem'
```

Isječak koda A.6: Dockerfile Kafka Streams aplikacije

```
FROM openjdk:17
LABEL authors="matejc"

ENV PROFILE=local

WORKDIR /app
COPY build/libs/error_aggregator_stream-1.0.0.jar
    /app/log_aggregation_kstream.jar

EXPOSE 8080

ENTRYPOINT ["java",
    "-jar", "-Dspring.profiles.active=${PROFILE}"
    , "/app/log_aggregation_kstream.jar"]

USER 1002
```

Isječak koda A.7: Primjer obogaćenog dnevničkog zapisa CloudVane servisa

```
{  
  "kafkaKey": {  
    "message": "CLOUD_ACC: azure_neos # FETCH_TYPE:  
               SCHEDULED # API: ListBlobsAmortized #  
               AzureCollectionService.fetchApi # EXC: I/O error  
               on GET request for  
               \\"https://storageaccounttestcv.blob.core.windows.net/  
               dailycontainer\\":  
               storageaccounttestcv.blob.core.windows.net;  
               nested exception is java.net.UnknownHostException:  
               storageaccounttestcv.blob.core.windows.net",  
    "service": "cloudvane-datacollector-kafka",  
    "kubernetes": {  
      "namespace": "cloudvane-test",  
      "pod":  
        "cloudvane-datacollector-kafka-test-dbb7f7b8f-fh88j"  
    }  
  },  
  "kafkaValue": {  
    "count": 1,  
    "messages": [  
      {  
        "@timestamp": "2023-06-18T01:04:20.397Z",  
        "@metadata": {  
          "beat": "filebeat",  
          "type": "_doc",  
          "version": "8.7.1"  
        },  
        "level": "ERROR",  
        "kubernetes": { ... },  
        "pod": { ... },  
        "namespace": "cloudvane-test",  
        "namespace_uid":  
          "4af72ed8-18a8-4d35-ad03-966d1804e9c7",  
        "service": "cloudvane-datacollector-kafka",  
        "type": "Filebeat",  
        "version": "8.7.1"  
      }  
    ]  
  }  
}
```

```

    "namespace_labels": {
        "kubernetes_io/metadata_name": "cloudvane-test"
    },
    "replicaset": {
        "name": "cloudvane-datacollector-kafka-test-dbb7f7b8f"
    },
    "labels": {
        "app": "cloudvane-datacollector-kafka",
        "client": "test",
        "pod-template-hash": "dbb7f7b8f"
    },
    "container": {
        "name": "cloudvane-datacollector"
    }
},
"ecs": {
    "version": "8.0.0"
},
"thread_name": "scheduling-1",
"indexName": "cloudvane-datacollector-kafka",
"kafkaKey": "{\"message\": \"CLOUD_ACC: azure_neos
# FETCH_TYPE: SCHEDULED # API:
ListBlobsAmortized #
AzureCollectionService.fetchApi # EXC: I/O
error on GET request for
\\\\\"https://storageaccounttestcv.blob.core.windows.net/
dailycontainer\\\\\":
storageaccounttestcv.blob.core.windows.net;
nested exception is
java.net.UnknownHostException:
storageaccounttestcv.blob.core.windows.net\", \"service\":
\"cloudvane-datacollector-kafka\", \"kubernetes\":
{ \"namespace\": \"cloudvane-test\", \"pod\": \"cloudvane-datacollector-kafka-test-dbb7f7b8f-fh88j\" } }",

```

```

"host": { ... },
"input": {
    "type": "container"
},
"stack_trace": [
    "org.springframework.web.client.ResourceAccessException:
        I/O error on GET request for ...",
"@version": "1",
"logger_name": [
    "hr.neos.cloudvane.datacollector.collector.cloud
        .azure.AzureCollectionService",
"container": { ... },
"agent": [
    "version": "8.7.1",
    "ephemeral_id": [
        "3c5a96f4-29dc-4e57-9a47-b8b7120752d5",
        "id": "e474b345-9e7c-4449-900d-c625d654d9d2",
        "name": "filebeat-kafka-matej-sfhmk",
        "type": "filebeat"
    ],
    "stream": "stdout",
    "message": "CLOUD_ACC: azure_neos # FETCH_TYPE:
        SCHEDULED # API: ListBlobsAmortized #
        AzureCollectionService.fetchApi # EXC: I/O
        error on GET request for
        \"https://storageaccounttestcv.blob.core.windows.net
        /dailycontainer \":"
    storageaccounttestcv.blob.core.windows.net;
    nested exception is java.net.UnknownHostException:
    storageaccounttestcv.blob.core.windows.net",
    "level_value": 40000,
    "log": { ... },
    "count": 1
}
],
"firstTimestamp": "2023-06-18T01:04:20.397Z",

```

```
    "instructions": "This error indicates that the  
    hostname  
    storageaccounttestcv.blob.core.windows.net could  
    not be found. This could be due to a typo in the  
    hostname, or it could be that the hostname is not  
    registered in the DNS. Check to ensure that the  
    hostname is correct, and if it is, then you may  
    need to register the hostname in the DNS."  
}  
}
```

Analiza dnevničkih zapisa raspodijeljenih sustava u stvarnom vremenu

Sažetak

U radu naslovljenom „Analiza dnevničkih zapisa raspodijeljenih sustava u stvarnom vremenu“ istražene su mogućnosti Apache Kafka platforme i komponenti ELK-a (ElasticSearch, Logstash i Kibana) za obradu i analizu dnevničkih zapisa raspodijeljenih mikro-servisnih aplikacija koje se izvode na Kubernetesu u stvarnom vremenu. Rad koristi OpenAI API za obogaćivanje dnevničkih zapisa s dodatnim kontekstom i informacijama. Implementirani su mehanizmi za stvarno-vremenske notifikacije i agregaciju dnevničkih zapisa koristeći Kafka Streams, što omogućuje pravovremeno otkrivanje i reagiranje na anomalije u sustavu. Korištenjem SaaS aplikacije CloudVane tvrtke Neos kao primjera, pokazano je kako se ove tehnologije mogu primijeniti u nadzoru dnevničkih zapisa u složenim distribuiranim sustavima. Ovaj rad naglašava važnost dnevničkih zapisa za otkrivanje anomalija i nadzor performansi te ulogu Apache Kafka i ELK platforme u tom kontekstu.

Ključne riječi: Kubernetes, Apache Kafka, Tokovi Podataka, Dnevnički zapisi, Obrada u stvarnom vremenu, Elastic, ELK, Nadzor aplikacija, Grozd računala, Logstash, Filebeat, ElasticSearch, Kibana, Minikube, Oblak

Real-Time Log Analysis for Monitoring Distributed Systems

Abstract

In the thesis titled „Real-time Log Analysis of Distributed Systems“, the potential of Apache Kafka platform and ELK stack components (ElasticSearch, Logstash, and Kibana) for real-time processing and analysis of distributed microservice applications running on Kubernetes was explored. The study utilizes the OpenAI API for enriching log files with additional context and information. The thesis introduces mechanisms for real-time notifications and log aggregations using Kafka Streams enabling timely detection and response to system anomalies. Using Neos's SaaS application CloudVane as an example, the study demonstrates how these technologies can be implemented in monitoring log files in complex distributed systems. This work emphasizes the importance of log files for anomaly detection and performance monitoring, as well as the role of Apache Kafka and ELK platforms in this context.

Keywords: Kubernetes, Apache Kafka, Streaming, Real-Time, Logs, Elastic, ELK, Monitoring, Cluster, Logstash, Filebeat, ElasticSearch, Kibana, Minikube, Cloud