



UNIVERZITET U SARAJEVU
ELEKTROTEHNIČKI FAKULTET
ODSJEK ZA RAČUNARSTVO I INFORMATIKU

Kontejnerizacijske i orkestracijske tehnike za upravljanje bazama podataka pomoću DevOps alata

ZAVRŠNI RAD
- DRUGI CIKLUS STUDIJA -

Student:
Ahmed Pašić

Mentor:
Vanr. prof. dr Emir Buza

Sarajevo,
juni 2024.

Sažetak

Ovaj rad se bavi upravljanjem bazama podataka primjenom kontejnerizacijskih i orkestracijskih tehnika. U njemu je analizirano trenutno stanje u razmatranoj oblasti istraživanja. Također prikazane su osnovne ideje DevOps metodologije te su detaljnije opisana dva veoma značajna principa iste: kontejnerizacija i orkestracija. U praktičnom dijelu urađena je demonstracija na koji način se može upravljati bazama podataka u jednom takvom okruženju te je izvršena analiza sa nabrojanim prednostima i manama.

Rad ima za cilj da čitaocu približi sam koncept te da prikaže moguće alternative koje nudi DevOps kao princip.

Ključne riječi: kontejnerizacija, orkestracija, Kubernetes, DevOps, PostgreSQL

Abstract

This thesis deals with database management through the application of containerization and orchestration techniques. It analyzes the current state of research in this area. The basic ideas of the DevOps methodology are presented, with detailed descriptions of two significant principles: containerization and orchestration. The practical part demonstrates how databases can be managed in such an environment and includes an analysis of the advantages and disadvantages. The aim is to familiarize the reader with the concept and to showcase the alternatives offered by DevOps as a principle.

Keywords: containerization, orchestration, Kubernetes, DevOps, PostgreSQL

Postavka zadatka završnog rada II ciklusa:
Kontejnerizacijske i orkestracijske tehnike za upravljanje bazama
podataka pomoću DevOps alata

Tradicionalni način razvoja softvera se fokusira na planiranju izdavanja verzija sa velikim izmjenama. Ovakav pristup može dovesti do većih rizika zbog veličine promjena. DevOps s druge strane, praktikuje drugačiji način razvoja. Ovaj princip je baziran na razvoju sa čestim manjim promjenama. Njegovo korištenje može znatno olakšati i ubrzati razvoj cjelokupnog programskog rješenja. DevOps prakse podstiču saradnju i komunikaciju između programera, operacija i drugih zainteresovanih strana, poboljšavajući ukupnu efikasnost i kvalitet isporuke softvera. Posljednjih godina, DevOps je postao sve popularniji i za razvoj baza podataka. Osim lakšeg procesa razvoja, dodatno omogućava i efikasnije održavanje sistema jer čini nadzor, skaliranje kao i praćenje metrika vezanih za korištene baze mnogo lakšim i bržim. Kroz ovaj rad će biti detaljnije objašnjen način primjene DevOps pristupa pri upravljanju bazama podataka, pri čemu će na primjeru biti objašnjen način uspostavljanja same baze u okruženju, način pristupa bazi, mogućnost skaliranja baze podataka u situacijama kada imamo veću količinu saobrajača prema istoj i način evidentiranja i praćenja tih procesa. Pri tome će biti korišteni danas popularni DevOps alati poput: Docker, Kubernetes, Grafana, PGWatch itd.

Polazna literatura:

- [1] Insights, G. M. (2023) Devops market size by component, service, by deployment model, by organization, application global forecast, 2023 - 2032, dostupno na:
<https://www.gminsights.com/industry-analysis/devops-market>
- [2] Carter, M. (2021) Docker index shows momentum in developer community activity, dostupno na:
<https://www.docker.com/blog/docker-index-shows-surging-momentum-in-developer-community-activity-again/>
- [3] Bass, L., Weber, I., Zhu, L., DevOps: A Software Architect's Perspective. Addison-Wesley Professional, 2015.
- [4] Shahin, M., Ali Babar, M., Zhu, L., "Continuous integration, delivery and deployment: A systematic review on approaches, tools, challenges and practices", IEEE Access, Vol. 5, 2017, str. 3909-3943.
- [5] Turnbull, J., The Art of Monitoring, 2014.

- [6] Humble, J., “Continuous integration, delivery and deployment: A systematic review on approaches, tools, challenges and practices”, in IEEE Software Engineering Workshop (SEW), 2015.
- [7] Atlassian, “Devops”, dostupno na: <https://www.atlassian.com/devops> 2023.
- [8] Hildred, T., “The history of containers”, dostupno na: <https://www.redhat.com/en/blog/history-containers> 2015.
- [9] Rad, B. B., Bhatti, H. J., Ahmadi, M., “An introduction to docker and analysis of its performance”, International Journal of Computer Science and Network Security (IJC-SNS), Vol. 17, No. 3, 2017, str. 228.
- [10] Sayfan, G., Mastering kubernetes. Packt Publishing Ltd, 2017.
- [11] Moravcik, M., Kontsek, M., “Overview of docker container orchestration tools”, in 2020 18th International Conference on Emerging eLearning Technologies and Applications (ICETA). IEEE, 2020, str. 475–480.

Vanr. prof. dr. Emir Buza

Izjava o autentičnosti radova
Završni rad
II ciklusa studija

Ime i prezime: Ahmed Pašić

Naslov rada: Kontejnerizacijske i orkestracijske tehnike za upravljanje bazama podataka pomoću DevOps alata

Vrsta rada: Završni rad drugog ciklusa studija

Broj stranica: 64

Potvrđujem:

- da sam pročitao dokumente koji se odnose na plagijarizam, kako je to definirano Statutom Univerziteta u Sarajevu, Etičkim kodeksom Univerziteta u Sarajevu i pravilima studiranja koja se odnose na I i II ciklus studija, integrirani studijski program I i II ciklusa i III ciklus studija na Univerzitetu u Sarajevu, kao i uputama o plagijarizmu navedenim na web stranici Univerziteta u Sarajevu;
- da sam svjestan univerzitetskih disciplinskih pravila koja se tiču plagijarizma;
- da je rad koji predajem potpuno moj, samostalni rad, osim u dijelovima gdje je to naznaceno;
- da rad nije predat, u cjelini ili djelimično, za stjecanje zvanja na Univerzitetu u Sarajevu ili nekoj drugoj visokoškolskoj ustanovi;
- da sam jasno naznačio prisustvo citiranog ili parafraziranog materijala i da sam se referirao na sve izvore;
- da sam dosljedno naveo korištene i citirane izvore ili bibliografiju po nekom od preporučenih stilova citiranja, sa navođenjem potpune reference koja obuhvata potpuni bibliografski opis korištenog i citiranog izvora;
- da sam odgovarajuće naznačio svaku pomoć koju sam dobio pored pomoći mentora i akademskih tutora/ica.

Sarajevo, 23. 06. 2024.

Potpis:

Ahmed Pašić

Sadržaj

Popis slika	vii
1 Uvod	1
1.1 Obrazloženje teme	1
1.2 Pregled stanja u oblasti istraživanja	2
1.2.1 "DevOps: A Software Architect's Perspective"	2
1.2.2 "Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices"	2
1.2.3 "Performance of containerized database management systems"	3
1.2.4 "Enabling Containerized, Parametric and Distributed Database Deployment and Benchmarking as a Service"	3
1.2.5 "Evaluation of Container Orchestration Systems for Deploying and Managing NoSQL Database Clusters"	4
1.2.6 "The Art of Monitoring"	4
1.3 Motivacija istraživanja	5
1.4 Osnovni ciljevi i plan istraživanja	5
1.5 Metodologija istraživanja	6
1.6 Očekivani rezultati	6
2 DevOps	8
2.1 Razlike između tradicionalnih pristupa razvoju i DevOps-a	9
2.2 Kontejnerizacija	10
2.2.1 Uvod u kontejnerizaciju	10
2.2.2 Docker	11
2.3 Orkestracija	15
2.3.1 Uvod u orkestraciju	15
2.3.2 Alati za kontejnersku orkestraciju	16
2.3.3 Kubernetes	17
3 Poređenje DevOps i tradicionalnog pristupa pri upravljanju bazama podataka	25
3.1 Generalno poređenje DevOps i tradicionalnog pristupa	25
3.1.1 Principi i metode rada	25
3.1.2 Ciljevi i rezultati	26
3.1.3 Prednosti i izazovi	26
3.2 Poređenje DevOps i tradicionalnog pristupa pri upravljanju bazama podataka	26
3.2.1 Principi i metode rada	26
3.2.2 Ciljevi i rezultati	26
3.2.3 Prednosti i izazovi	27

4 Primjer korištenja kontejnerizacijskih i orkestracijskih tehnika za upravljanje DB serverom	28
4.1 Priprema okruženja	29
4.2 Stavljanje aplikacija u upotrebu	31
4.2.1 Helm	32
4.2.2 ArgoCD	33
4.2.3 PostgreSQL	37
4.3 Grafana	39
4.4 PGWatch2 i InfluxDB	41
4.5 Kubernetes operator	45
4.5.1 Mane korištenja opisanog pristupa	45
4.6 Rad sa bazom podataka u klasteru	49
4.6.1 Pristup bazi podataka preko aplikacije	49
4.6.2 Direktan rad nad bazom kroz pgAdmin4 alat	54
5 Analiza postignutih rezultata	58
5.1 Skup sa stanjem vs. Kubernetes operatori za baze podataka	58
5.2 Baza podataka u Kubernetesu?	59
5.3 Najbolje prakse za DBMS na Kubernetesu	59
Literatura	62

Popis slika

2.1	Životni ciklus razvoja softvera koristeći DevOps pristup [1]	8
2.2	Arhitektura Docker sistema	12
2.3	Arhitektura virtualnih mašina	13
2.4	Arhitektura Docker kontejnera	14
2.5	Alati za kontejnersku orkestraciju [2]	17
2.6	Kubernetes arhitektura [3]	19
2.7	Kubernetes resursi [4]	22
4.1	Pokretanje klastera - Docker Desktop	30
4.2	Izgled Lens aplikacije	31
4.3	Prosljeđivanje porta u Lens alatu	35
4.4	ArgoCD alat	36
4.5	Izgled prozora za PostgreSQL u ArgoCD-u	37
4.6	Informacije o PostgreSQL kontejneru	39
4.7	Početni prozor alata Grafana	40
4.8	Arhitektura programa pgwatch2 [5]	42
4.9	InfluxDB izvor podataka	44
4.10	Predefinirana PGWatch2 tabla sa grafovima	44
4.11	Predefinirane kontrolne table za PGO	48
4.12	Paneli za stanje poda instance servera baze podataka	48
4.13	Paneli za upite nad određenom bazom	49
4.14	Kubernetes tajna sa pristupnim podacima za bazu	50
4.15	Pregled novog stanja u bazi izvršavanjem upita	52
4.16	Dodavanje vlasnika kroz grafički interfejs	53
4.17	Provjera stanja u bazi nakon dodavanja vlasnika	53
4.18	Kreiranje nove baze podataka	54
4.19	Kreiranje tabele za čuvanje slika	54
4.20	Dodavanje slike u tabelu	55
4.21	Prikaz sadržaja tabele "images"	55
4.22	Promjena reda u tabeli "images"	56
4.23	Brisanje reda iz tabele "images"	56
4.24	Izvršeni upiti nad bazom prikazani u Grafana panelu	57

Poglavlje 1

Uvod

U ovom poglavlju prikazani su osnovni motivi kao i ciljevi pisanja rada. U uvodu će biti razmotreni sljedeći elementi:

- Obrazloženje teme
- Pregled stanja u oblasti istraživanja
- Motivacija istraživanja
- Osnovni ciljevi i plan istraživanja
- Metodologija istraživanja
- Očekivani rezultati
- Struktura disertacije

U narednom tekstu će detaljnije biti obrazložena svaka od tačaka.

1.1 Obrazloženje teme

U današnjem tehnološki dinamičnom okruženju, kontinuirana isporuka softvera postala je ključna komponenta za uspješno poslovanje i konkurentnost organizacija. DevOps prakse, koje kombinuju razvoj i operacije softvera u jedan integriran proces, predstavljaju ključni faktor u ostvarivanju brze, stabilne i pouzdane isporuke aplikacija. Međutim, upravljanje bazama podataka često predstavlja izazov u ovom okruženju, jer zahtijeva posebnu pažnju i planiranje kako bi se osigurala konzistentnost, skalabilnost i bezbjednost podataka. Kontejnerizacija i orkestracija, kao metodologije koje se proučavaju kroz ovaj rad, upravo omogućavaju lakše upravljanje bazama podataka u sistemima koji zahtijevaju brzu isporuku kao i stalnu dostupnost. Uz to, rad omogućava detaljniji uvid u metode najbolje prakse pri radu sa bazama podataka, što uveliko može poboljšati cjelokupni rad sa bazama u današnjim sistemima koji pretežno implementiraju različite vidove DevOps metodologije.

Postojeći istraživački radovi uglavnom su usmjereni na primjenu DevOps praksi u razvoju i isporuci softvera, a manje pažnje je posvećeno primjeni ovih praksi na upravljanje bazama podataka. Stoga, ova tema je od suštinskog značaja kako bi se bolje razumjelo kako kontejnerizacijske i orkestracijske tehnike mogu poboljšati upravljanje bazama podataka, uključujući aspekte kao što su automatsko skaliranje, postizanje visoke dostupnosti i brza obnova u slučaju neuspjeha.

Ovaj rad će istražiti prednosti i izazove primjene kontejnerizacije i orkestracije za upravljanje bazama podataka u okviru DevOps praksi, kao i različite alate i tehnologije koji su dostupni za ove svrhe. Također, bit će razmotrone najbolje prakse i preporuke za implementaciju ovih tehnika u cilju doprinosa daljem razvoju DevOps praksi u kontekstu baza podataka.

1.2 Pregled stanja u oblasti istraživanja

U proteklih nekoliko godina, DevOps pristupi i tehnike, kao što su kontejnerizacija i orkestracija, imale su veliki utjecaj na način na koji se razvijaju i održavaju softverski sistemi [6, 7]. Kroz integraciju razvoja (Development) i operacija (Operations), DevOps omogućava organizacijama da brže isporučuju softver, poboljšavaju kvalitet proizvoda i smanjuju vrijeme koje je potrebno za rješavanje problema. Kontejnerizacija i orkestracija su dvije ključne komponente ovog pristupa, omogućavajući efikasno pakovanje, distribuciju, skaliranje i upravljanje softverskim aplikacijama.

Statistika pokazuje da su DevOps tehnike i prakse sve prisutnije u industriji [6]. Postoji značajna količina literature vezana za ovdje razmatranu temu. U nastavku razmatrano je nekoliko knjiga i radova koji su temom vezani za DevOps, baze podataka kao i kontejnerizacijske i orkestracijske metode i tehnike.

1.2.1 "DevOps: A Software Architect's Perspective"

Rad "DevOps: A Software Architect's Perspective" autora Len Bassa, Ingo Webera i Liming Zhua [8] je jedan od prepoznatljivijih radova koji se bavi konceptom DevOps-a iz perspektive arhitekture softvera. Rad pruža duboko razumijevanje o DevOps metodologiji, objašnjavajući njenu filozofiju, prakse i alate koji se koriste za istu. Rad gleda na DevOps sa arhitektonskog stajališta, što pruža uvid u to kako se DevOps integrira s dizajnom i razvojem softverskih sistema [8]. Kroz različite studije slučaja, rad daje stvarne primjere kako se DevOps može primijeniti u raznim industrijskim kontekstima [8]. Međutim, budući da je fokusiran na arhitektonsku perspektivu, za rad se može reći da posjeduje nedostatak detalja u drugim oblastima DevOps-a, kao što su operativni i procesni aspekti. Također, može biti neprikladan za čitaoce koji su potpuno novi u DevOps-u, jer prepostavlja određeno razumijevanje softverske arhitekture.

Ovaj rad može predstavljati dobar temelj za upoznavanje sa DevOps metodama i tehnikama te kako DevOps pristupi, uključujući automatizaciju i kontinuiranu integraciju, mogu biti primjenjeni na kontejnerizaciju i orkestraciju baza podataka [8]. Pruža širok pregled DevOps-a dok će u ovom magistarskog radu biti specifičan fokus na upravljanje bazama podataka. Rad nudi brojne reference i smjernice koje mogu poslužiti kao osnova za daljnje istraživanje.

1.2.2 "Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices"

Rad pruža sveobuhvatan pregled tehnika i alata koji se koriste u kontinuiranoj integraciji, isporuci i postavljanju (CI/CD), kao i izazova i praksi u industriji [9]. U njemu imamo detaljno razmatranje različitih alata (kao npr. Jenkins) i praksi koji se koriste u CI/CD, uz dodatnu analizu kako se ti alati mogu primjenjivati u praksi te kako mogu da ubrzaju kompletan proces isporuke softvera krajnjem korisniku [9]. Prikazana su poređenja alata te šta je ostvarivo s kojim alatom.

Generalno, rad je baziran na metodama, tehnikama i praksama prilikom omogućavanja CI/CD,

Što je zapravo jedan od glavnih temelja DevOps metodologije [9]. Saznanja u radu se mogu primijeniti i pri radu sa bazama podataka, da omogućavaju njihovu što veću dostupnost kao i što jednostavnije ažuriranje i praćenje verzija.

1.2.3 "Performance of containerized database management systems"

U svom istraživanju, Rehmann i Folkerts [10] analiziraju performanse sistema za upravljanje bazama podataka koji su kontejnerizovani u okruženjima u oblaku. Rad se fokusira na uticaj konfiguracije kontejnera na performanse sistema za upravljanje bazama podataka, sa posebnim osvrtom na upravljanje resursima i rezultate performansi u kontejnerizovanim okruženjima. Autori navode da su, iako su sistemi za upravljanje bazama podataka relativno nova primjena računarstva u oblaku, prednosti kontejnera u okruženjima u oblaku podstakle nastojanja za njihovu kontejnerizaciju. Međutim, naglašavaju da su performanse DBMS-a u kontejnerima kompleksne za razumijevanje i kontrolu zbog efekata kao što su ograničenja resursa i međusobni uticaji kontejnera.

Rad detaljno opisuje mehanizme za upravljanje resursima kontejnera i politike za operacije sa kontejnerima, kao što su promjena veličine ili migracija kontejnera. Jedan od ključnih zaključaka iz rada jeste da virtualizacija na nivou operativnog sistema, iako pruža veću upravljivost i kontrolu resursa u poređenju s izvršavanjem na virtualizovanom hardveru, također donosi slabiju izolaciju među kontejnerima u odnosu na hardversku virtualizaciju. Ovo može rezultirati situacijom gdje jedan kontejner negativno utiče na performanse drugih kontejnera na istom hostu [10].

Ovaj rad pruža važan uvid u prednosti i izazove kontejnerizacije DBMS-a i može poslužiti kao koristan izvor za razumijevanje kako konfiguracija kontejnera može uticati na performanse baza podataka u cloud okruženjima. Rezultati Rehmann-a i Folkerts-a su značajni za razvoj efikasnijih i pouzdanih cloud rješenja za upravljanje bazama podataka.

1.2.4 "Enabling Containerized, Parametric and Distributed Database Deployment and Benchmarking as a Service"

U ovom radu autori George Kousiouris i Dimosthenis Kyriazis istražuju automatizirano i parametrizirano pokretanje i mjerjenje performansi distribuiranih baza podataka kao usluge u kontejneriziranim okruženjima. Autori se detaljno bave adaptacijom i sinhronizacijom potrebnom za osiguravanje sekvence testiranja, koristeći studiju slučaja na MySQL bazi podataka. Ovo uključuje prilagođavanje službenih Docker slika za MySQL i OLTP Bench klijente, te istraživanje scenarija promjene parametara i ko-alokacijskih scenarija gdje više instanci baza dijeli fizičke čvorove, što je uobičajeno u paradigm DBaaS (Database as a Service).

Posebno se ističe razvoj alata Flexibench, koji omogućava uvođenje opterećenja kao uslugu putem virtualiziranih klastera za uvođenje opterećenja, čime se značajno olakšava rad inženjera za performanse. Alat omogućava izravno testiranje odabrane konfiguracije i performansi baze podataka pod specifičnim ciljanim opterećenjem kroz jednostavne REST pozive.

Kako autorи navode, baze podataka predstavljaju dodatne izazove s obzirom na njihovu veliku potrebu za sinhronizacijom i orkestracijom izvršavanja benchmarka, posebno u tehnologijama orijentiranim na mikroservise (kao što su platforme za kontejnere) i dinamičkim poslovnim modelima kao što je DBaaS [11]. Također, rad opisuje prilagodbu alata otvorenog koda, osnovnog alata za uvođenje opterećenja kao usluge, Flexibench, kako bi omogućio automatizirano, parametrizirano pokretanje i mjerjenje kontejneriziranih i distribuiranih baza podataka kao usluge. Ovaj rad pruža značajan doprinos razumijevanju kako se baze podataka mogu učinkovito im-

plementirati i testirati u kontejneriziranom okruženju, što je ključno za razvoj učinkovitijih i fleksibilnijih rješenja za upravljanje bazama podataka u cloud okruženjima.

1.2.5 "Evaluation of Container Orchestration Systems for Deploying and Managing NoSQL Database Clusters"

U radu "Evaluation of Container Orchestration Systems for Deploying and Managing NoSQL Database Clusters", autori Eddy Truyen, Dimitri Van Landuyt, Bert Lagaisse, Wouter Joosen, i Matt Bruzek istražuju performanse i dodatno opterećenje koje sistemi za orkestraciju kontejnera poput Docker Swarma, Kubernetes i Mesos uvode prilikom upravljanja NoSQL bazama podataka. Fokus rada je na evaluaciji kako ovi sistemi podržavaju implementaciju i upravljanje klasterima NoSQL baza podataka, sa MongoDB sistemom za upravljanje bazama podataka kao studijom slučaja.

Autori objašnjavaju da iako su sistemi za orkestraciju kontejnera prvobitno dizajnirani za pokretanje balansiranih servisa bez stanja (eng. "stateless"), oni se također koriste za pokretanje klastera baza podataka zbog poboljšanih atributa otpornosti, kao što su brz automatski oporavak od kvara čvorova baze podataka i transparentnost lokacije na nivou TCP/IP veza između instanci baza podataka.

Ovaj rad pruža važne uvide u to kako orkestracijski alati mogu utjecati na performanse i upravljanje klasterima NoSQL baza podataka, što je ključno za organizacije koje razmatraju implementaciju ovih tehnologija u proizvodnim okruženjima.

1.2.6 "The Art of Monitoring"

"The Art of Monitoring" autora Jamesa Turnbulla [12] pruža sveobuhvatni vodič za nadgledanje, mjerjenje i vizualizaciju računarskih sistema i aplikacija. Rad se bavi metodologijama, alatima i praksama potrebnim za implementaciju uspješnog sistema nadgledanja, uzimajući u obzir performanse, raspoloživost i razmjere sistema.

Iako rad ne cilja specifično na baze podataka, tehnike i alati za nadgledanje koje opisuje mogu se primijeniti na njih [12]. To uključuje praćenje performansi, ispitivanje opterećenja, dijagnostiku problema i automatsko djelovanje na anomalije.

Nadziranje je ključna komponenta DevOps-a jer pruža uvid u rad aplikacije kroz cijeli životni ciklus, od razvoja do proizvodnje [12]. "The Art of Monitoring" pokriva aspekte koji su relevantni za DevOps, kao što su automatizacija, kolaboracija između timova, brza dostava i stalno poboljšanje.

Magistarski rad bavi se kontejnerizacijom i orkestracijom u kontekstu upravljanja bazama podataka pomoću DevOps alata, a nadziranje je veoma bitan dio te cjeline. Turnbullov rad [12] pruža dubinsko razumijevanje kako implementirati i optimizirati nadziranje u složenim sistemima.

Proučeni radovi u ovom poglavlju predstavljaju bogat skup istraživanja koji obuhvata različite aspekte DevOps pristupa, kontejnerizacije, orkestracije i nadziranja u kontekstu upravljanja bazama podataka. Analiza ovih radova omogućava razumijevanje trenutnih trendova, izazova i mogućnosti u ovim oblastima.

Radovi koji se bave DevOps pristupom pružaju uvid u važnost kolaboracije, automatizacije i kontinuirane integracije. Istovremeno, radovi na temu kontejnerizacije i orkestracije baza po-

dataka istražuju performanse, fleksibilnost i skalabilnost ovih sistema. Osim toga, literatura koja se bavi nadziranjem nudi sveobuhvatni pregled tehnika i praksi za praćenje i upravljanje resursima.

Ova sveobuhvatna analiza otvara prostor za daljnje razmatranje i istraživanje integracije DevOps-a s kontejnerizacijom i orkestracijom baza podataka. Temeljeći se na ovim spoznajama, ovaj rad nastoji postaviti i testirati hipoteze o mogućnostima poboljšanja efikasnosti, fleksibilnosti i kontrole performansi u upravljanju bazama podataka.

Ovi zaključci postavljaju okvir za daljnja istraživanja i mogu poslužiti kao polazna tačka za praktičnu primjenu i teorijsko oblikovanje predloženih rješenja.

1.3 Motivacija istraživanja

Motivacija za izbor ove oblasti istraživanja proizlazi iz složenosti i nužnosti optimizacije integracije DevOps procesa s upravljanjem baza podataka. Obzirom na stalno mijenjanje tehnološkog pejzaža i sveprisutnost baza podataka u modernim aplikacijama, postoji značajna potreba za unapređenjem praksi koje omogućavaju efikasno upravljanje i orkestraciju tih sistema. Proučavanje ovih aspekata ne samo da će pružiti dublje razumijevanje teorijskih koncepta već će otvoriti i novu perspektivu za praktična rješenja u industriji.

Lična motivacija autora za ovu temu odražava njegovu svakodnevnu interakciju s ovim izazovima kao DevOps inženjera. Susretanje s problemima integracije, automatizacije i skaliranja bazama podataka unutar različitih sistema pružilo mu je osjećaj odgovornosti da dublje istraži ovu oblast. Iako je svjestan složenosti i izazova koji ga čekaju, autor vjeruje da njegovo iskustvo i strast prema kontinuiranom učenju mogu pridonijeti ovom polju. Kroz ovo istraživanje, autor se nada da će ne samo povećati svoje razumijevanje o ovoj oblasti, već i pridonijeti širem razumijevanju i primjeni DevOps-a u kontekstu baza podataka.

1.4 Osnovni ciljevi i plan istraživanja

Osnovni ciljevi i plan istraživanja za ovaj magistarski rad temelje se na potrebi za dubljim razumijevanjem i integracijom DevOps praksi sa upravljanjem i održavanjem baza podataka. Sa sve većom primjenom ovih tehnologija u raznim industrijama, postoji jasna potreba za metodologijama koje omogućavaju efikasniju saradnju, automatizaciju i skaliranje.

Razlozi zbog kojih je potrebno istražiti ovaj problem su mnogobrojni. U praksi, mnoge organizacije suočavaju se sa izazovima u integraciji DevOps-a sa bazama podataka, što može dovesti do problema u performansama, sigurnosti i održivosti sistema. Nedostatak koherentnih strategija i praksi može ograničiti mogućnosti organizacija i smanjiti njihovu konkurenčku prednost.

Očekivanje je da će ovo istraživanje doprinijeti povećanju znanja u oblasti istraživanja na nekoliko načina. Prvo, istraživanje će pružiti bolji uvid u postojeće metode i prakse, identificirati njihove prednosti i mane, te razmotriti potencijalne puteve za unapređenje. Drugo, pružiti će praktične smjernice i alate koji mogu pomoći organizacijama u razvoju i implementaciji uspješnih strategija.

Plan istraživanja uključuje nekoliko ključnih koraka. To će uključivati detaljnu analizu literature i postojećih rješenja, razvoj teorijskog modela, te praktično testiranje kroz implementaciju prototipa ili korištenje postojećih alata. Također, istraživanje će se baviti i etičkim i sigurnosnim pitanjima, kako bi se osiguralo da predložene metode budu održive i odgovorne.

1.5 Metodologija istraživanja

Metodologija istraživanja za ovaj magistarski rad pomno je osmišljena kako bi se adekvatno pristupilo problemima i pitanjima koji su definirani. Za istraživanje integracije DevOps praksi sa bazama podataka, dostupne su različite metode, uključujući analizu literature, empirijska istraživanja, studije slučaja, eksperimente i simulacije. Ove metode mogu se primjeniti za analizu postojećih praksi, identifikaciju izazova i pružanje rješenja.

Za ovaj rad izabrane su sljedeće metode:

- Analiza literature: Proučavanje i sinteza postojeće literature da bi se utvrdilo trenutačno stanje u oblasti.
- Studije slučaja: Detaljna analiza specifičnih primjera kako bi se dobili uvidi iz stvarnog svijeta.
- Eksperimenti i simulacije: Primjena praktičnih testiranja i modeliranja da bi se procijenila učinkovitost i primjenljivost predloženih rješenja.

Istraživanje će se provesti u nekoliko faza. Prvo će se obaviti temeljita analiza literature kako bi se identificirale ključne teme, izazovi i postojeća rješenja. Nakon toga, selektirat će se relevantni slučajevi za detaljniju studiju, a u skladu s njima oblikovat će se eksperimenti ili simulacije.

Eksperimentalni dio će uključivati razvoj i testiranje prototipa ili korištenje postojećih alata u kontroliranom okruženju. Rezultati će se analizirati i interpretirati kako bi se ocijenila njihova značajnost i primjenljivost.

Ovaj pristup omogućava kombinaciju teorijskog i praktičnog razumijevanja, pružajući holistički uvid u problematiku. Cilj je identificirati konkretne strategije i rješenja koja mogu biti praktično primjenjiva i koja će pružiti značajan doprinos razumijevanju kako DevOps može biti efikasno integriran s upravljanjem bazama podataka.

1.6 Očekivani rezultati

U ovom magistarskom radu, cilj je pružiti sveobuhvatno razumijevanje i praktičnu primjenu DevOps pristupa u upravljanju bazama podataka, koristeći specifične alate i tehnologije koje su postale standard u industriji.

Konkretnе tehnologije koje će se koristiti uključuju Docker za kontejnerizaciju, Kubernetes za orkestraciju, Grafana za nadziranje i ArgoCD za kontinuiranu isporuku. Ovaj pristup omogućit će razvoj modela sistema upravljanja bazama podataka koji je kontejnerizovan i orkestiran, što predstavlja centralni doprinos ovog rada.

Očekuje se da će kroz analizu i implementaciju ovih alata rad pružiti dublje uvide u mogućnosti i izazove koji se susreću u realnim aplikacijama. To će uključiti razmatranje najboljih praksi, identifikaciju potencijalnih poteškoća i razvoj rješenja koja se mogu primjeniti u širem kontekstu.

Osim toga, očekuje se da će rad služiti kao osnova za daljnje istraživanje i razvoj u ovom području, povezujući teorijske osnove s praktičnim iskustvom i pružajući konkretne smjernice i primjere koji mogu koristiti stručnjacima u polju.

U konačnici, rad ima za cilj pokazati kako integracija modernih alata i tehnologija može dovesti do veće efikasnosti, sigurnosti i skalabilnosti sistema. Ova integracija omogućit će bolje razumijevanje dinamičkog i sve važnijeg područja upravljanja bazama podataka u kontekstu DevOps pristupa.

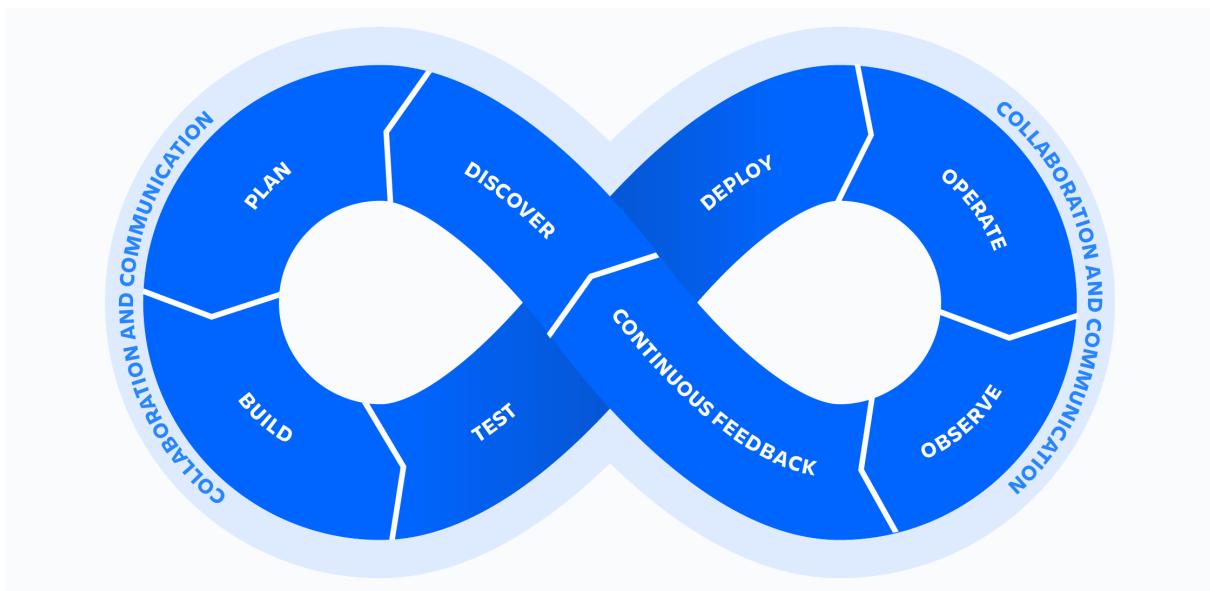
* * *

U ovom poglavlju istražene su ideje primjene DevOps pristupa kao i samih popratnih alata pri upravljanju i radom sa bazama podataka. Razmotreno je stanje u oblasti, analizirajući par konkurenčnih i aktualnih radova zasnovanih na oblasti koju istražuje ovaj magistarski rad. Kao motivacija istraživanja navedena je aktualnost teme u praksi i primjeni sličnih pristupa u praksi čak i u većim korporacijama. Naveden je redoslijed analize koja će se vršiti u radu te je spomenuto koje metode istraživanja će biti korištene. U narednom poglavlju će biti opisan sam DevOps kao metodologija razvoja softvera, te najpopularniji alati (Docker, Kubernetes, ..) koji će biti korišteni i u studiji slučaja provedenoj u svrhu pisanja ovog rada.

Poglavlje 2

DevOps

DevOps, spoj riječi *DEVelopment* (razvoj) i *OPerationS* (operacije), predstavlja praksu koja zahtijeva suradnju između softverskih programera i sistemskih administratora te automatizaciju svih faza razvojnog životnog ciklusa [8]. DevOps je nastao kao odgovor na potrebu za bržom isporukom softvera i boljom integracijom između razvoja i operacija [13]. Na slici 2.1 prikazan je životni ciklus razvoja softvera koristeći DevOps pristup razvoju.



Slika 2.1: Životni ciklus razvoja softvera koristeći DevOps pristup [1]

DevOps se zasniva na principima agilnog razvoja i kontinuirane isporuke. Njegov fokus je na postizanju bržeg vremena isporuke, veće pouzdanosti i bolje kvalitete kroz blisku suradnju i brzu komunikaciju između timova. Tehnike i prakse koje se koriste pri ovom pristupu su:

1. Kontinuirana Integracija (CI): Automatizira proces integracije koda od strane različitih članova tima.
2. Kontinuirana Dostava (CD): Automatizira isporuku koda u proizvodnu okolinu.
3. Kontinuirani nadzor: Praćenje sistema u realnom vremenu da bi se identificirali problemi i rješili na vrijeme.

Alati igraju ključnu ulogu u automatizaciji i olakšavanju DevOps procesa. Neki popularni alati uključuju Jenkins za automatizaciju, Docker za kontejnerizaciju, Kubernetes za orkestraciju,

Grafana za nadziranje, Ansible za automatizaciju konfigurisanja, Terraform za automatizaciju upravljanja infrastrukturom (eng. "Infrastructure as Code" - IaC) itd.

Implementacija DevOps-a donosi mnoge prednosti, uključujući bržu isporuku, veću stabilnost operacija i poboljšanu komunikaciju između timova. Ipak, također donosi i izazove, kao što su potreba za kulturološkom promjenom, složenost novih alata i tehnologija te potreba za kontinuiranim učenjem i usvajanjem novih vještina. DevOps je transformativni pristup koji može donijeti značajne dobiti za organizacije koje ga usvajaju. Integracija i automatizacija procesa ne samo da ubrzava razvojni ciklus, već također potiče veću kvalitetu i stabilnost. DevOps kultura je složen i dinamičan pristup koji promoviše saradnju između razvojnih (Dev) i operativnih (Ops) timova, te integraciju i automatizaciju procesa. Cilj je postići brži i učinkovitiji razvoj i isporuku softvera.

Kolaboracija i integracija su srž DevOps-a, omogućavajući timovima da rade zajedno prema zajedničkim ciljevima. Automatizacija procesa, kao što su integracija, testiranje, isporuka i nadzor, ključna je za brže i pouzdanije isporuke, smanjujući mogućnost ljudske greške.

Kontinuirana integracija i isporuka (CI/CD) su bitne komponente koje olakšavaju redovite isporuke koda spremnog za produkciju. DevOps također naglašava odgovornost i vlasništvo, gdje su timovi odgovorni za cijeli životni ciklus aplikacije, od razvoja do produkcije i podrške.

Fokus na klijenta je još jedan važan aspekt DevOps kulture, gdje su sve odluke vođene potrebama i očekivanjima klijenta, a brza povratna informacija od korisnika je ključna za kontinuirano usmjeravanje i unapređenje.

DevOps podržava agilne metodologije, omogućavajući timovima da budu fleksibilni i da se prilagode promjenama. Stalno praćenje i brza povratna informacija omogućavaju timovima da brzo reaguju na probleme.

Kultura eksperimentiranja i učenja, gdje se potiče eksperimentiranje i prihvatanje neuspjeha kao prilika za učenje, ključna je za inovacije i rast. Integracija sigurnosnih praksi u cijeli životni ciklus razvoja, poznata kao "DevSecOps", osigurava da se sigurnosni zahtjevi adekvatno rješavaju. Na kraju, DevOps pristup omogućava razvoj skalabilnih i otpornih sistema koji se mogu lahko prilagoditi različitim operativnim uvjetima i zahtjevima korisnika. Sve u svemu, DevOps kultura je više od skupa alata ili tehnika; to je način razmišljanja koji potiče timove da razvijaju, isporučuju i podržavaju softver na način koji je brži, pouzdaniji i usklađeniji s poslovnim i korisničkim potrebama.

2.1 Razlike između tradicionalnih pristupa razvoju i DevOps-a

Tradicionalni pristup razvoju softvera i DevOps predstavljaju dvije različite paradigme koje se razlikuju u nizu ključnih aspekata:

1. Organizacija i saradnja:

- Tradicionalni pristupi: Timovi su često odvojeni i segmentirani, gdje su razvojni timovi odgovorni za pisanje koda, a operativni timovi za održavanje i podršku. Ova separacija može dovesti do nedostatka komunikacije i sporijeg procesa odlučivanja.
- DevOps: Naglašava blisku saradnju između razvoja i operacija, integrirajući timove u zajednički radni tok. Ovo potiče brzu komunikaciju i bolju koordinaciju.

2. Automatizacija i integracija:

- Tradicionalni pristupi: Procesi kao što su integracija, testiranje, isporuka i nadzor često se obavljaju ručno, što može biti vremenski zahtjevno i podložno greškama.
- DevOps: Ovi procesi su u velikoj mjeri automatizirani, čime se povećava brzina, pouzdanost i efikasnost isporuke.

3. Metodologija razvoja:

- Tradicionalni pristupi: Ovaj pristup često koristi "vodopadni" model, gdje su faze razvoja sekvensijalne i promjene su teške za implementirati nakon što je projekat započeo.
- DevOps: Omogućava kontinuiranu integraciju i isporuku (CI/CD), podržavajući agilne metodologije koje olakšavaju brze iteracije i promjene.

4. Odziv na korisničku povratnu informaciju:

- Tradicionalni pristupi: Dugi ciklusi razvoja mogu otežati prilagodbu proizvoda na temelju korisničke povratne informacije.
- DevOps: Brza iteracija i kontinuirana interakcija s korisnicima omogućavaju pravo-vremeno usklađivanje s tržišnim potrebama i očekivanjima korisnika.

5. Sigurnost i usklađenost:

- Tradicionalni pristupi: Sigurnost je često dodana kao naknadna faza, što može dovesti do problema u kasnijim fazama razvoja.
- DevOps: Integrira sigurnosne prakse kroz cijeli životni ciklus razvoja (DevSecOps), čime se poboljšava sigurnost i usklađenost.

Ovi različiti aspekti ilustriraju kako DevOps pristup može donijeti značajne prednosti u odnosu na tradicionalne metode, u smislu brzine, fleksibilnosti, kvalitete i odgovornosti. Ipak, primjena DevOps-a također zahtjeva promjene u organizacijskoj kulturi, procesima i alatima, što može predstavljati izazove u nekim kontekstima.

U ostatku poglavlja će se razmatrati glavni procesi i alati koji se koriste pri primjeni DevOps principa u razvojnog ciklusu.

2.2 Kontejnerizacija

2.2.1 Uvod u kontejnerizaciju

Kontejnerizacija je proces koji uključuje ogradijanje i pakovanje aplikacije i biblioteka i dodatnog softvera od kojeg zavisi unutar "kontejnera". Ovaj pristup omogućuje da aplikacija bude konzistentna i pouzdana kada se pokreće u različitim računarskim okolinama. Kontejnerizacija se odnosi na tehnologiju koja omogućava izolaciju aplikacije i njenih ovisnosti unutar kontejnera. Kontejner ima vlastiti korisnički prostor, biblioteke i postavke, pri čemu je sve pakovano u jednu cjelinu. Za razliku od tradicionalne virtualizacije, koja virtualizira cijeli fizički računar, kontejnerizacija se izvodi na nivou operativnog sistema. To znači da kontejneri dijele isto jezgro operativnog sistema, ali rade izolirano jedni od drugih. Takav pristup čini kontejnere laganim i brzim za pokretanje. Svrha korištenja kontejnerizacije može biti povećanje konzistentnosti, portabilnosti, efikasnosti i sigurnosti.

Kontejnerizacija je tehnologija koja je revolucionarizirala način na koji se razvijaju, raspoređuju i upravljaju aplikacije. Početak kontejnerizacije može se pratiti još iz 1970-ih, kada je stvoren "Chroot". Ovaj sistem je omogućavao izolaciju datotečnih sistema. Naredni koraci u polju kontejnerizacije napravljeni su 2000. godine, kada su dodani tzv. "zatvori" (eng. "jails") u sklop FreeBSD operativnog sistema. Operativni sistem bi se sastojao od više tih nezavisnih manjih sistema zvanim zatvori [14].

Sigurnost kontejnera postala je važna tema s razvojem Linux Containers (LXC) projekta 2008. godine [14]. Dodane su značajke kao što su SELinux i Seccomp da se ojača sigurnost kontejnera. Ovo je dovelo do pitanja kao što je "Da li su LXC kontejneri dovoljni?", te je sigurnost kontinuirano unaprijeđena [14].

Međutim, LXC kontejneri su bili nezgrapni za korištenje i zahtijevali su veliku količinu rada i poznavanja od strane čovjeka. Najveća prekretnica u kontejnerizaciji javila se 2013. godine pojavom alata Docker.

Zbog brzo stečene popularnosti i velike primjene Dockera, bio je potreban način kako da se organizuje upravljanje kontejnerima. 2015. godine razvijen je orkestracijski sistem Kubernetes koji rješava navedene probleme. Docker i Kubernetes danas predstavljaju skoro pa neophodne alate pri implementiranju DevOps pristupa u proces razvoja softvera.

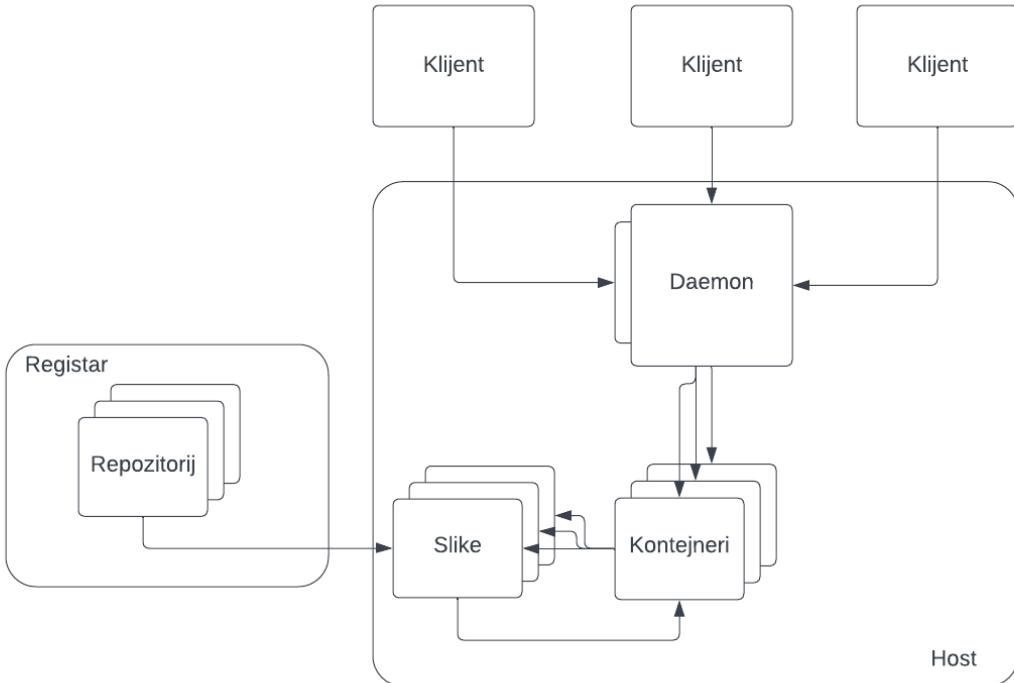
2.2.2 Docker

Docker je platforma otvorenog koda na kojoj se pokreću aplikacije i olakšava proces razvoja i distribucije [15]. Potreba za pojavitom kontejnerizacije i samog Dockera pojavila se zbog problema praćenja verzija biblioteka i drugih zavisnosti potrebnih za rad aplikacije, tzv. pakao zavisnosti (eng. "dependency hell"). Aplikacije koje su izgrađene u Dockeru su upakovane sa svim potrebnim zavisnostima u standardni oblik koji se naziva kontejner. Kontejneri rade na izoliran način na sloju iznad jezgre operativnog sistema [15]. Velika prednost ovih kontejnera jeste što pomoću njih aplikacije mogu biti pokrenute na različitim okolinama, nezavisno od operativnog sistema na kojem se nalaze.

Arhitektura Dockera

Postoje 4 osnovne komponente [15]:

1. klijent i server
2. slike
3. registri
4. kontejneri



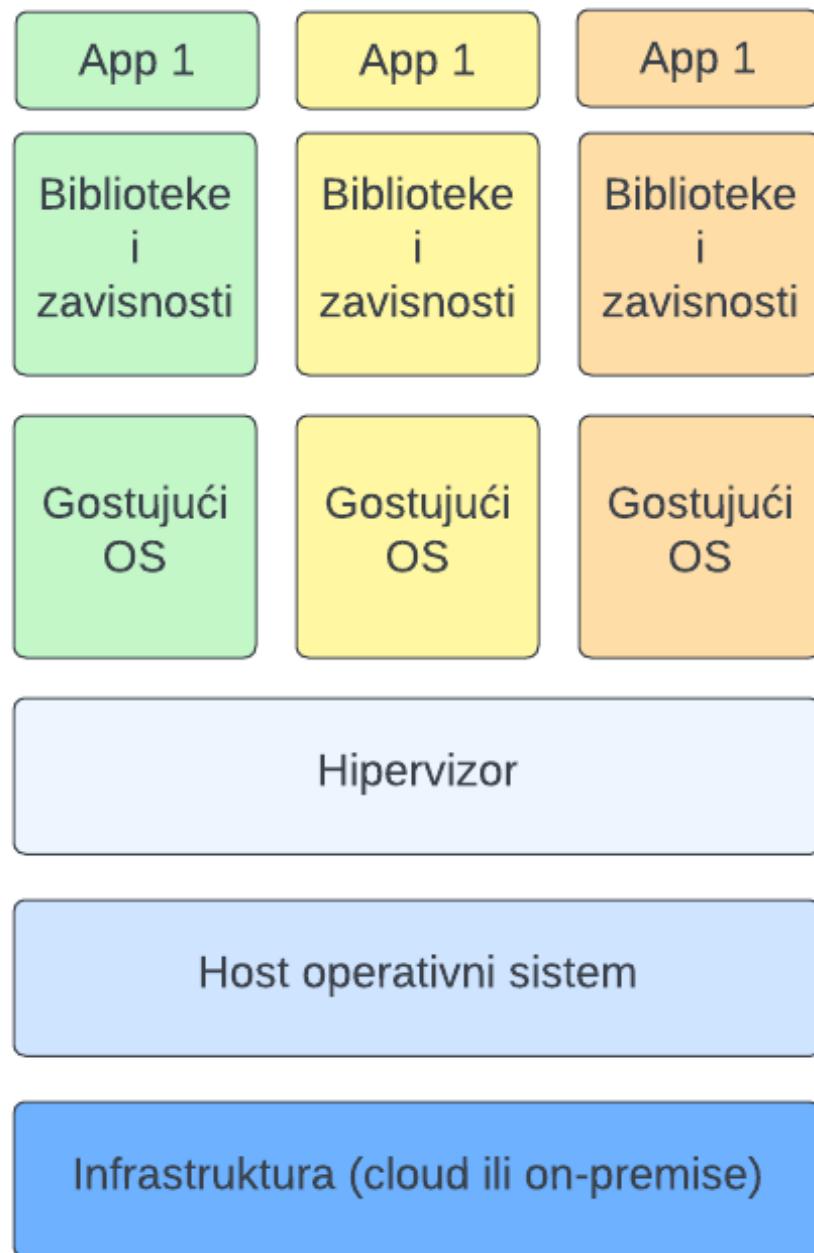
Slika 2.2: Arhitektura Docker sistema

Na slici 2.2 prikazana je cijelokupna struktura Dockera sa svim svojim komponentama. Docker se može opisati kao aplikacija bazirana na klijent-server arhitekturi. Klijent preko API poziva ili komandne linije pristupa serveru, koji ustvari predstavlja pozadinski servis koji upravlja ostalim komponentama. Na osnovu zahtjeva klijenta, server vrši radnje nad kontejnerima koje je sam klijent specificirao preko poziva. Komunikacija servera sa kontejnerima se vrše putem kontrolnih grupa (eng. "cgroups") i imenskih prostora (eng. "namespaces"). Kontrolne grupe su Linux funkcionalnost koja omogućava ograničavanje i nadzor upotrebe resursa kao što su CPU, memorija, disk I/O, mreža i drugo za skupinu procesa. Ovo je ključno za kontejnerizaciju jer omogućava host sistemu da precizno kontroliše koliko resursa svaki kontejner može koristiti, i da osigura da jedan kontejner ne može monopolizirati resurse i time utjecati na druge kontejnere. Imenski prostori predstavljaju drugu ključnu Linux funkcionalnost koja se koristi u kontejnerizaciji. Oni pružaju izolaciju tako da svaki kontejner može imati vlastite globalne resurse (kao što su ID korisnika, imena procesa, IP adrese) bez međusobnog ometanja. Na primjer, uz pomoć imenskih prostora, dva različita kontejnera mogu imati proces sa PID 1 ili imati vlastitu mrežnu konfiguraciju bez međusobnog ometanja.

Dalje imamo slike. Slika je neizmjenjivi, čitljivi šablon koji sadrži niz instrukcija i potrebnih datoteka za kreiranje radne instance kontejnera. To uključuje kod aplikacije, biblioteke, varijable okoline, konfiguracijske datoteke i metapodatke koji opisuju kako kontejner treba raditi. Docker slike se mogu pronaći i dijeliti putem registara slika, kao što je Docker Hub, što olakšava pronalaženje i distribuciju aplikacija i servisa koje su spakovane kao Docker kontejneri. Radna instanca slike predstavlja kontejner.

Razlika između virtualnih mašina i kontejnera

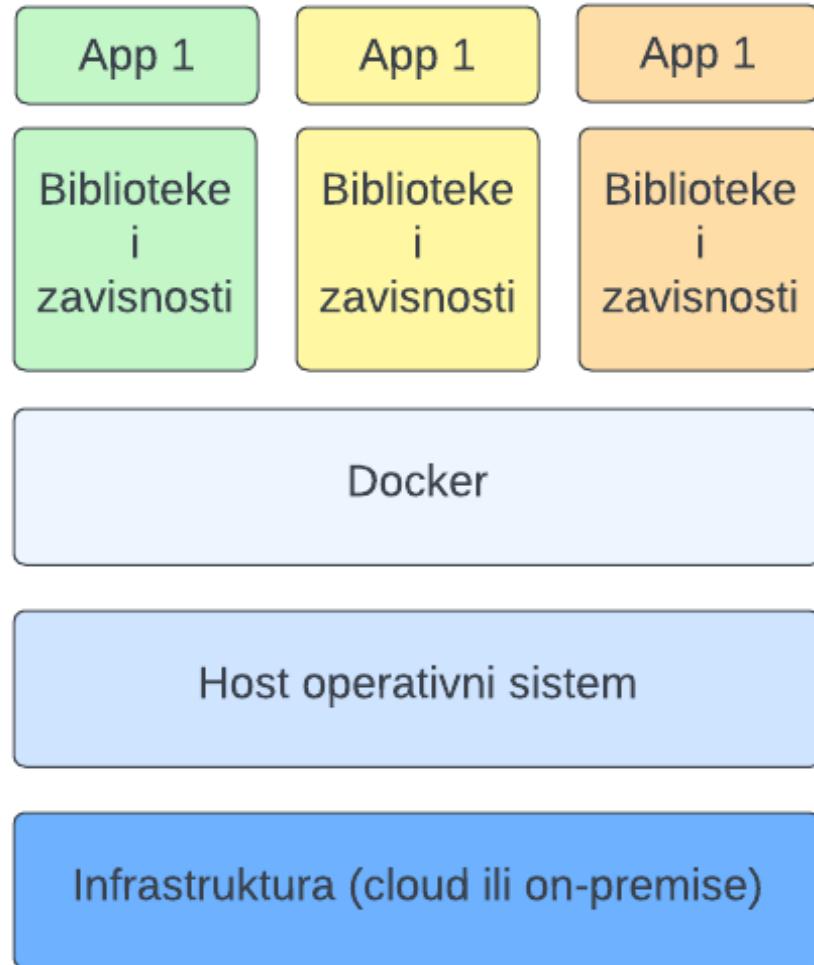
Virtualizacija je tehnologija koja omogućava stvaranje više simuliranih okruženja ili posvećenih resursa iz jednog fizičkog hardverskog sistema. Virtualizacija se često postiže korištenjem hipervizora, posebnog softvera koji upravlja virtualnim mašinama i pruža apstrakciju između operativnog sistema i fizičkog hardvera. Ovo omogućava virtualnim mašinama da dijele resurse fizičkog računara, poput procesora, memorije i diskovnog prostora, dok su ujedno izolirani jedan od drugoga. Na slici 2.3 prikazana je arhitektura na kojoj je zasnovan rad virtualnih mašina.



Slika 2.3: Arhitektura virtualnih mašina

S druge strane, kontejneri dijele isto OS jezgro, dok svaka virtualna mašina ima zasebno. To

čini kontejnere manjim, lakšim za prenošenje i pokretanje te zahtijevaju i manje resursa. Ipak, sama posljedica toga jeste da su kontejneri manje izolirani u odnosu na virtualne mašine. Na slici 2.4 je prikazana arhitektura Docker kontejnera.



Slika 2.4: Arhitektura Docker kontejnera

Prema Waldsprugeru, zbog zahtjeva za resursima, u praksi je moguće imati do 13 pokrenutih virtualnih mašina na istom uređaju [15]. Ovaj problem se ne javlja kod kontejnera.

Prednosti i mane kontejnerizacije

Kontejnerizacija ima brojne prednosti koje opravdavaju ogromnu zainteresovanost i primjenu u posljednjim godinama. Neke od najvećih prednosti su nabrojane u nastavku [15]:

1. Brzina - kreiranje kontejnera te njihovo pokretanje traži veoma malo vremena u odnosu na neke alternativne pristupe pakovanja softvera.
2. Prenosivost - jednostavno prenosivi kao cjeline.

3. Skalabilnost - moguć je veći broj instanci iste slike na veoma jednostavan način, što je veoma korisno u brzo rastućim sistemima.
4. Olakšana isporuka - sprječavaju pojavu problema vezanih za verzije paketa i biblioteka potrebnih za rad aplikacije.
5. Dobro iskorištavanje resursa

Osim ovih nabrojanih prednosti, konkretno Docker posjeduje određene nedostatke. Na primjer, Docker nije podržan za rad na starijim računarima zasnovanih na 32-bitnoj arhitekturi [15]. Također, iako je moguće pokretati i kontejnere bazirane na Linuxu i na Windows sistemima, to nije baš u potpunosti tačno. Docker na Windows i Mac operativnim sistemima zapravo prvo instalira Linux virtualnu mašinu na host OS, na koji se dalje instalira sam Docker.

U nastavku poglavlja će biti prikazan način na koji se može upravljati kontejnerima u sistemima velikih dimenzija i zahtjeva za skalabilnošću.

2.3 Orkestracija

2.3.1 Uvod u orkestraciju

Sve počinje i završava sa hardverom [16]. U svrhu pokretanja bilo kakvog računarskog programa, potreban je hardver na kojem će se pokrenuti. Nad tim hardverom pomoću velikog broja apstrakcija, krajnji korisnik često i ne mora voditi brigu o samom hardveru. Toj činjenici dodatno doprinosi i pojava Dockera kao alata. Međutim, povećanjem zahtjeva dolazi i do potreba za skaliranjem. Samim tim, broj kontejnera kojima je potrebno upravljati se višestruko povećava. Zbog ovoga bio je potreban način automatizovanja upravljanja kontejnerima.

Orkestracija se odnosi na automatizovanu konfiguraciju, koordinaciju i upravljanje računarskim sistemima, aplikacijama i uslugama. Pojam "orkestracija" često se koristi u kontekstu automatizacije zadataka i radnih tokova u kojima se različite komponente i usluge koordiniraju kako bi se postigao određeni cilj ili rezultat. Termin je preuzet iz muzike, gdje dirigent orkestira različite instrumente kako bi stvorio harmoničnu kompoziciju. Neke od prednosti koje nudi sama orkestracija u savremenom računarstvu su:

1. Automatizacija radnih tokova: Orkestracija omogućava automatizaciju kompleksnih radnih tokova u kojima se više zadataka izvodi u određenom redoslijedu ili pod određenim uslovima.
2. Koordinacija resursa: U IT kontekstu, orkestracija može obuhvatiti koordinaciju virtualnih mašina, kontejnera, mrežnih resursa i skladištenja kako bi se postigla optimizacija performansi i iskorištenosti resursa.
3. Konfiguracija i upravljanje: Orkestracija omogućava centralizovano upravljanje i konfiguraciju različitih resursa i usluga, često kroz jedinstven interfejs ili alat.
4. Povećanje efikasnosti: Kroz orkestraciju, organizacije mogu postići bržu isporuku usluga, bolje iskorištenje resursa i smanjenje čovjekovih grešaka.
5. Primjena u modernim tehnologijama: Jedna od najpoznatijih primjena orkestracije je u kontekstu kontejnerizacije, gdje orkestracioni alati poput Kubernetes-a upravljaju razmještanjem, skaliranjem i operacijama kontejnera u klasterima.

U nastavku će se pri spominjanju orkestracije podrazumijevati orkestracija u kontekstu kontejnerizacije.

Dakle, orkestracija se bavi automatizovanim upravljanjem životnog ciklusa kontejnera. Dok kontejnerizacija olakšava proces razvoja, razmještanja i pokretanja aplikacija, kada se koristi na većoj skali, postavlja se potreba za koordinacijom, upravljanjem i automatizacijom svih tih kontejnera. Tu na scenu stupa orkestracija kontejnera. Ovim je omogućeno razmještanje kontejnera, skaliranje, balansiranje opterećenja, automatski popravak, mrežne konfiguracije, upravljanje pohranom podataka, bezbjednost te još mnogi drugi aspekti.

2.3.2 Alati za kontejnersku orkestraciju

Postoje mnogobrojni alati napravljeni u svrhu orkestriranja u kontekstu kontejnerizacije. Neki od najpopularnijih su:

1. Kubernetes - najpopularniji alat za kontejnersku orkestraciju. Razvijen je od strane Google-a i sada je pod upravom Cloud Native Computing Foundation (CNCF). Kubernetes pruža visoku fleksibilnost, skalabilnost i podršku za različite cloud provajdere i lokalne infrastrukture [16].
2. Docker Swarm - Integriran je direktno u Docker, što ga čini lakšim za postavljanje i korištenje, posebno za one koji su već upoznati s Docker-om. Iako nije tako funkcionalno bogat kao Kubernetes, Docker Swarm je često izbor za manje i jednostavnije primjene.
3. Openshift - Proizvod od Red Hat-a koji je zasnovan na Kubernetesu. Pruža dodatne funkcije, kao što su razvojna platforma, integracija sa CI/CD alatima i poboljšano umrežavanje.
4. Apache Mesos i Apache Marathon - Apache Mesos je platforma za upravljanje klasterima, dok je Marathon alat za orkestraciju kontejnera koji radi na Mesos platformi.

Od prethodno nabrojanih alata, Kubernetes dobiva najveću pažnju te je i najpopularniji. Razlog tome leži u podrškama koje stiče od strane velikih cloud provajdera poput Azure, AWS, Google Cloud itd. Na slici 2.5 se mogu vidjeti prethodno opisani alati.

Naredna sekcija se bavi konkretno radom Kubernetes alata.



Slika 2.5: Alati za kontejnersku orkestraciju [2]

2.3.3 Kubernetes

Kubernetes, često nazivan K8s, je platforma otvorenog koda dizajnirana za automatizaciju, skaliiranje i upravljanje kontejnerizovanim aplikacijama. Razvijen je prvenstveno za rad sa kontejnerima, posebno Docker, ali podržava i druge kontejnerske tehnologije. Kubernetes je pokrenuo Google 2014. godine, ali sada ga održava Cloud Native Computing Foundation (CNCF). Ovaj alat je postao standard u industriji za orkestraciju kontejnera i koristi se širom svijeta u mnogim proizvodjачkim okruženjima. Alat je razvijen koristeći programski jezik Go, a sami rad s alatom baziran je na korištenju YAML serijalizacijskog jezika [17].

Jedan od ključnih izazova u upravljanju kontejneriziranim aplikacijama je upravljanje njihovim životnim ciklusom i održavanje dostupnosti. Kubernetes adresira ove izazove kroz koncepte kao što su automatsko liječenje komponenti, automatsko skaliranje, i automatizirano raspoređivanje (eng. "rollouts") i vraćanje na prethodno stanje (eng. "rollbacks"), što omogućava aplikacijama da ostanu operativne čak i u slučaju grešaka ili promjena u infrastrukturi.

Arhitektura Kuberntesa

Kubernetes koristi arhitekturu klastera za upravljanje i orkestraciju kontejnera. U osnovi, klaster je skup računarskih čvorova koji se sastoji od dva tipa istih: master čvor/kontrolna ravan (eng. "Master node/Control plane") i radni čvorovi (eng. "Worker nodes").

Master čvor predstavlja mozak Kubernetes klastera i odgovoran je za upravljanje istim. Sadrži nekoliko ključnih komponenti:

1. API server - to je server koji predstavlja poveznici između klijenta (onoga ko izriče komande za manipulaciju nad klasterom) i samog klastera.
2. etcd - predstavlja ključnu komponentu Kubernetes ekosistema. To je distribuirani sistem za pohranu podataka u formatu par ključ-vrijednost. Često, za veće klastere, etcd

je distribuiran na veći broj čvorova radi redundantnosti. Sadrži informacije o konfiguraciji klastera, detalje o aplikacijama i servisima, podatke o korisnicima i njihova prava pristupa.

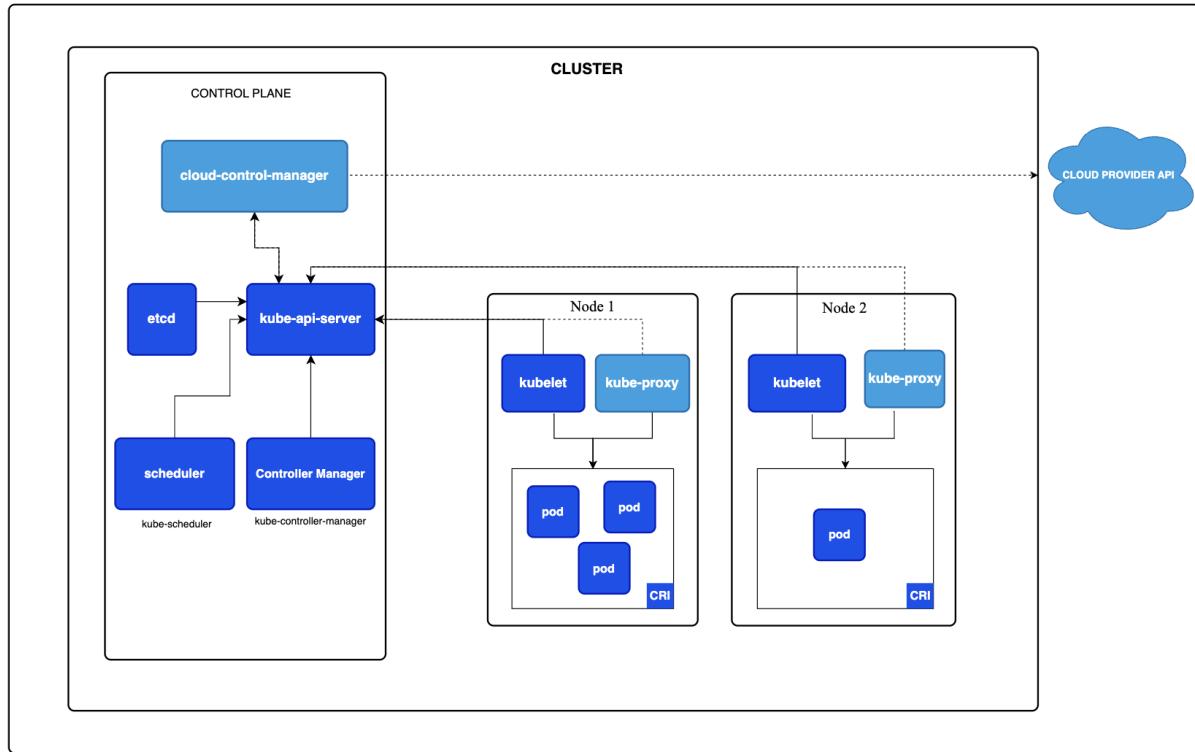
3. Raspoređivač (eng. "Scheduler") - donosi odluku o raspoređivanju kontejnera po čvorovima kao i drugih Kubernetes komponenti radi balansiranja opterećenja na pojedine mašine.
4. Menadžer kontrolera (eng. "Controller Manager") - Kontroleri u kontekstu Kubernetes-a su procesi koji kontinuirano prate stanje klastera i pokušavaju da ga usklade sa željenim stanjem koje je definisano kroz razne Kubernetes resurse. Svaki kontroler se fokusira na specifičan aspekt klastera, upravljujući određenim resursima i pravilima kako bi osigurao da aplikacije i servisi rade ispravno i efikasno. Kontroleri rade na principu rekurentne provjere stanja. Oni periodično pregledavaju trenutno stanje klastera (koje mogu dobiti iz API servera), upoređuju ga sa željenim stanjem, i preduzimaju akcije za usklađivanje ako je potrebno. Ovaj proces se često naziva i "kontrolna petlja" ili "rekurentna petlja". Menadžer kontrolera upravlja različitim kontrolerima koji automatiziraju procese poput skaliranja, popravki i orkestracije komponenti kao što je na primjer tzv. pod (o kojem će biti više govora u nastavku).

Radni čvorovi su ključni sastavni dijelovi Kubernetes klastera koji izvršavaju zadatke u klasteru. Svaki radni čvor je fizička ili virtualna mašina koja sadrži potrebne komponente za pokretanje samih aplikacija. Komponente ovih čvorova su:

1. Kubelet - agent koji se pokreće na svakom radnom čvoru i odgovoran je za komunikaciju s kontrolnim čvorom. Kubelet upravlja životnim ciklusom podova [18] (najmanja komponenta u kojoj je pokrenuta aplikacija u klasteru) na radnom čvoru, uključujući pokretanje, zaustavljanje i održavanje stanja podova u skladu sa željenom konfiguracijom.
2. Kube-proxy - mrežni posrednik koji se pokreće na svakom radnom čvoru. Njegova uloga je da održava mrežnu komunikaciju unutar klastera, osiguravajući da svaki pod može komunicirati s drugim podovima, bez obzira na to na kojem se radnom čvoru nalaze.
3. Container Runtime - softver koji pokreće kontejnere. Zbog nedostatka adekvatnog preveda u nastavku će se koristiti izvorni naziv na engleskom jeziku. Najčešće korišteni container runtime-ovi su Docker, containerd i CRI-O. Container runtime pokreće, zaustavlja i upravlja životnim ciklusom kontejnera na radnom čvoru.

Radni čvorovi primaju zadatke od kontrolnog čvora i izvršavaju ih, što omogućava distribuirano i skalabilno pokretanje aplikacija unutar Kubernetes klastera. Efikasno upravljanje radnim čvorovima ključ je za postizanje visokih performansi i pouzdanosti u proizvodnim okruženjima.

U nastavku na slici 2.6 su prikazane sve prethodno nabrojane komponente i način na koji one međusobno komuniciraju i izvršavaju svoje zadatke.



Slika 2.6: Kubernetes arhitektura [3]

Ključne funkcionalnosti Kuberntesa

Kubernetes je moćan alat za upravljanje kontejnerskim aplikacijama, nudi širok spekter funkcionalnosti koje omogućavaju skalabilnost, otpornost i efikasno upravljanje aplikacijama i u kojima se upravo i ogleda korist samog alata i njegov doprinos u današnjem načinu razvoja softvera. Ključne funkcionalnosti uključuju:

1. Automatsko raspoređivanje (eng. "Automatic Scheduling") - Kubernetes koristi sofisticirane algoritme za automatsko raspoređivanje kontejnera na čvorove u klasteru, uzimajući u obzir resurse, ograničenja i druge faktore kako bi optimizirao upotrebu resursa.
2. Samoozdravljenje (eng. "Self-Healing") - Kubernetes automatski restartuje neuspjele podove, zamjenjuje i ubija kontejnere koji ne odgovaraju na korisnički definisane zdravstvene provjere, i vrši ponovno raspoređivanje podova kada čvorovi klastera postanu nedostupni.
3. Horizontalno skaliranje (eng. "Horizontal Scaling") - Kubernetes omogućava horizontalno skaliranje aplikacija dodavanjem ili uklanjanjem podova na osnovu korisnički definisanih metrika, kao što su CPU i memorijска opterećenja. Ovo se može raditi automatski ili ručno putem komandi.
4. Otkrivanje usluga i balansiranje opterećenja (eng. "Service Discovery and Load Balancing") - Kubernetes automatski otkriva usluge i koristi DNS ili IP adrese za balansiranje opterećenja između različitih podova iste aplikacije, osiguravajući visoku dostupnost i optimalno korištenje resursa.

5. Upravljanje konfiguracijom (eng. "Configuration Management") - Kubernetes upravlja konfiguracijom aplikacija pomoću konfiguracijskih mapa i tajni, omogućavajući bezbjedno skladištenje i upravljanje osjetljivim informacijama kao što su lozinke i ključevi.
6. Raspoređivanje i ažuriranje aplikacija (eng. "Deployment and Rollout Management") - Kubernetes upravlja raspoređivanjem aplikacija i njihovim ažuriranjem, omogućavajući postupno uvođenje novih verzija aplikacija bez prekida rada, kao i povratak na prethodne verzije u slučaju grešaka.
7. Persistencija skladišta (eng. "Storage Orchestration") - Kubernetes omogućava automatsko montiranje skladišta po izboru korisnika, bilo da je riječ o lokalnom disku, NAS ili skladištu baziranom na oblaku (eng. "Cloud-based").
8. Upravljanje resursima (eng. "Resource Management") - Kubernetes pruža detaljnu kontrolu nad resursima kao što su CPU i memorija za svaki pod i čvor, omogućavajući korisnicima da definišu ograničenja i zahtjeve za resurse.
9. Nadziranje i evidentiranje (eng. "Monitoring and Logging") - Kubernetes integrira alate za nadzor i evidenciju, kao što su Prometheus i EFK (Elasticsearch, Fluentd, Kibana) stek, omogućavajući detaljno praćenje performansi aplikacija i resursa, kao i analizu logova za brže rješavanje problema.

Kubernetes resursi

Kubernetes resursi su ključni entiteti koji omogućavaju upravljanje aplikacijama i njihovom konfiguracijom unutar Kubernetes klastera. Predstavljaju sve komponente potrebne za ispravan rad i komunikaciju aplikacija unutar klastera. Glavni resursi uključuju:

Podovi (eng. "Pods") [18] Najmanja i najjednostavnija jedinica u Kubernetesu koja može biti raspoređena. Podovi sadrže jedan ili više kontejnera koji dijele istu mrežu i skladište. Svaki pod ima jedinstvenu IP adresu i može komunicirati s drugim podovima unutar klastera. Podovi omogućavaju lako raspoređivanje, skaliranje i upravljanje aplikacijama.

Servisi (eng. "Services") [19] Apstrakcija koja definiše logički skup podova i politiku za pristup njima. Servisi omogućavaju stabilnu IP adresu i DNS ime za skup podova, što omogućava jednostavno balansiranje opterećenja i otkrivanje usluga. Oni omogućavaju da aplikacije budu otpornije i skalabilnije, bez obzira na to kako se podovi mijenjaju.

Deployment-i [20] (korišten izvorni oblik na engleskom uslijed nedostatka adekvatnog prevoda) Resurs koji pruža deklarativni način za ažuriranje aplikacija. Deployment-i upravljaju replikama podova i omogućavaju vraćanje na prethodne verzije u slučaju problema. Oni također omogućavaju postupno uvođenje novih verzija aplikacija bez prekida rada, što je ključno za kontinuirano integrisanje i isporuku (CI/CD).

Ingresi (eng. "Ingresses") [21] Resurs koji upravlja pristupom spoljnjim korisnicima do servisa unutar klastera, obično putem HTTP-a ili HTTPS-a. Ingress pravila definišu kako se zahtjevi usmjeravaju do različitih servisa na osnovu URL putanje ili domenskom imenu u zahtjevu. Ovo omogućava složenije scenarije usmjeravanja prometa i centralizovano upravljanje pristupom.

Konfiguracijske mape (eng. "Config maps") [22] Koriste se za upravljanje konfiguracijskim informacijama koje nisu osjetljive, kao što su konfiguracijske datoteke, varijable okruženja itd. One omogućavaju da aplikacije budu konfigurabilne bez potrebe za ponovnom izgradnjom kontejnera. Ovo omogućava brže i fleksibilnije promjene u konfiguraciji.

Tajne (eng. "Secrets") [23] Koriste se za pohranu i upravljanje osjetljivim informacijama kao što su lozinke, OAuth tokeni, SSH ključevi itd. Tajne osiguravaju da osjetljive informacije budu sigurno pohranjene i dostupne samo onim podovima i servisima koji ih trebaju. Ovo pomaže u zaštiti aplikacija i podataka od neovlaštenog pristupa.

Replika skupovi (eng. "ReplicaSets") [24] Upravljaju brojem identičnih podova koji rade u klantru. Osiguravaju da određeni broj replika podova bude uvijek pokrenut. Ako pod ne uspije, automatski se stvara novi pod da zamijeni onaj koji je propao.

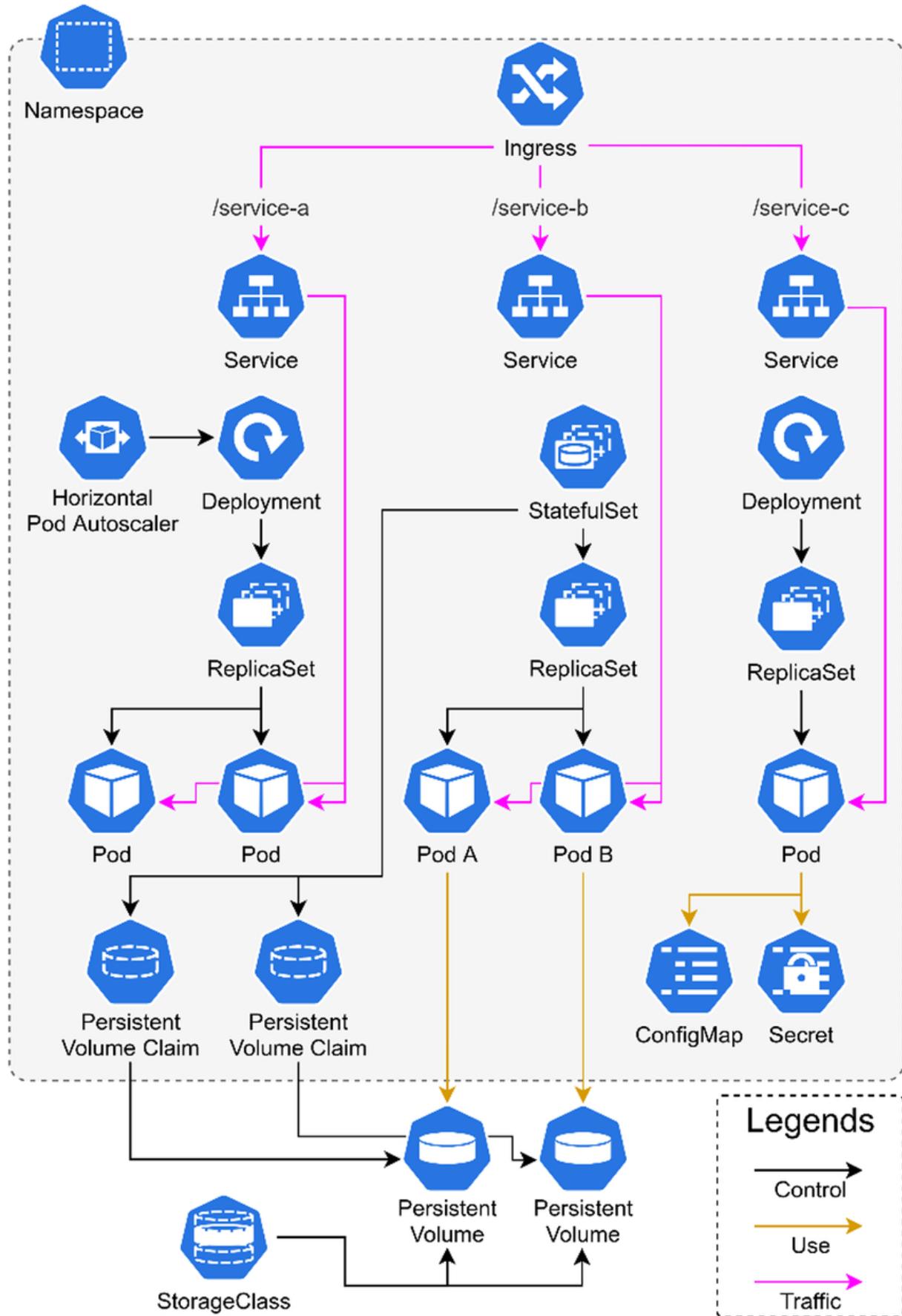
Skupovi sa stanjem (eng. "StatefulSets") [25] dizajniran je za upravljanje aplikacijama koje zahtijevaju stabilne identitete mreže i trajno stanje. Za razliku od Deployment-a, koji su namijenjeni za aplikacije koje ne čuvaju stanje, skup sa stanjem je idealan za takve aplikacije kao što su baze podataka ili distribuirani sistemi.

Poslovi i vremenski poslovi (eng. "Jobs and CronJobs") [26] Poslovi kao Kubernetes resurs se koriste za pokretanje poslova koji trebaju biti izvršeni jednom do završetka, dok vremenski poslovi raspoređuju zadatke koji se trebaju ponavljati u određenim intervalima. Kao posao se može zadati izvršavanje bilo kakvog zadatka, kao što je npr. pokretanje neke Bash skripte.

PV i PVC [27] (skraćeno za engleske nazive "**Persistent Volumes**" i "**Peristent Volume Claims**", respektivno) su resursi za upravljanje trajnim skladištem podataka. PV je skladišni resurs koji administratori konfigurišu, dok PVC predstavlja zahtjev korisnika za skladištem. PVC omogućava podovima da koriste trajno skladište koje ostaje dostupno čak i ako se podovi ponovo raspoređuju ili brišu.

Korisnički kreirani resursi (eng. "Custom resources") [28] Kubernetes omogućava korisnicima da kreiraju vlastite resurse pomoću eng. "Custom Resource Definitions" (CRD). Ovo omogućava proširenje Kubernetesa sa specifičnim resursima prilagođenim potrebama aplikacija ili organizacija. Korisnički kreirani resursi omogućavaju korisnicima da definišu i upravljaju novim vrstama objekata, proširujući mogućnosti Kubernetesa bez potrebe za modifikacijom osnovnog koda.

Ovi resursi igraju ključnu ulogu u omogućavanju deklarativnog upravljanja aplikacijama u Kubernetes klanterima, što čini Kubernetes snažnim alatom za orkestraciju kontejnera. Također, svi ovi resursi čine jednu cjelinu načinom na koji su oni zapravo uvezani i načinom njihove komunikacije. Cjelokupan odnos između resursa prikazan je na slici 2.7.



Slika 2.7: Kubernetes resursi [4]

Napredne teme

Kubernetes je jako širok i kompleksan alat. Postoje mnogi načini konfigurisanja i postavljanja samog klastera za prilagođavanje korisničkim potrebama. Osim do sada prethodno navedenih resursa i funkcionalnosti, postoje i dodatne koje omogućavaju bolju organizaciju, sigurnost i efikasnost u upravljanju klasterima. Ove napredne teme uključuju imenski prostor, kontrola pristupa na osnovu rola (eng. "Role-Based Access Control" - RBAC) i Helm.

Imenski prostor (eng. "namespace") [29] omogućava logičku izolaciju resursa unutar klastera. Koristi se za odvajanje različitih okruženja kao što su razvojno, testno i produksijsko okruženje, kao i za upravljanje resursima različitih timova ili projekata unutar istog klastera. Svaki imenski prostor ima svoje resurse, što omogućava bolju organizaciju i kontrolu pristupa.

RBAC (eng. "Role-Based Access Control") [30] je mehanizam za kontrolu pristupa koji omogućava granularno upravljanje pristupnim pravima korisnika i servisa u klasteru. Administratori mogu definisati uloge i pridružiti im dozvole za obavljanje specifičnih akcija na resursima unutar određenih imenskih prostora, što povećava sigurnost i sprječava neovlašteni pristup.

Helm [31] je alat za upravljanje paketima u Kubernetesu, često nazivan menadžerom paketa za Kubernetes. Omogućava definisanje, instalaciju i nadogradnju kompleksnih Kubernetes aplikacija pomoću Helm paketa. Paketi su kolekcije datoteka koje opisuju Kubernetes resurse potrebne za pokretanje aplikacije. Helm olakšava ponovno korištenje i dijeljenje aplikacija između timova.

Primjena Kuberntes-a

Kubernetes se koristi u različitim okruženjima i za različite primjene zbog svoje fleksibilnosti i moćnih funkcionalnosti. Primjene uključuju upravljanje mikroservisima, skaliranje aplikacija, orkestraciju kontejnera i upravljanje trajnim stanjem.

Kubernetes je idealan za upravljanje mikroservisima jer omogućava jednostavno raspoređivanje, skaliranje i ažuriranje servisa. Pruža alate za automatsko skaliranje aplikacija na osnovu opterećenja, čime se osigurava optimalna upotreba resursa. Orkestracija kontejnera omogućava automatizaciju raspoređivanja, upravljanja i skaliranja kontejnera, smanjujući potrebu za ručnim intervencijama. Upravljanje trajnim stanjem putem PV omogućava aplikacijama da čuvaju podatke čak i ako se podovi premještaju ili ponovo kreiraju, što je ključno za aplikacije kao što su baze podataka, te se samim time može vidjeti i primjena Kuberntesa za upravljanje istim. Kubernetes je također široko korišten za implementaciju CI/CD (kontinuirane integracije i isporuke), omogućavajući timovima da brže i efikasnije isporučuju promjene u aplikacijama. Kombinacija s alatima kao što su Helm olakšava upravljanje složenim aplikacijama i njihovim ovisnostima.

Ove primjene čine Kubernetes univerzalnim rješenjem za moderne aplikacije koje zahtijevaju skalabilnost, otpornost i efikasno upravljanje resursima.

Sigurnost u Kuberntesu

Sigurnost u Kuberntesu obuhvata niz praksi i alata dizajniranih za zaštitu klastera, aplikacija i podataka [16]. Ključni aspekti sigurnosti uključuju kontrolu pristupa, enkripciju podataka, sigurnosne politike i redovno ažuriranje sistema. RBAC omogućava granularno upravljanje

pristupom resursima u klasteru. Administratori mogu definirati uloge i dozvole, osiguravajući da korisnici i servisi imaju samo one privilegije koje su im potrebne. Enkripcija podataka je ključna za zaštitu osjetljivih informacija. Kubernetes podržava enkripciju podataka u mirovanju i u prenosu, koristeći tajne za sigurno pohranjivanje osjetljivih podataka. Sigurnosne politike definiraju pravila za dopuštanje ili blokiranje određenih akcija unutar klastera. Mrežne politike (eng. "Network Policies") omogućavaju kontrolu mrežnog prometa između podova, dok politike sigurnosti podova (engl "Pod Security Policies") određuju sigurnosne standarde koje podovi moraju ispunjavati [16].

Redovno ažuriranje Kubernetes klastera i komponenti pomaže u zaštiti od poznatih sigurnosnih ranjivosti. Automatizirani alati i skripte mogu pomoći u održavanju ažuriranosti klastera bez prekida rada. Kombinacija ovih praksi i alata pomaže u osiguravanju robustne sigurnosne posture za Kubernetes klaster.

Izazovi i ograničenja

Kubernetes, iako široko korišten alat za orkestraciju kontejnera, suočava se s nizom izazova i ograničenja. Jedan od glavnih izazova je složenost upravljanja klasterima, koja može zahtijevati značajne resurse i stručnost. Postavljanje, konfiguracija i održavanje klastera može biti kompleksno, posebno za timove bez prethodnog iskustva. Skalabilnost aplikacija može donijeti nepredviđene probleme s performansama, mrežnom komunikacijom i skladištem podataka. Upravljanje skalabilnošću zahtijeva pažljivo planiranje i optimizaciju resursa kako bi se osigurala efikasnost i stabilnost. Sigurnosni rizici također predstavljaju izazov, s obzirom na potencijalne prijetnje kao što su neovlašteni pristup, ranjivosti unutar kontejnera i potrebe za stalnim ažuriranjem sigurnosnih politika.

Troškovi su još jedan značajan faktor, jer upravljanje Kubernetes klasterima može biti skupo zbog potrebne infrastrukture, licence za softver i troškova za obuku i podršku. Praćenje i optimizacija troškova su ključni za održavanje isplativosti.

Konačno, interoperabilnost s postojećim sistemima i alatima može biti izazovna. Integracija Kubernetes klastera sa starijim aplikacijama ili različitim devops alatima može zahtijevati dodatne prilagodbe i razvoj, što povećava kompleksnost projekta.

Ovi izazovi i ograničenja zahtijevaju pažljivo planiranje, stručnost i kontinuirano upravljanje kako bi se osigurala uspješna implementacija i operacija Kubernetes klastera. Međutim, Kubernetes ipak ima veći doprinos te definitivno manji broj izazova i ograničenja nego doprinosa u kvalitetnijem razvoju softvera.

* * *

Poglavlje o DevOpsu pružilo je sveobuhvatan pregled ključnih aspekata koji čine ovu metodologiju ključnom za moderni razvoj softvera. Razlike između tradicionalnih pristupa i DevOps-a naglašene su u kontekstu agilnosti i brzine isporuke. Kontejnerizacija, predstavljena kroz Docker, demonstrirala je prednosti izolacije i efikasnosti resursa. Orkestracija, s posebnim fokusom na Kubernetes, pokazala je kako se kompleksne aplikacije mogu efikasno upravljati i skalirati. Kroz analizu ključnih funkcionalnosti, sigurnosnih izazova i budućih trendova, prikazano je kako se DevOps pristup može efikasno koristiti i pri radu i upravljanju bazama podataka.

U narednom poglavlju će ovaj pristup upravljanja bazama biti upoređen sa tradicionalnim u kontekstu samih baza podataka.

Poglavlje 3

Poređenje DevOps i tradicionalnog pristupa pri upravljanju bazama podataka

Upravljanje bazama podataka predstavlja kritičan aspekt u modernim IT okruženjima, gdje se od baza podataka zahtijeva visoka dostupnost, skalabilnost i sigurnost. Tradicionalni pristup upravljanju bazama podataka oslanja se na strogo definirane procese i alate koje implementiraju administratori baza podataka (DBA). Ovaj pristup, iako pouzdan, često može biti spor i manje fleksibilan u dinamičnim okruženjima.

S druge strane, DevOps pristup, koji integrira razvoj i operativne timove, uvodi principe automatizacije, kontinuirane integracije i kontinuirane isporuke (CI/CD). DevOps pristup omogućava brže isporuke, bolju suradnju među timovima i povećanu agilnost u upravljanju bazama podataka.

Cilj ovog poglavlja je detaljno istražiti i usporediti oba pristupa, ističući njihove prednosti, nedostatke i prikladnost za različite poslovne potrebe.

3.1 Generalno poređenje DevOps i tradicionalnog pristupa

Upravljanje IT operacijama može se provoditi kroz dva različita pristupa: tradicionalni pristup i DevOps pristup. Ova sekcija analizira ključne razlike između ovih pristupa, s naglaskom na njihove osnovne principe, metode rada i ciljeve.

3.1.1 Principi i metode rada

Tradisionalni pristup karakterizira odvojenost razvojnih i operativnih timova. Ovi timovi imaju jasno definirane odgovornosti, što često dovodi do dugih ciklusa razvoja i isporuke. Procesi su strogo definirani, a administracija koristi specifične alate za svoje zadatke. Ovaj pristup osigurava stabilnost i pouzdanost, ali može biti manje fleksibilan u dinamičnim okruženjima.

S druge strane, DevOps pristup integrira razvojne i operativne timove, promovirajući suradnju i zajedničku odgovornost za cijeli proces razvoja i isporuke. Automatizacija i kontinuirana isporuka (CI/CD) ključni su elementi DevOps prakse, omogućavajući brže isporuke i povratne informacije. DevOps timovi koriste alate za automatizaciju, orkestraciju i nadziranje, što povećava agilnost i efikasnost.

3.1.2 Ciljevi i rezultati

Ciljevi tradicionalnog pristupa usmjereni su na stabilnost i pouzdanost. Kontrolirani procesi i stroga sigurnosna pravila osiguravaju da sistemi budu sigurni i pouzdani. Međutim, sporiji ciklusi isporuke mogu ograničiti brzinu inovacija i prilagodbu novim zahtjevima.

DevOps pristup teži povećanju agilnosti i brzine isporuke. Brža implementacija promjena i poboljšanja omogućava organizacijama da se brzo prilagode promjenama na tržištu i implementiraju nove funkcionalnosti. DevOps također potiče inovacije, omogućavajući brže usvajanje novih tehnologija i praksi.

3.1.3 Prednosti i izazovi

Prednosti tradicionalnog pristupa uključuju visoku stabilnost i sigurnost, ali su ograničene sporijim isporukama i manjom fleksibilnošću. Nasuprot tome, DevOps pristup donosi veću agilnost i efikasnost, ali zahtijeva promjenu organizacijske kulture i uvođenje novih alata i praksi. Integracija razvojnih i operativnih timova može biti izazovna, ali donosi značajne prednosti u smislu bržeg i efikasnijeg upravljanja IT operacijama.

3.2 Poređenje DevOps i tradicionalnog pristupa pri upravljanju bazama podataka

Nakon generalnog poređenja navedena dva pristupa, u ovoj sekciji će biti razmatrano ponovno poređenje ali konkretno u kontekstu rada sa bazama podataka.

3.2.1 Principi i metode rada

Tradisionalni pristup upravljanju bazama podataka oslanja se na odvojene timove za razvoj i operacije [8]. Administratori baza podataka (DBA) su odgovorni za sve aspekte upravljanja bazama, uključujući sigurnosne kopije, obnavljanje, optimizaciju performansi i sigurnost. Procesi su često strogo definirani, a promjene se uvode kroz formalne procedure koje mogu biti spore i manje fleksibilne.

DevOps pristup integrira razvojne i operativne timove, promovirajući suradnju i zajedničku odgovornost za baze podataka. Automatizacija je ključna komponenta, s fokusom na kontinuiranu integraciju i isporuku (CI/CD). Alati za automatizaciju, orkestraciju i nadziranje baza podataka omogućavaju brže i efikasnije upravljanje, smanjujući vrijeme potrebno za implementaciju promjena i poboljšanja. [8]

3.2.2 Ciljevi i rezultati

Tradisionalni pristup prioritizira stabilnost i sigurnost baza podataka. Kroz strogo kontrolirane procese osigurava se pouzdanost sistema, ali često po cijenu sporijih ciklusa isporuke i manje fleksibilnosti u prilagodbi novim zahtjevima.

DevOps pristup naglašava agilnost i brzinu isporuke, omogućavajući organizacijama da se brzo prilagode promjenama u zahtjevima i tehnologijama. Implementacija promjena i novih funkcionalnosti je brža, što potiče inovacije i omogućava efikasnije upravljanje bazama podataka u dinamičnim okruženjima.

3.2.3 Prednosti i izazovi

Prednosti tradicionalnog pristupa uključuju visoku stabilnost i sigurnost baza podataka, ali su često praćene sporijim ciklusima isporuke i manjom fleksibilnošću te često i većom cijenom. Nasuprot tome, DevOps pristup omogućava veću agilnost i efikasnost, ali zahtijeva promjenu organizacijske kulture i uvođenje novih alata i praksi. Integracija razvojnih i operativnih timova može biti izazovna, ali donosi značajne prednosti u smislu bržeg i efikasnijeg upravljanja bazama podataka.

* * *

Poglavlje o poređenju DevOps i tradicionalnog pristupa pri upravljanju bazama podataka pružilo je dubinski uvid u ključne razlike između ovih metoda. Tradicionalni pristup naglašava stabilnost i sigurnost kroz strogo definirane procese, ali pati od sporije isporuke i manje fleksibilnosti. DevOps pristup, s fokusom na automatizaciju i suradnju timova, omogućava bržu adaptaciju i efikasnije upravljanje bazama podataka. Iako zahtijeva promjenu kulture i uvođenje novih alata, DevOps donosi značajne prednosti u agilnosti i inovacijama. U narednom poglavlju

Poglavlje 4

Primjer korištenja kontejnerizacijskih i orkestracijskih tehnika za upravljanje DB serverom

U ovom poglavlju bit će demonstriran (eng. "Proof of Concept" - PoC) koncept implementacije i orkestracije PostgreSQL baza podataka koristeći DevOps alate.

Kao prvo, potrebno se nakratko osvrnuti na izbor sistema za upravljanje bazama podataka (eng. " DataBase Management System" - DBMS). PostgreSQL server baza podataka je izabran iz nekoliko razloga. Prije svega, riječ je o jednom od najpopularnijih sistema za upravljanje bazama podataka. To je DBMS otvorenog koda i doživljava veliku popularnost u radu sa kontejnerizacijskim tehnologijama što donosi razne prednosti kao npr. **PGWatch2** alat za nadziranje PostgreSQL baza podataka. Također, najpopularniji napravljeni Kubernetes DBMS operatori su baš oni napravljeni za PostgreSQL. Izvođenje ovog primjera bi naravno bio moguć i za druge DBMS, ali u ovom konkretnom slučaju PostgreSQL olakšava implementaciju i rješavanje problema svojom podržanošću i funkcionalnostima.

Cilj ovog PoC-a je demonstrirati upotrebu kontejnerizacijskih i orkestracijskih tehnika za učinkovito upravljanje bazama podataka, što je ključno za moderne aplikacije koje zahtijevaju skalabilnost, dostupnost i jednostavno održavanje.

Za implementaciju ovog praktičnog dijela rada, korišteni su poznati alati, često primijenjeni u DevOps metodologiji:

1. Docker
2. Kubernetes
3. Lens
4. Helm
5. ArgoCD
6. Grafana
7. PGWatch2

Dakle, korištenim alatima je obuhvaćen veliki opseg principa DevOps metodologije - Docker za kontejnerizaciju, Kubernetes za orkestraciju, Lens za olakšano upravljanje klasterom, Helm za automatizaciju, ArgoCD za CI/CD, Grafana za nadziranje rada te PGWatch2 za lakše nadziranje. U nastavku će biti opisan kompletan tok implementacije korak po korak (da bi čitalac mogao stvoriti što bolju sliku o načinu rada i o samom DevOps pristupu), alati koji su korišteni (radi boljeg razumijevanja samog urađenog i implementiranog primjera), izazovi pri implementaciji te krajnje dobiveno rješenje.

Sav kod korišten pri implementaciji samog rješenja se može pronaći na narednom GitHub rezervoriju: <https://github.com/ahmedpasic/devops-db-management>.

4.1 Priprema okruženja

Na samom početku, potrebno je odrediti koja platforma će se koristiti za samu orkestraciju. Kao što je već rečeno, najpopularniji alat za te svrhe jeste Kubernetes. Dostupan je kod skoro svih pružatelja usluge u oblaku te oni imaju svoju posebnu Kubernetes implementaciju. Korisnici mogu vrlo jednostavno u veoma kratkom periodu (u roku par minuta) imati ispravno ustavljjen, konfigurisan i pokrenut klaster.

Međutim, u svrhe ovog primjera korištena je druga opcija, a u pitanju je pokretanje Kuberntesa na lokalnom uređaju što je moguće jer je Kubernetes otvorenog koda. Postoje različiti alati koji omogućavaju pokretanje Kuberntesa lokalno, među kojima su najpoznatiji:

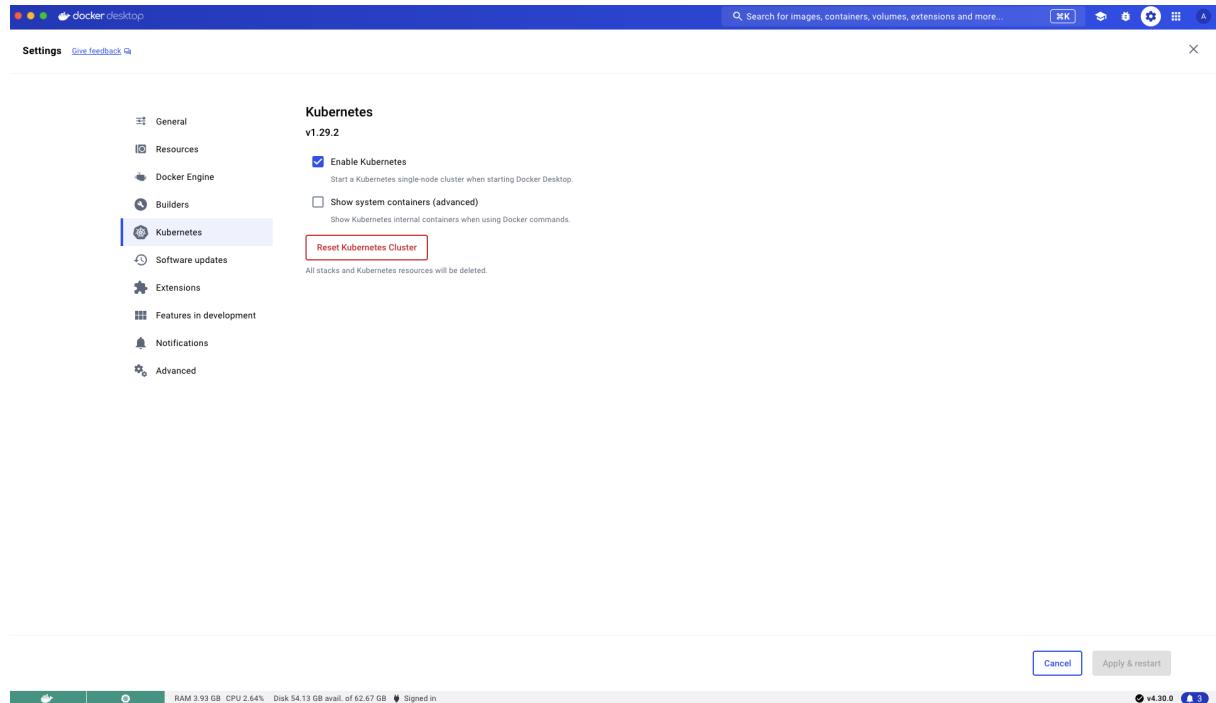
1. Minikube
2. Kind
3. Docker Desktop
4. K3D

Za primjer opisan u ovom poglavlju izabran je Docker Desktop. Njegove prednosti u odnosu na ostatak navedenih alata jeste njegova jednostavnost korištenja, brzina kao i pouzdanost. Ostale opcije koje su navedene imaju neke prednosti u odnosu na izabranu. Na primjer, K3D omogućava podizanje klastera sa više čvorova, što nije moguće ako se koristi Docker Desktop. Također, Minikube i K3D imaju ugrađen server za metrike klastera. Još jedan nedostatak je neposjedovanje interfejsa komandne linije (eng. "Command Line Interface" - CLI). Ove prednosti mogu biti značajne ako se klaster koristi u produkcijskom okruženju.

U ovom slučaju, jednostavnost, brzina i pouzdanost koju pruža Docker Desktop je značajnija od navedenih nedostataka.

Pokretanje klastera je veoma jednostavno. Unutar Docker Desktop grafičkog korisničkog interfejsa, u postavkama, nalazi se opcija "Enable Kubernetes". Spašavanjem izbora opcije, Docker Desktop u pozadini kreira klaster, ali ne na standardan način (kreiranjem virtualne mašine koja će služiti kao čvor u klasteru), nego Docker Desktop koristi Docker kontejnere za pokretanje Kubernetes komponenti unutar jedne lokalne Docker maštine. Na slici 4.1 prikazan je korisnički interfejs gdje se može omogućiti Kubernetes klaster u Docker Desktop-u.

Primjer korištenja kontejnerizacijskih i orkestracijskih tehnika za upravljanje DB serverom



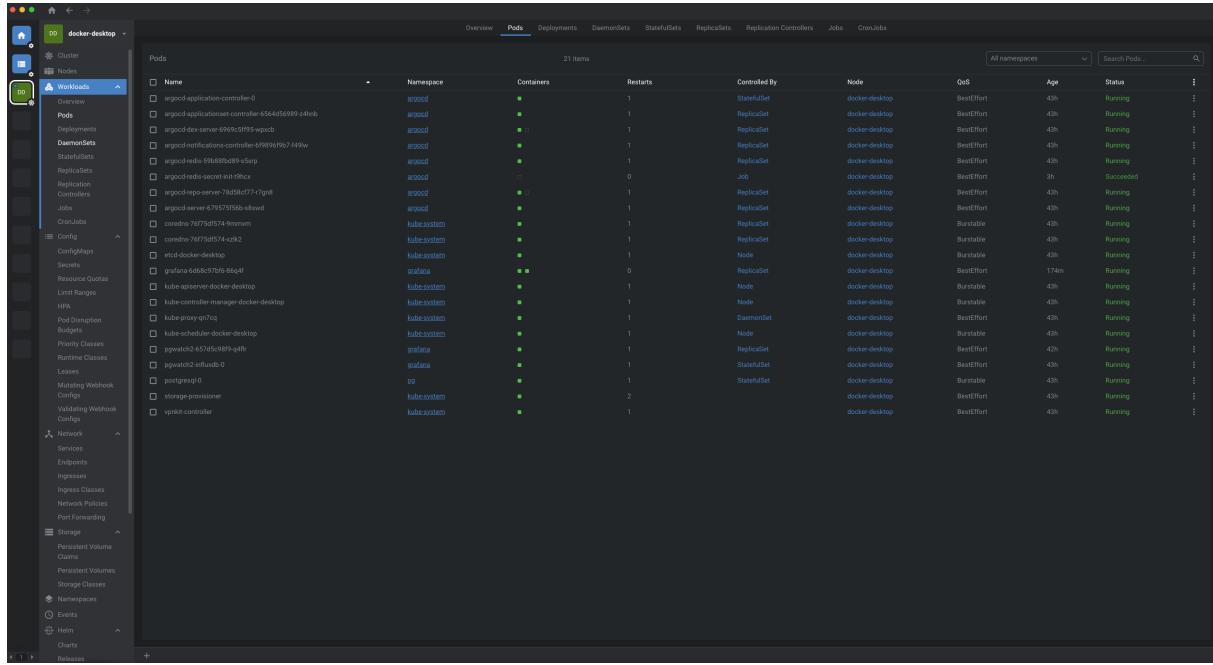
Slika 4.1: Pokretanje klastera - Docker Desktop

Prethodni korak je dovoljan za uspostavljanje ispravnog Kubernetes klastera. Sam klaster je dostupan te se njim može upravljati alatom kao što je "kubectl", klijent koji omogućava slanje API poziva prema klasteru od strane korisnika.

Još jedan bitan korak koji obavlja Docker Desktop je dodavanje konfiguracije klastera, tzv. "kubeconfig".

To je konfiguraciona datoteka koja se koristi za interakciju sa Kubernetes klasterom. U njoj se nalaze informacije o klasteru, korisnički podaci, konteksti i drugi parametri potrebni alatima kao što su kubectl za uspostavljanje veze sa klasterom. Obično sadrži informacije o klasteru, poput IP adrese API servera, porta i certifikata za sigurnu komunikaciju. Nalazi se na putanji `./kube/config`.

Nakon što je ova konfiguraciona datoteka dodana na odgovarajuću putanju, može se koristiti "kubectl" klijent za upravljanje samim klasterom. Međutim, radi jednostavnijeg rada sa istim, ovdje je korišteno grafičko okruženje pod nazivom **Lens**. Lens je grafičko okruženje za olakšan način izvršavanja kubectl komandi nad klasterom. Pomoću ovog alata moguće je upravljati svim Kubernetes resursima i komponentama koje su navedene u 2.3.3. Na slici 4.2 može se vidjeti kako izgleda sama aplikacija, a u lijevom meniju prikazani su svi Kubernetes resursi nad kojima je omogućeno upravljanje.



Slika 4.2: Izgled Lens aplikacije

4.2 Stavljanje aplikacija u upotrebu

Korak uspostavljanja klastera je ispunio sve preuvjetne potrebne za stavljanje potrebnih aplikacija u upotrebu. Sada, potrebno je odlučiti kojim načinom doći do krajnjeg cilja, to jest da je server baza podataka dostupan unutar klastera te da se može koristiti. Dakle, potrebno je pokrenuti Docker kontejner koji sadrži sve neophodne zavisnosti i biblioteke za ispravan rad servera.

Za pokretanje kontejnera potrebna je Docker slika. Docker slika je temeljna jedinica za distribuciju i izvršavanje softvera pomoću Docker platforme. Ona sadrži sve potrebne informacije i resurse za pokretanje aplikacije, uključujući operativni sistem, kod, zavisnosti, sistemske biblioteke, alate i konfiguraciju. Slike se sastoje od slojeva pri čemu se može uzeti analogija slojeva na slici: svaki sloj dodaje ili mijenja specifične dijelove slike [32]. To omogućava da se slike dijele i koriste u različitim okruženjima bez potrebe za ponovnim konfiguriranjem aplikacije, što olakšava razvoj, distribuciju i upravljanje aplikacijama. Umjesto ručnog kreiranja Docker slike za PostgreSQL kao i za bilo koju drugu komponentu u ovom primjeru, koriste se gotove Docker slike dostupne na javnim repozitorijima unutar registra kao sto je "Docker Hub". Pri tome je najbolje koristiti oficijalne slike napravljene od strane samih vlasnika aplikacija čije se slike žele preuzeti.

Taj kontejner će biti pokrenut unutar Kubernetes poda, kao najmanja jedinica te i najmanji resurs unutar klastera. Pod se može staviti u upotrebu tzv. manifest datotekom, datoteka u YAML formatu koja sadrži potrebnu konfiguraciju za pod. Svi Kubernetes resursi se mogu staviti u upotrebu na ovaj način. Međutim, za neku ozbiljniju i komplikovaniju aplikaciju (kao što je npr. PostgreSQL, a i sve druge aplikacije u današnje vrijeme) samo pod u kojem bi bio pokrenut kontejner ne bi bio dovoljan. Potrebno je popratnih resursa, kao što su servisi (za konfiguriranje pristupa PostgreSQL-u), PVC i PV za podešavanje trajne pohrane podataka itd. Ako bismo željeli izbrisati sve resurse vezane za PostgreSQL morali bismo brisati jedan po jedan, u biti, nisu povezani u jednu cjelinu. Uz to, teško je upravljati verzijama, resursi se moraju stavljati u upotrebu pojedinačno, postoji manjak ponovne upotrebljivosti itd. Iz ovih razloga postoji alat

pod nazivom **Helm**.

4.2.1 Helm

Helm je alat koji znatno pojednostavljuje upravljanje Kubernetes aplikacijama. Njegove prednosti uključuju jednostavnu instalaciju i nadogradnju, podršku za vraćanje na stare verzije (eng. "rollback"), dinamičku konfiguraciju, lako dijeljenje i smanjenje kompleksnosti upravljanja složenim aplikacijama. Korištenje Helma pomaže u automatizaciji procesa raspoređivanja i održavanja aplikacija u Kubernetes klasteru, čineći ih pouzdanim i lakšim za upravljanje.

Glavna komponenta Helma jesu paketi (eng. "charts"). To su paketne definicije za Kubernetes aplikacije koje sadrže sve potrebne resurse i konfiguracije za njihovo pokretanje. Helm paketi omogućavaju jednostavnu instalaciju, nadogradnju i upravljanje aplikacija unutar Kubernetes klastera. Svaki paket je kolekcija datoteka koje definišu skup povezanih Kubernetes resursa. U nastavku je prikazana struktura Helm paketa pod imenom "mychart".

```
1 mychart/
2   Chart.yaml          # Metapodaci o paketu (ime, verzija, opis)
3   values.yaml         # Podrazumijevane vrijednosti za konfiguraciju
4   charts/             # Direktorij za zavisnosti (drugi paketi)
5   templates/          # Direktorij sa Kubernetes manifest datotekama
6   koji koriste sablonski jezik
7   LICENSE             # Licenca (opcionalno)
8   README.md           # Dokumentacija o paketu (opcionalno)
```

Unutar direktorija "templates" nalaze se šabloni za kreiranje svih resursa potrebnih za razmatranu aplikaciju, pri čemu se podrazumijevane vrijednosti mogu zamijeniti vlastitim vrijednostima za konkretni slučaj primjene.

Slično kao kod Docker slika, za poznate aplikacije Helm paketi su javno dostupni te ih nije potrebno u potpunosti pisati. Svi paketi korišteni u repozitoriju, koji je naveden na kraju samog uvida poglavlja, su preuzeti s javnih repozitorija te su slobodni za korištenje od strane bilo kojih korisnika. U nastavku su prikazane najčešće korištene komande za upravljanje Helm paketima:

Program 4.1: Upravljanje Helm paketima

```
1 # Dodavanje repozitorija u lokalno dostupne
2 helm repo add grafana https://grafana.github.io/helm-charts
3
4 # Azuriranje lokalnih repozitorija
5 helm repo update
6
7 # Preuzimanje paketa lokalno u trenutni direktorij
8 helm pull grafana/grafana
9
10 # Instaliranje paketa, ovim sve potrebne komponente
11 # bivaju stavljene u upotrebu unutar klastera
12 # helm install - prvi parametar ime izdanja aplikacije
13 # drugi parametar je putanja do Helm paketa
14 helm install grafana ./grafana -f myvalues.yaml
```

4.2.2 ArgoCD

Validan način upravljanja svim potrebnim aplikacijama unutar klastera bi bio koristiti Helm, kreirati bash skripte koje izvršavaju odgovarajuće Helm komande (što će biti vidljivo na primjeru alata ArgoCD), te te skripte koristiti unutar CI/CD cijevi u nekom alatu kao što je npr. Jenkins.

Međutim, za primjer koji se ovdje obrađuje korišten je relativno nov pristup upravljanju infrastrukturom i aplikacijama, koji se pojavio prvi put 2017. godine. Poznat pod imenom GitOps. GitOps je pristup upravljanju infrastrukturom i aplikacijama koristeći Git kao izvor istine za deklarativne konfiguracije i automatizaciju operacija. GitOps koristi principe kontinuirane isporuke (CD) i infrastrukture kao koda (IaC) kako bi omogućio efikasno, sigurno i ponovljivo upravljanje infrastrukturom.

Princip na kojem se temelji jeste da je git repozitorij jedini izvor istine. Razmatra se samo konfiguracija unutar repozitorija. Sve što se nalazi izvan njega se ne smatra relevantnim. Često u različitim izvedbama GitOps-a je čak i onemogućeno mijenjanje bilo kakve konfiguracije na bilo koji način osim kroz specificiranje unutar koda repozitorija.

GitOps izvedba koja je korištena u svrhe rješavanja problema ovog rada jeste ArgoCD. Taj GitOps alat je prvi put najavljen i otvoren za javnost u oktobru 2018. godine, i kako brzo je doživio veliku popularnost.

Ideja je sljedeća. Unutar ArgoCD-a se specificira korisnički kreiran resurs - ArgoCD aplikacija. Jedna ArgoCD aplikacija odgovara aplikaciji koja će biti stavljena u upotrebu pomoću ArgoCD-a. Pri definiciji aplikacije se specificira putanja do Helm paketa za istu te također i putanja do korisnički definiranog values.yaml. Nakon definisanja aplikacije, ArgoCD prati promjene na definiranim putanjama te ukoliko postoje, prijavljuje korisniku ili samostalno vrši prilagođavanje tim promjenama ukoliko je takvo ponašanje zadano. Ovim se postiže mogućnost da je zapravo kraj posla čovjeka pri postavljanju aplikacije u upotrebu trenutak kada korisnik obavi "git push". Primjer definiranja aplikacije unutar ArgoCD alata je prikazan u kodu 4.2

Program 4.2: Konfiguracija ArgoCD aplikacije

```
1  - apiVersion: argoproj.io/v1alpha1
2  kind: Application
3  metadata:
4    name: postgresql
5    namespace: argocd
6  spec:
7    project: default
8    source:
9      repoURL: 'https://github.com/ahmedasic/devops-db-management'
10     path: charts/postgresql
11     targetRevision: HEAD
12     helm:
13       valueFiles:
14         - ../../deploy/postgresql/values.yaml
15   destination:
16     server: 'https://kubernetes.default.svc'
17     namespace: pg
18   syncPolicy:
19     managedNamespaceMetadata:
20       labels:
21         name: pg
22         app.kubernetes.io/name: pg
23         app.kubernetes.io/instance: pg
```

```
24     app.kubernetes.io/project: postgresql
25   syncOptions:
26     - CreateNamespace=true
```

Sam ArgoCD je program koji radi unutar samog klastera. Moguće je čak i da ArgoCD upravlja sam sobom. Prvo se mora ručno pokrenuti ArgoCD i staviti u upotrebu unutar klastera, nakon toga se može podesiti ArgoCD aplikacija i za sam ArgoCD. To nije urađeno u ovom primjeru te je ArgoCD jedina komponenta unutar klastera kojom ne upravlja isti nego je stavljena u upotrebu putem bash skripte prikazane u kodu 4.3. Skripta koristi Helm komandu. Time će Helm izdanje biti dostupno unutar klastera u posebnom imenskom prostoru.

Program 4.3: Postavljanje ArgoCD-a u upotrebu

```
1 #!/bin/bash
2
3 NAMESPACE="argocd"
4
5 helm upgrade argocd ../../charts/argo-cd \
6   --install --create-namespace --namespace "${NAMESPACE}" \
7   -f values.yaml
```

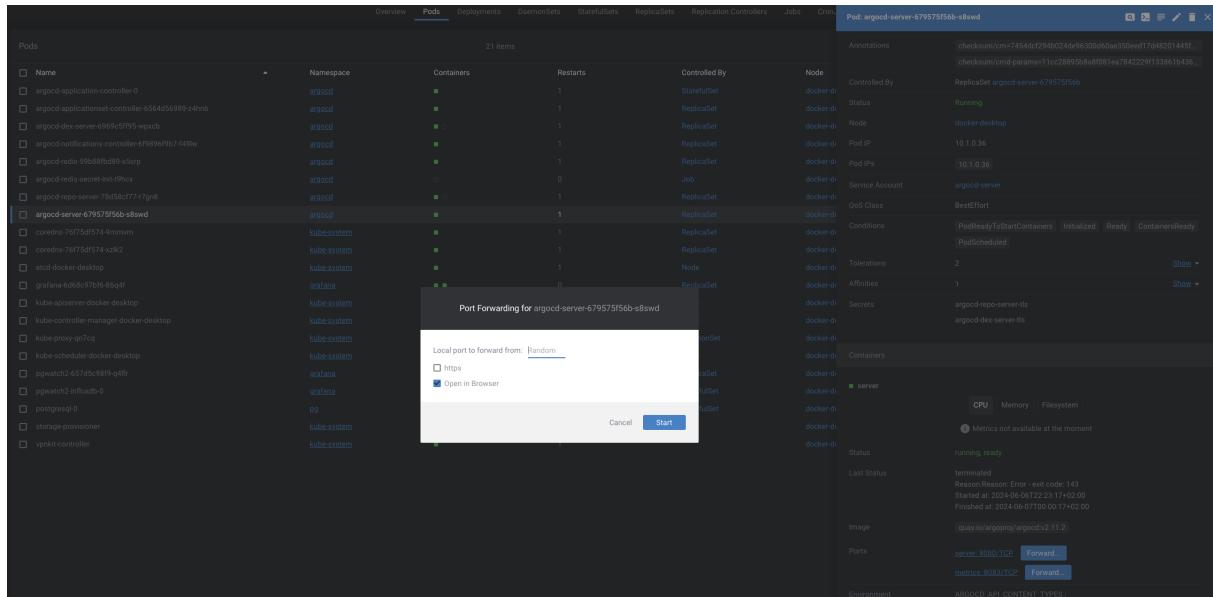
Nakon izvršavanja skripte komandom *bash deploy.sh*, ArgoCD je dostupan u klasteru. Daљe, postavlja se pitanje kako pristupiti korisničkom interfejsu istog.

U ovom slučaju postoje dvije opcije. Prva jeste da se koristi prosljeđivanje porta pomoću Kubernetes klijenta, *kubectl*. To podrazumijeva mapiranje porta sa stvarnog uređaja na Kubernetes klaster, tačnije na Kubernetes čvor na kojem se nalazi zadani pod, tj. kontejner na koji želimo namapirati.

Druga opcija je korištenje *NodePort* servisa dostupnog u Kubernetesu. Kada se kreira *NodePort* servis, Kubernetes će mapirati odabrani port na svakom čvoru u klasteru na odabrani port u servisu. To znači da se može pristupiti aplikaciji preko IP adrese i odabranog port-a na bilo kom čvoru u klasteru. Međutim, u lokalnom klasteru putem Docker Desktop-a, čvorovi su obično virtualne mašine ili kontejneri, pa će pristup preko IP adrese i port-a biti preusmjeren na odgovarajući servis u klasteru zbog načina na koji funkcioniра Docker Desktop (automatski reguliše). Ovo ne bi bio moguć odgovor za prethodno postavljeno pitanje da se klaster ne izvršava na kućnom računaru nego npr. u oblaku (Azure, AWS itd.).

Konkretno pri korištenju ovakvog tipa klastera, *NodePort* se smatra boljim rješenjem jer je stalno, dok je prosljeđivanje porta privremeno koje je vezano za životni ciklus razmatranog poda. Ipak, konkretno za ArgoCD je korišteno prosljeđivanje porta. To se može obaviti preko komandne linije, ali alat Lens nudi pogodniji način klikom na odgovarajući pod te izborom opcije prikazanoj na slici 4.3.

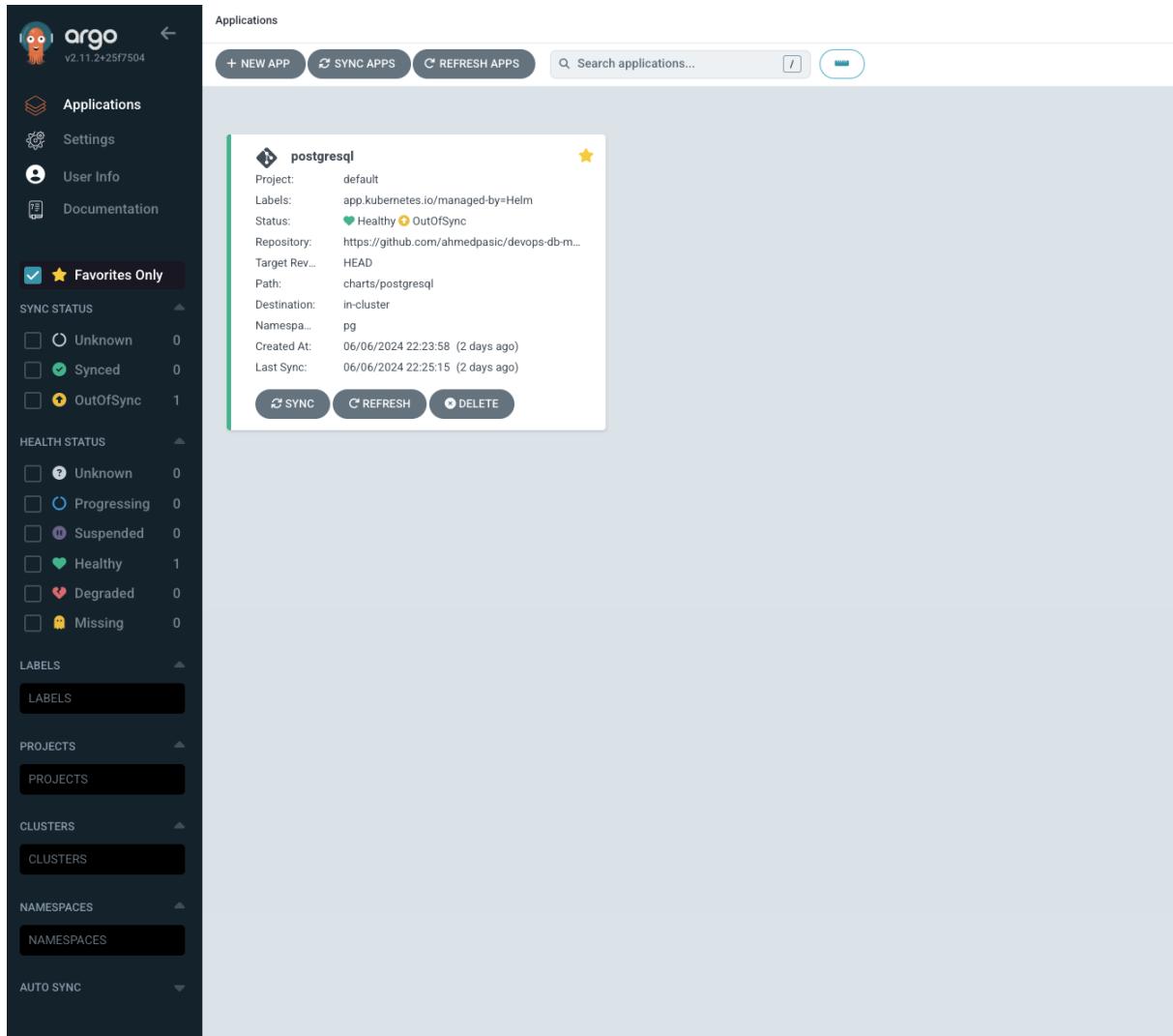
Primjer korištenja kontejnerizacijskih i orkestracijskih tehnika za upravljanje DB serverom



Slika 4.3: Prosljeđivanje porta u Lens alatu

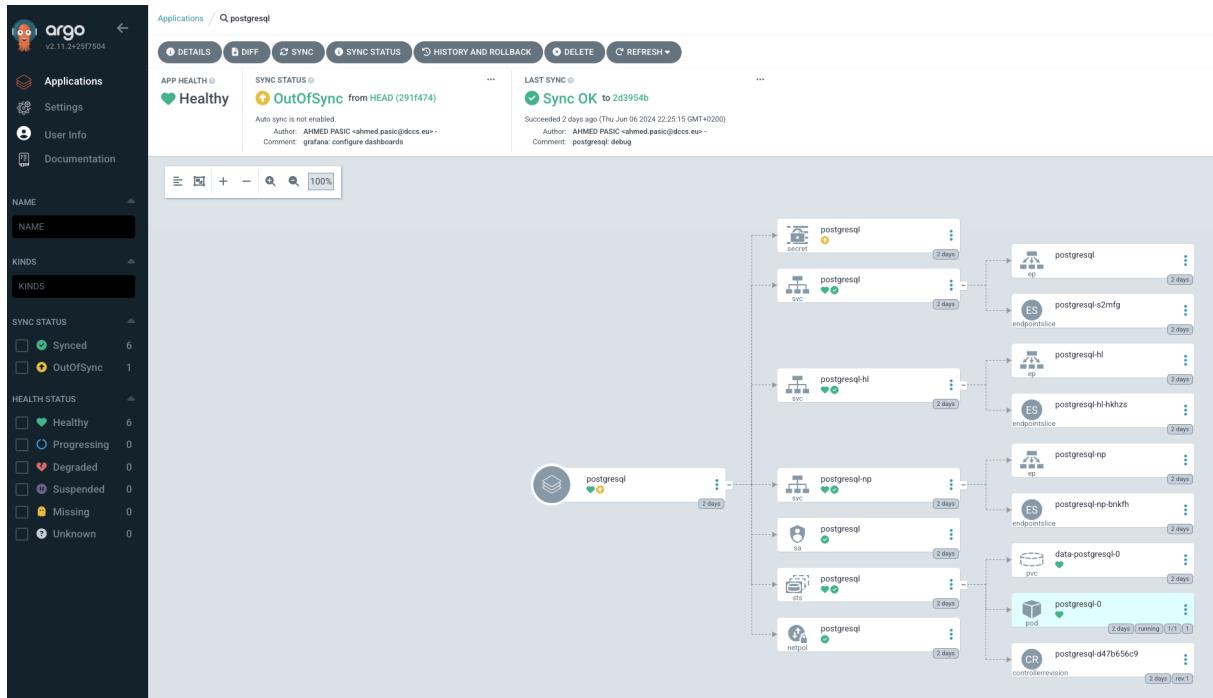
Naredni korak je dodavanje aplikacije koju je potrebno postaviti u upotrebu u ArgoCD. To je moguće preko korisničkog interfejsa. Međutim, ukoliko se prate DevOps principi, koji govore da treba optimizovati i automatizirati što je god moguće više procesa, bolje rješenje je da se konfiguracija svih aplikacija čuva u obliku koda. To je i moguće preko konfiguracionih YAML datoteka čiji je primjer već naveden pod 4.2. Navedeni primjer je upravo konfiguracija PostgreSQL aplikacije unutar ArgoCD-a. Tu konfiguraciju je potrebno primijeniti unutar ArgoCD Helm izdajanja, te ukoliko se otvorи prozor preko web preglednika nakon urađenog prosljeđivanja porta, dobije se prikaz kao na slici 4.4.

Primjer korištenja kontejnerizacijskih i orkestracijskih tehnika za upravljanje DB serverom



Slika 4.4: ArgoCD alat

Ukoliko u postavkama nije uključena automatska sinhronizacija, klikom na dugme za istu ArgoCD kreira sve potrebne Kubernetes resurse na osnovu svega specificiranog u Helm paketu. Ulaskom u karticu za prikazani *postgresql* vide se svi resursi kojima upravlja sam ArgoCD, slika 4.5.



Slika 4.5: Izgled prozora za PostgreSQL u ArgoCD-u

Ukoliko bi se promijenila verzija Helm paketa ili ukoliko bi se promijenile bilo koje vrijednosti na specificiranim putanjama, ArgoCD bi automatski mogao, odmah nakon izvršenog "guranja" koda na repozitorij, ažurirati resurse u skladu sa novim vrijednostima.

4.2.3 PostgreSQL

Uspostavljanjem alata kao što je ArgoCD omogućeno je postavljanje svih potrebnih aplikacija u upotrebu. Do sada se moglo vidjeti kako se to može uraditi za PostgreSQL. Detaljnije, proces konfigurisanja PostgreSQL-a se sastoji od par koraka:

1. Preuzimanje Helm paketa za PostgreSQL
2. Prolazak kroz paket i prilagođavanje podrazumijevanih vrijednosti
3. Dodavanje PostgreSQL u ArgoCD
4. Sinhronizacija

Preuzimanje Helm paketa za PostgreSQL

Pošto je u pitanju DBMS otvorenog koda, podrška od vlasnika za izradu Helm paketa koji bi sadržavao sve potrebne funkcionalnosti i postavke ne postoji. Međutim, u ovom primjeru se koristi paket napravljen od strane kompanije Bitnami. Bitnami je kompanija koja pruža unaprijed konfiguirane aplikacijske pakete i kontejnere za različite softverske aplikacije. Jedan od njihovih proizvoda su Helm paketi za Kubernetes, koji omogućavaju lako instaliranje i upravljanje aplikacija u Kubernetes klasterima. Bitnami Helm paketi su popularni zbog njihove jednostavnosti, pouzdanosti i dobre dokumentacije. Kada je u pitanju PostgreSQL, Bitnami pruža Helm paket koji je dobro testiran i održavan. Ovaj paket uključuje konfiguracijske datoteke, parametre i resurse potrebne za pokretanje PostgreSQL instance u Kubernetes klasteru.

Bitnami PostgreSQL paket često dolazi s dodatnim funkcionalnostima kao što su visoka dostupnost, automatsko stvaranje kopije i opcije za konfiguraciju skladišta. Paket posjeduje aktivno održavanje, dobru dokumentaciju, široku upotrebu i podršku. Paket je preuzet komandom *helm pull* i sačuvan u repozitoriju.

Prolazak kroz paket i prilagođavanje podrazumijevanih vrijednosti

U samom paketu postoje razne postavke koje se mogu podešavati radi što veće mogućnosti prilagođavanja za potrebe korisnika. Jedna od ključnih mogućnosti je podešavanje PostgreSQL korisnika i baza podataka. Moguće je odrediti korisničko ime i lozinku za PostgreSQL superuser-a, kao i ime inicijalne baze podataka koja će biti kreirana prilikom pokretanja. Dodatno, mogu se definisati napredne postavke za performanse, kao što su dijeljeni baferi i radna memorija ("shared_buffers" i "work_mem"), što omogućava optimizaciju baze podataka za specifična radna opterećenja.

Fleksibilnost u upravljanju resursima je još jedna bitna karakteristika. Moguće je precizno definisati minimalne i maksimalne resurse (CPU i memorija) koji će biti dodeljeni PostgreSQL instanci, čime se osigurava optimalna upotreba resursa i stabilnost aplikacije.

Postojanost podataka je osigurana kroz mogućnosti konfiguracije trajnih podataka pomoću PV i PVC. Može se omogućiti ili onemogućiti postojanost podataka, odabratи klasa za pohranu, te definisati veličina i pristupni modovi za PV. Ova postavka je ključna za osiguravanje da podaci ostanu sačuvani čak i ako se pod restartuje ili premjesti. Što se tiče mrežnih postavki, Bitnami PostgreSQL Helm paket omogućava konfiguraciju tipa servisa (ClusterIP, NodePort, LoadBalancer) i mrežnih politika, čime se olakšava integracija s drugim servisima i osigurava sigurnost mrežnog saobraćaja unutar klastera.

Nadzor i metrički podaci su također važan aspekt. Paket podržava integraciju s Prometheusom, omogućavajući prikupljanje detaljnih metričkih podataka o performansama PostgreSQL instance. Ove informacije su ključne za proaktivno upravljanje i optimizaciju baze podataka.

U ovom primjeru, pošto je u pitanju jednostavan način primjene PostgreSQL-a, nisu mijenjane podrazumijevane postavke osim navođenja konfiguracije za kreiranje NodePort servisa radi mogućnosti trajnog pristupa serveru sa stvarnog fizičkog uređaja.

Podrazumijevane postavke podrazumijevaju također i trajnu pohranu podataka, što znači da podaci koji su sačuvani u bazi neće nestati prilikom prestanka rada poda ili pri brisanju istog, dakle, podaci neće biti izgubljeni.

Dodavanje PostgreSQL u ArgoCD i sinhronizacija

Nakon podešavanja same konfiguracije, potrebno je dodati PostgreSQL kao aplikaciju u ArgoCD. Ovaj proces je prethodno već opisan. Nakon sinhronizacije, PostgreSQL pod sa kontejnerom u kojem je pokrenut DBMS se nalazi unutar klastera i može se koristiti. Na slici 4.6 se mogu vidjeti osnovne informacije o pokrenutom kontejneru poput korištene Docker slike, promjenljivih okruženja, port na kojem kontejner radi itd. Te informacije su dostupne klikom na sami pod.

The screenshot shows the Grafana interface for monitoring a PostgreSQL pod named "postgresql-0". The top navigation bar indicates the pod name. Below it, there's a "Containers" section with a single entry for "postgresql". A "CPU" tab is selected, while "Memory" and "Filesystem" tabs are available. A message states "Metrics not available at the moment". The main content area displays the following information:

Category	Value
Status	running, ready
Last Status	terminated Reason:Completed - exit code: 0 Started at: 2024-06-06T22:25:40+02:00 Finished at: 2024-06-07T00:00:17+02:00
Image	docker.io/bitnami/postgresql:16.3.0-debian-12-r8
Ports	tcp-postgresql: 5432/TCP Forward...
Environment	BITNAMI_DEBUG : false PGDATA : /bitnami/postgresql/data POSTGRESQL_CLIENT_MIN_MESSAGES : error POSTGRESQL_ENABLE_LDAP : no POSTGRESQL_ENABLE_TLS : no POSTGRESQL_LOG_CONNECTIONS : false POSTGRESQL_LOG_DISCONNECTIONS : false POSTGRESQL_LOG_HOSTNAME : false POSTGRESQL_PGAUDIT_LOG_CATALOG : off POSTGRESQL_PORT_NUMBER : 5432 POSTGRESQL_SHARED_PRELOAD_LIBRARIES : pgaudit POSTGRESQL_VOLUME_DIR : /bitnami/postgresql POSTGRES_PASSWORD : secret(postgresql)[postgres-password] (edit)
Mounts	/tmp from empty-dir (rw) /opt/bitnami/postgresql/conf from empty-dir (rw) /opt/bitnami/postgresql/tmp from empty-dir (rw) /dev/shm from dshm (rw) /bitnami/postgresql from data (rw)

Slika 4.6: Informacije o PostgreSQL kontejneru

4.3 Grafana

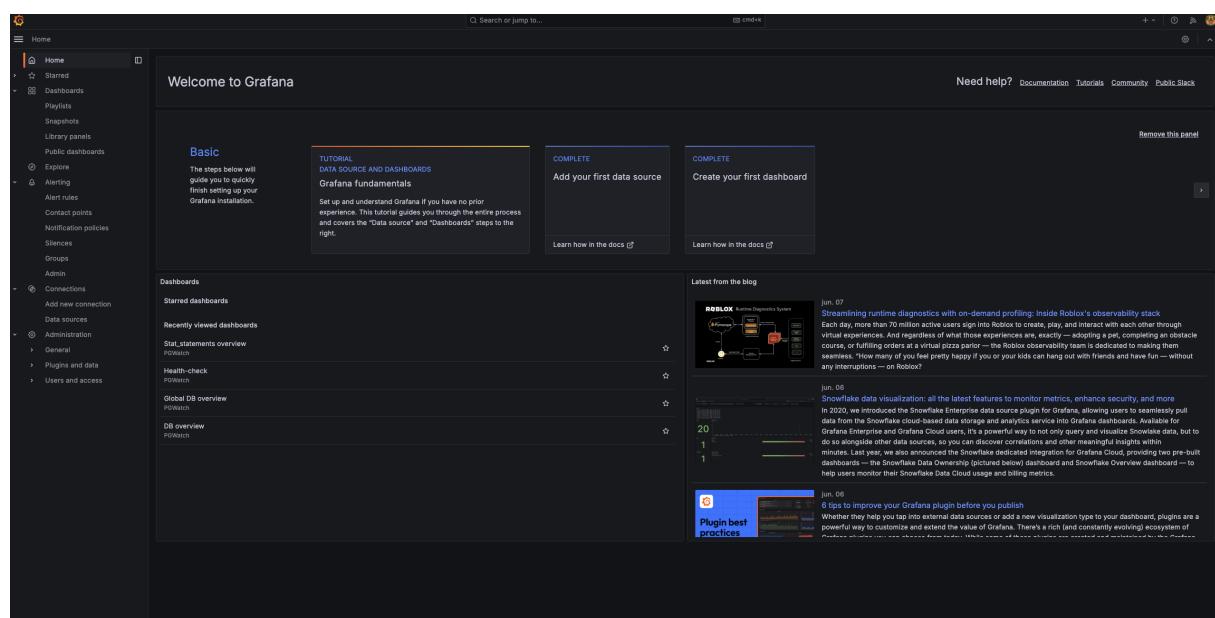
PostgreSQL je sada dostupan unutar klastera te je omogućeno spajanje na server spajanjem na `localhost:30000`. Kredencijali za pristup se mogu pronaći u Kubernetes tajnama unutar klastera

te su automatski generisani ukoliko korisnik ne specificira vrijednosti za kredencijale. Također je omogućena i komunikacija sa ostatkom klastera, tj. sa ostalim aplikacijama unutar drugih podova. U samom klasteru, server baza podataka je dostupan na domeni `postgresql`. `postgres.svc.cluster.local:5432`. Aplikacija u klasteru koja bi željela koristiti ovaj server za svoju bazu podataka bi unutar konekcijskog stringa navela prethodnu domenu.

Sada, za svaku bazu podataka koja bi bila korištena u proizvodnjom okruženju preporučljivo je imati neki vid nadziranja rada iste. Kubernetes nudi različite opcije. Najpopularniji alat za nadziranje je **Grafana**.

Grafana je platforma otvorenog koda za nadzor i vizualizaciju metrika, logova i drugih podataka. Omogućava korisnicima da kreiraju prilagođene komandne table (eng. "dashboards") za vizualizaciju podataka prikupljenih iz različitih izvora kao što su Prometheus, InfluxDB, Graphite, Elasticsearch i mnogi drugi. Grafana podržava širok spektar za grafičko predstavljanje podataka, uključujući linijske grafove, barove, kao i alarme (eng. "Alert rules") koji omogućavaju korisnicima da budu obaviješteni o važnim promjenama u njihovim podacima. Sa svojim intuitivnim korisničkim interfejsom i mnogobrojnim funkcionalnostima, Grafana je popularan izbor za nadzor i analizu u IT industriji.

Za postavljanje Grafane u upotrebu može se koristiti isti pristup kao za DBMS. Dakle, potrebno je preuzeti Helm paket, prilagoditi podrazumijevane vrijednosti konkretnim potrebama, dodati Grafanu kao aplikaciju u ArgoCD te izvršiti sinhronizaciju. Helm paket se može preuzeti sa oficijalnog git repozitorija. Prvobitno, bez prilagođavanja vrijednosti, i izvršavanjem potrebnih koraka u ArgoCD, dobije se pod unutar klastera koji pokreće Grafana kontejner. Izvršavanjem prosljeđivanja porta na isti način kao za ArgoCD, dobije se Grafanin grafički korisnički interfejs prikazan na slici 4.7.



Slika 4.7: Početni prozor alata Grafana

Desni meni prikazuje osnovne komponente ovog alata kao što su:

1. Table sa grafovima (eng. "Dashboards")
2. Alarmiranje (eng. "Alerting")

3. Konekcije (eng. "Connections")

4. Istraživanje (eng. "Explore")

Table sa grafovima su značajna komponenta koja omogućava grafičko predstavljanje podataka i analizu istih.

Alarmsiranje je opcija kojom se može podesiti određena akcija kada neki upit nad resursom dosegne neku neočekivanu vrijednost (npr. potrošnja memorije prešla prag tolerancije). Najčešća akcija koja se podešava jeste slanje elektronske pošte odgovornim osobama.

Konekcije omogućavaju spajanje na izvore podataka (eng. "Data sources"). Izvori podataka su komponente iz kojih Grafana uzima podatke koje vizualizira i nadgleda. Najpoznatiji izvori su Prometheus, Elasticsearch, PostgreSQL, InfluxDB itd.

Istraživanje je komponenta u kojoj se mogu izvršavati različiti upiti nad definiranim izvorima podataka.

U ovom primjeru, bit će demonstrirano kreiranje izvora podataka te njegovo korištenje za definiranje grafova unutar table sa grafovima. Za kreiranje grafova za DBMS moguće su različite opcije: Prometheus (sistem za nadziranje cijelog klastera i prikupljanje metrika sa podova u klasteru), direktno sa PostgreSQL servera (tabele vraćene iz SQL upita se grafički predstavljaju) ili alat kao što je PGWatch2 koji je upravo i izabran te opisan u nastavku.

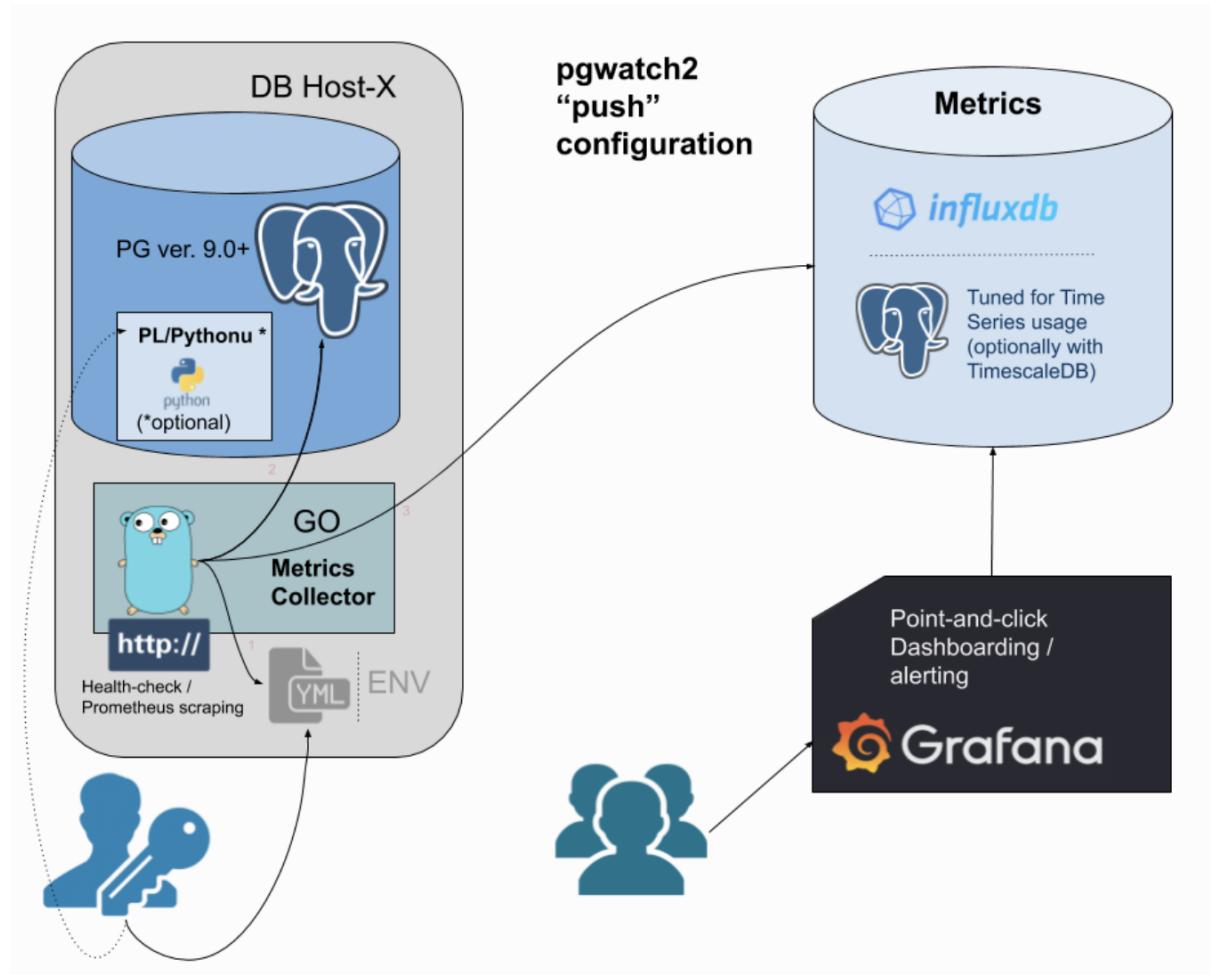
4.4 PGWatch2 i InfluxDB

Kreiranje grafova pomoću direktnih upita na DBMS može biti mukotrpan posao. U tom slučaju je potrebno definirati sve SQL upite, pravove za vraćene vrijednosti, izabrati način prikaza grafa itd. Dodatno, mnoge metrike nisu direktno dostupne te se moraju izvoditi što može dovesti do jako komplikovanih upita nad bazom. Uz to, ukoliko imamo više baza podataka koje želimo nadzirati potrebno je za svaku pojedinačno kreirati izvor podataka u Grafani. Te probleme uspješno rješava alat pod nazivom PGWatch2.

PGWatch2 je napredni alat otvorenog koda za nadziranje performansi PostgreSQL baza podataka, koji je dizajniran za prikupljanje, pohranu i vizualizaciju širokog spektra metrika iz PostgreSQL instanci. Omogućava centralizirano praćenje više PostgreSQL instanci s jednog mjesta. Prikuplja raznovrsne metrike uključujući performanse upita, korištenje resursa, veličinu baza podataka, zaključavanja, statistike replikacije i još mnogo toga. Ovo pruža sveobuhvatan uvid u zdravlje i performanse PostgreSQL instanci. Također je moguće i proširivati skup metrika i prilagođavanje postojećih. Dostupne su i predefinirane table sa grafovima za nadziranje baza podataka.

Koncept rada je sljedeći - PGWatch2 funkcioniše kao centralizovani sistem za prikupljanje i nadzor metrika iz PostgreSQL baza podataka. Prikupljači metrika, koji su skripte sa specifičnim SQL upitim, prikupljaju podatke iz PostgreSQL instanci. Ovi podaci se pohranjuju u bazu podataka koja je konfigurirana kao backend za PGWatch2, kao što su PostgreSQL, InfluxDB, TimescaleDB ili Prometheus. Grafana, alat za vizualizaciju, prikazuje prikupljene metrike putem unaprijed konfigurisanih kontrolnih tabli. PGWatch2 omogućava jednostavnu konfiguraciju putem web korisničkog interfejsa ili konfiguracijskih datoteka, gdje korisnici mogu dodavati nove PostgreSQL instance, postavljati intervale prikupljanja metrika i definirati koje metrike žele prikupljati. Podaci se prikupljaju prema definisanom rasporedu i pohranjuju u bazu za pohranu metrika, nakon čega Grafana kontrolne table prikazuju vizualizacije prikupljenih metrika. Korisnici mogu postaviti pravila za generiranje upozorenja, koja pomažu u brzom reagiranju na

potencijalne probleme, čime PGWatch2 omogućava efikasno i centralizirano nadziranje performansi PostgreSQL baza podataka, pružajući sveobuhvatan uvid u stanje sistema. Na slici 4.8 je grafički prikaz arhitekture "push" konfiguracije ovog alata (slika preuzeta iz zvanične dokumentacije alata).



Slika 4.8: Arhitektura programa pgwatch2 [5]

Proces postavljanja ovog alata u upotrebu je isti kao i za prethodne dvije komponente. Jedina razlika je u konfiguraciji Helm vrijednosti. Ovdje je potrebno kreirati Kubernetes tajne koje će sadržavati neophodne pristupne podatke da se PGWatch2 može autenticirati pri prikupljanju podataka iz PostgreSQL baze. Navodi se konekcioni string, korisničko ime i lozinka. Još jedan neophodan korak pri podešavanju alata jeste kreiranje korisnika u bazi podataka kojim će PGWatch2 izvršavati upite i prikupljati podatke.

Tajne informacije kao što su lozinke nije pametno čuvati samo unutar Kubernetes klastera. U slučaju krahiranja istog, tajne nestaju. Najbolja praksa je tajne informacije čuvati u nekom sigurnom skladištu za senzitivne podatke kao što su to servisi dostupni u oblaku (kao npr Azure key vault u Azure-ovom oblaku) te vršiti sinhronizaciju tih podataka u klaster. U ovom slučaju, pošto takva opcija nije moguća jer je riječ o lokalnom klasteru, i za potrebe ovog jednostavnog primjera, tajne su čuvane samo unutar klastera.

Nakon autentikacije, PGWatch2 prikuplja podatke i metrike iz baze podataka i šalje u svoje

konfigurisano skladište, koje će biti ujedno i izvor podataka za Grafanu pri vizualizaciji istih. Kao skladište prikupljenih podataka korištena je još jedna baza podataka pod nazivom *InfluxDB*.

InfluxDB je visoko-performansna vremenska serijska baza podataka otvorenog koda dizajnirana za brzu pohranu, upite i analizu vremenskih serija podataka kao što su metrički podaci, događaji i analitički podaci u realnom vremenu. Razvijena od strane InfluxData, InfluxDB je optimizirana za velike brzine pisanja i čitanja, omogućavajući pohranu miliona tačaka podataka po sekundi. To je posebno korisno za aplikacije koje zahtijevaju praćenje i nadziranje performansi, kao što su IoT uređaji, aplikacije za nadziranje infrastrukture, te aplikacije za analizu podataka u stvarnom vremenu.

InfluxDB koristi vremenski optimiziran jezik za upite pod nazivom InfluxQL, koji je sličan SQL-u, što olakšava korisnicima da pišu upite za analizu podataka. Baza podataka podržava kompleksne operacije agregacije, selekcije i grupiranja podataka po vremenskim intervalima. Također, InfluxDB omogućava horizontalnu skalabilnost putem klasteringa, što znači da se može proširiti kako bi podržala velike količine podataka i korisnika.

Nakon specificiranja potrebne konfiguracije, neophodno je konfigurisati i Grafanu da je u stanju pročitati podatke iz InfluxDB baze. To se može uraditi kroz konfiguracijske YAML datoteke, na način prikazan u bloku koda 4.4:

Program 4.4: Konfiguracija InfluxDB izvora podataka kroz kod

```
1  datasources:
2    datasources.yaml:
3      apiVersion: 1
4      datasources:
5        - name: InfluxDB
6          type: influxdb
7          uid: influxdb
8          database: ${INFLUXDB_CONNECT_DBNAME}
9          url: http://$${INFLUXDB_CONNECT_ADDRESS}:${
10            INFLUXDB_CONNECT_PORT}
11          user: ${INFLUXDB_CONNECT_DBUSER}
12          secureJsonData:
13            password: ${INFLUXDB_CONNECT_PASSWORD}
14          jsonData:
15            httpMode: GET
```

Navedene varijable okruženja su kodom 4.5 pročitane iz Kubernetes tajni koje su kreirane ručno:

Program 4.5: Konfiguracija InfluxDB varijabli okruženja čitajući tajne

```
1 envValueFrom:
2   INFLUXDB_CONNECT_DBNAME:
3     secretKeyRef:
4       name: influxdb-auth
5       key: influxdb-db
6   INFLUXDB_CONNECT_DBUSER:
7     secretKeyRef:
8       name: influxdb-auth
9       key: influxdb-user
10  INFLUXDB_CONNECT_PASSWORD:
11    secretKeyRef:
12      name: influxdb-auth
13      key: influxdb-password
```

```

14 INFLUXDB_CONNECT_ADDRESS:
15   secretKeyRef:
16     name: pgwatch2-secret
17     key: pgwatch2-ihost
18 INFLUXDB_CONNECT_PORT:
19   secretKeyRef:
20     name: pgwatch2-secret
21     key: pgwatch2-iport

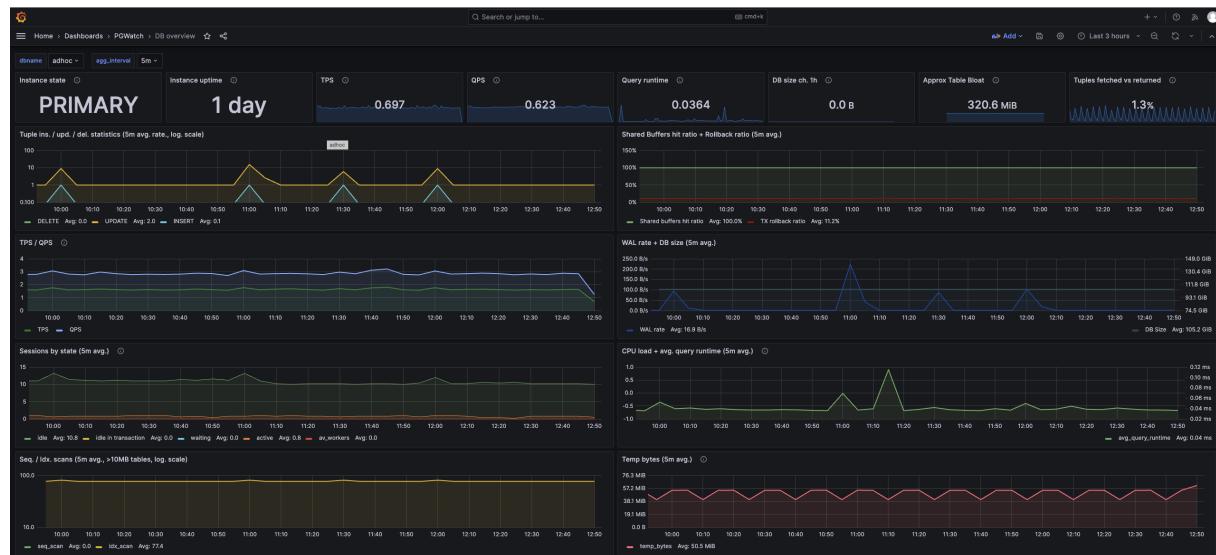
```

Nakon sinhroniziranja Grafane u ArgoCD alatu, pojavit će se izvor podataka kao na slici 4.9.

Slika 4.9: InfluxDB izvor podataka

Sve je spremno za vizualizaciju prikupljenih podataka. PGWatch2 nudi veliki broj predefiniranih tabli sa grafovima u ".json" formatu (**i ovo se može specificirati unutar koda, sve je automatizirano te u slučaju potrebe za ponovnim dizanjem klastera ili kreiranjem novog klastera za drugo okruženje, nije potrebno manualno podešavati konfiguraciju**).

Na slici 4.10 se može vidjeti primjer table sa grafovima koja sadrži veliki broj metrika za praćenje rada razmatrane baze podataka.



Slika 4.10: Predefinirana PGWatch2 tabla sa grafovima

Sa prethodne slike se mogu vidjeti korisne metrike kao što su vrijeme koliko dugo je instanca baze pokrenuta, broj transakcija po sekundi, broj upita po sekundi, dužina trajanja upita, odnos pogodaka keširanih podataka u dijeljenim baferima, odnos transakcija i upita po sekundi i mnoge druge.

4.5 Kubernetes operator

4.5.1 Mane korištenja opisanog pristupa

Prethodno urađeni primjer je dobar pokazatelj kako je Kubernetes, od koncepta namijenjenog prvo bitno za veoma jednostavne aplikacije koje na zahtijevaju čuvanje stanja niti dug životni ciklus instance, postao alat koji ima veliku primjenu i za aplikacije koje zahtijevaju očuvanje stanja i visoku dostupnost, pa čak i za ekstreman slučaj kao što su baze podataka.

Međutim, prethodni pristup ima određene nedostatke. Jedan od tih nedostataka, možda i najveći, jeste komplikovano upravljanje većim brojem replika baze podataka. Neka su specificirane 3 replike. Pošto je u pitanju skup sa stanjem (eng. "Stateful Set"), svaka replika ima zasebno mjesto za pohranu podataka. Osoba koja upravlja bazom mora biti u mogućnosti ispravno posetiti njihovu međusobnu komunikaciju što može stvarati probleme te ujedno narušava principe Kuberntesa kao što je automatizacija. Postoji još drugih nedostataka o kojima će biti riječ u narednom poglavlju.

Problemi koji se javljaju kod prethodnog načina rada sa bazama u Kuberntesu se rješava tzv. *Kubernetes operatorima*.

Opis koncepta

Kubernetes operatori su softverski moduli koji se koriste za automatizaciju upravljanja aplikacijama i servisima u Kubernetes okruženju. Oni implementiraju specifičnu logiku upravljanja za određenu aplikaciju ili servis, omogućavajući Kuberntesu da bolje razumije i upravlja tim aplikacijama. Operatori se obično koriste za složene aplikacije koje zahtijevaju više od osnovne Kuberntes funkcionalnosti, kao što su PostgreSQL baze podataka, Kafka klasteri ili Elasticsearch servisi. Operatori omogućavaju automatizaciju različitih zadataka, kao što su instalacija, konfiguracija, skaliranje, sigurnosna kopija, obnova i nadgledanje aplikacija. Glavna prednost korištenja operatora je u tome što omogućavaju timovima da definiraju specifičnu logiku upravljanja za svoje aplikacije, prilagođavajući je potrebama i zahtjevima aplikacije. To može značajno pojednostaviti upravljanje složenim aplikacijama u Kuberntes okruženju i omogućiti brže i efikasnije upravljanje životnim ciklusom aplikacije.

Konkretno, za rad sa serverima baza podataka operatori služe za olakšavanje automatizacijom i preuzimanjem odgovornosti za svu konfiguraciju koja nastaje korištenjem većeg broja replika u skupu sa stanjem te je glavna prednost operatora za bazu podataka jednostavnost pisanja konfiguracije za istu.

U svrhu poređenja u nastavku rada će ukratko biti prikazana primjena operatora na uspostavljanje istog steka komponenti prikazanih u primjeru skupa sa stanjem.

Konfigurisanje PostgreSQL operatora

Postoje mnogi operatori za PostgreSQL server napravljeni od strane različitih kompanija. Među najpopularnijima je operator kojeg je napravio CrunchyData [33], i koji je izabran u svrhe ove demonstracije. Razlog izbora ovog operatora je jednostavnost upravljanja komponentama, visoka dostupnost, skalabilnost, integracija sa Helmom i samim klasterom i zajednica otvorenog koda.

Proces postavljanja operatora kao i PostgreSQL klastera u upotrebu se sastoji od narednih koraka:

1. preuzimanje Helm paketa za PostgreSQL operator
2. dodavanje konfiguracije za ArgoCD aplikaciju za operator i sinhronizacija
3. preuzimanje Helm paketa za PostgreSQL server
4. prilagođavanje podrazumijevanih Helm vrijednosti
5. dodavanje konfiguracije za ArgoCD aplikaciju za DBMS i sinhronizacija
6. preuzimanje Helm paketa za nadziranje
7. dodavanje konfiguracije za ArgoCD aplikaciju za nadziranje i sinhronizacija
8. testiranje pristupa bazi podataka kao i pregled dobivenih mogućnosti za nadziranje iste

Dakle, prvo je potrebno staviti PostgreSQL operator (PGO) u upotrebu. PGO je Helm paket koji se sastoji od definicija korisnički definiranih resursa. U detalje konfiguracije tih definicija se ovdje neće ulaziti (što zapravo i predstavlja cilj operatora, stvaranje apstrakcije za korisnika i preuzimanje obaveza na operator i automatizacija postavki). Stavljanje PGO-a u upotrebu se radi na isti način kao i sve druge komponente, preko ArgoCD-a.

Nakon toga, može se kreirati PostgreSQL klaster (server sa svojim replikama) kreirajući korisnički definiran resurs na osnovu sada dostupnih definicija. Primjer jednog takvog resursa se može vidjeti u kodu 4.6

Program 4.6: Crunchy Data PostgreSQL klaster manifest datoteka

```
1 apiVersion: postgres-operator.crunchydata.com/v1beta1
2 kind: PostgresCluster
3 metadata:
4   name: my-postgres-cluster
5   namespace: postgres-operator
6 spec:
7   instances:
8     - name: my-postgres-instance1
9       replicas: 2
10  backups:
11    pgbackrest:
12      repos:
13        - name: rep01
14          volume:
15            volumeSource:
16              pvc:
17                spec:
18                  accessModes: [ "ReadWriteOnce" ]
19                  resources:
20                    requests:
21                      storage: 1Gi
22  patroni:
23    dynamicConfiguration:
24      postgresql:
25        parameters:
26          max_connections: "100"
27          shared_buffers: "128MB"
```

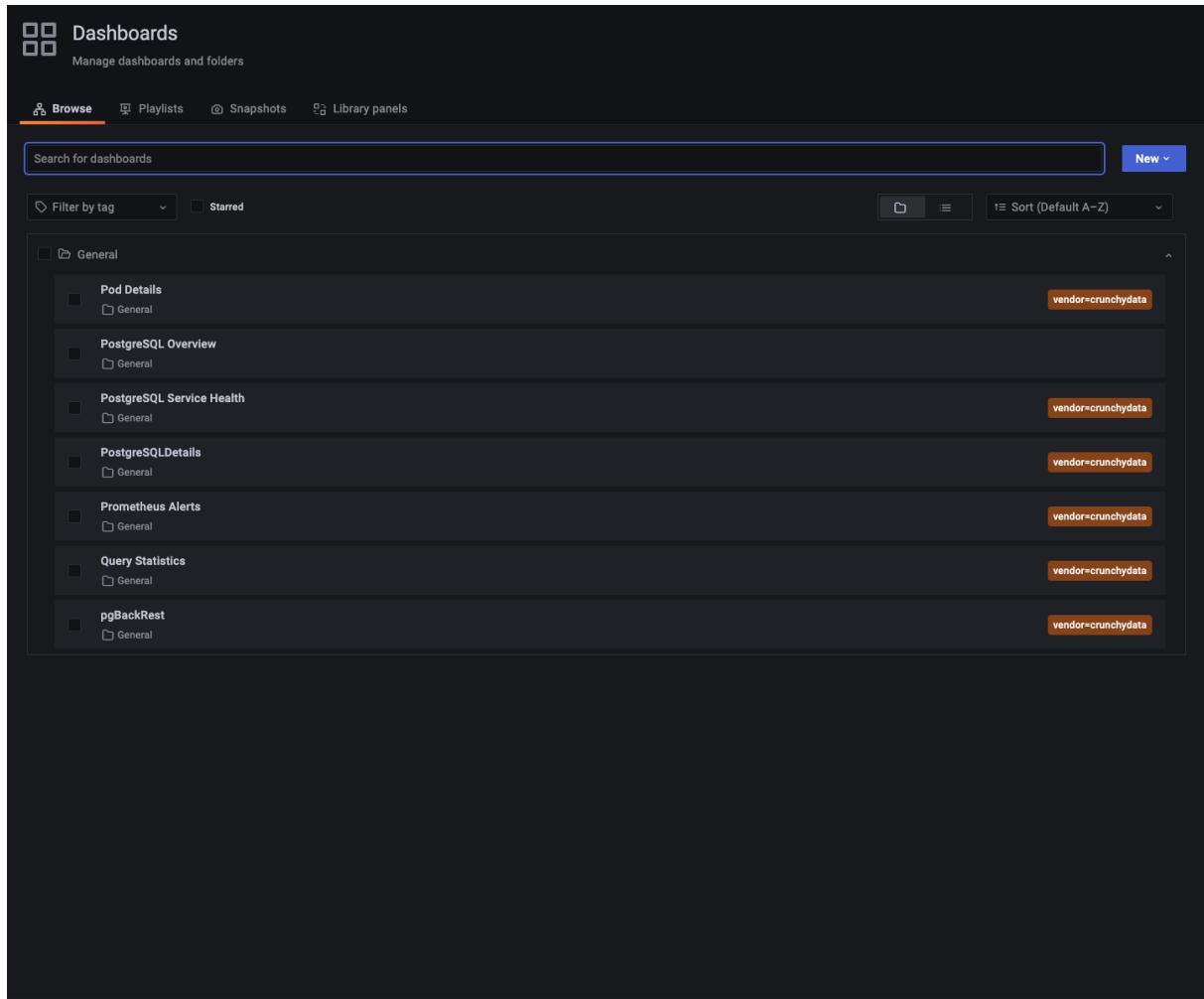
```
28   postgresVersion: 13
29   users:
30     - name: myuser
31       databases:
32         - mydb
33       passwordSecret:
34         name: myuser-secret
35   monitoring:
36     pgmonitor:
37       exporter:
38         image: crunchy-pgmonitor
39         imageTag: v4.5.1
40   ports:
41     - name: postgres
42       port: 5432
43     - name: replication
44       port: 5433
```

Kod 4.6 definira sve potrebne komponente za uspostavu klastera baza podataka. U zaglavlju se nalaze osnovne informacije poput verzije API-ja (*apiVersion*) i vrste resursa (*kind*), te metapodaci (*metadata*) koji uključuju ime i imenski prostor (*namespace*) u klasteru. Specifikacija klastera (*spec*) uključuje definiciju instanci PostgreSQL klastera, gdje se navodi broj replika za svaku instancu. Također je definirana konfiguracija za izradu sigurnosnih kopija pomoću *pgBackRest* alata, gdje je naveden volumen za pohranu sigurnosnih kopija. Dodatno, konfigurirana je *Patroni* dinamička konfiguracija za postavke PostgreSQL-a, uključujući parametre poput maksimalnog broja konekcija i memorije za dijeljene bafere. Verzija PostgreSQL-a je također specificirana, a korisnici i njihove baze podataka su definirani zajedno sa tajnim lozinkama pohranjenim u Kubernetes tajnama (*Secrets*). Monitoring je konfiguriran pomoću *pgMonitor* alata, a navedeni su i portovi koji će se koristiti za PostgreSQL i replikaciju.

Kao što se može vidjeti, sa vrlo malo konfiguracije uspješno je postavljen klaster sa jednom glavnom instancom te sa jednom dodatnom replikom radi redundantnosti i dostupnosti servera baze podataka. Dalje, podešeno je i kreiranje rezervne kopije, konfigurisanje parametara servera, verzije, nadziranja itd.

Da bi se vizualiziralo ponašanje cijelog klastera baza podataka, može se koristiti Grafana. Za te potrebe, od iste kompanije kreiran je i Helm paket za podešavanje Grafane i Prometheusa za nadziranje klastera. Dodavanjem tog paketa kao aplikacije u ArgoCD, Grafana postaje dostupna i za PostgreSQL klaster kreiran operatorom. Dostupne su i sve neophodne metrike u vidu grafova na predefiniranim kontrolnim tablama. Na slici 4.11 vidljiva je lista predefiniranih kontrolnih tabli, a na slikama 4.12 i 4.13 su prikazani primjeri panela koji sadrže korisne informacije pri nadziranju nad bazama u Kubernetes klasteru (konkretno, slika 4.12 prikazuje podatke o potrošnji procesora, radne memorije, diska itd. a slika 4.13 prikazuje izvršene upite od strane izabranog korisnika, njihovo vrijeme trajanja, nad kojom bazom kao i druge korisne informacije).

Primjer korištenja kontejnerizacijskih i orkestracijskih tehnika za upravljanje DB serverom



Slika 4.11: Predefinirane kontrolne table za PGO



Slika 4.12: Paneli za stanje poda instance servera baze podataka



Slika 4.13: Paneli za upite nad određenom bazom

4.6 Rad sa bazom podataka u klasteru

U prethodnim sekcijama je prikazano kako se može konfigurisati i pokrenuti baza podataka u Kubernetes klasteru. U ovoj sekciji bit će prikazano korištenje pokrenute baze podataka, one koja je pokrenuta putem PostgreSQL operatora, kroz jednostavnu aplikaciju.

4.6.1 Pristup bazi podataka preko aplikacije

U nastavku će biti prikazano kako se može pristupiti bazi podataka, koja je u klasteru, preko aplikacije, koja je također postavljena unutar klastera. U pitanju je jednostavna aplikacija za vođenje veterinarske stanice, eng. "petclinic" [34]. U pitanju je javno dostupan primjer otvorenog koda, rađen u Java programskom jeziku te se koristi "Spring" tehnologija za komunikaciju sa bazom.

Kod je javno dostupan te je dozvoljeno njegovo preuzimanje te pokretanje aplikacije. Prije pokretanja, potrebno je konfigurisati bazu podataka koju će koristiti. Podrazumijevano ponašanje je korištenje tzv. "H2" baze podataka. "H2" je brza, laka za korištenje, otvorena Java SQL baza podataka. Ona podržava SQL i JDBC standarde, te nudi napredne funkcije poput podrške za memoriske i diskovne baze, podršku za replikaciju, enkripciju podataka i integraciju sa alatima za razvoj. H2 je popularan izbor za testiranje i razvoj Java aplikacija zbog svoje jednostavnosti i performansi. Funkcionira kao baza podataka u memoriji te čuva podatke samo dok je aplikacija pokrenuta. [35]

Da bi razmatrana aplikacija koristila PostgreSQL kao bazu podataka, potrebno je prilagoditi vrijednosti specificirane u datoteci `spring-petclinic-main/src/main/resources/application.properties` čiji je sadržaj prikazan u bloku koda 4.7.

Program 4.7: Konfiguracijska datoteka za pristup bazi podataka

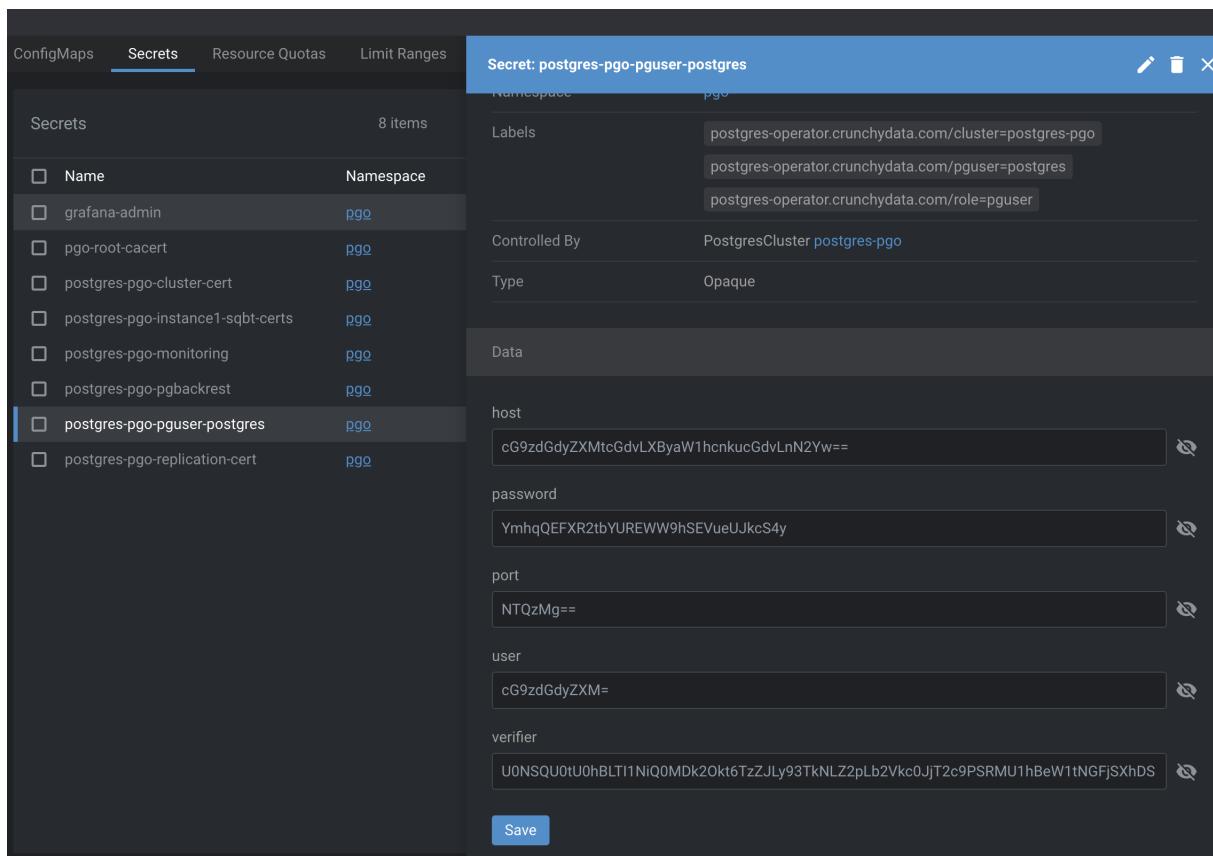
```

1 database=postgres
2 spring.datasource.url=${POSTGRES_URL:jdbc:postgresql://postgres-pgo-
   primary.pgo.svc:5432/petclinic}
3 spring.datasource.username=${POSTGRES_USER:petclinic-user}
4 spring.datasource.password=${POSTGRES_PASS:ahmedpasic}

```

```
s | spring.sql.init.mode=always
```

Linija 1 označava da će korištena baza biti postgres. Druga linija omogućava specificiranje konekcionog stringa. Prisutno domensko ime *postgres-pgo-primary.pgo.svc* je automatski kreirano pri kreiranju servera baza podataka putem PostgreSQL operatora. Sve informacije potrebne za pristupanje samoj bazi su unutar Kubernetes tajne koja je također automatski kreirana pri kreiranju servera. Prikaz Kubernetes tajne je vidljiv na slici 4.14. (Pristupni podaci su senzitivne informacije. Za ovaj primjer nije uzeta u obzir sigurnost istih te su sačuvani u repozitoriju u tekstualnom obliku, što se nikako ne preporučuje u produkcijskim okruženjima)



Slika 4.14: Kubernetes tajna sa pristupnim podacima za bazu

Verzija PostgreSQL servera koja je korištena je 16. Počevši od verzije 14, običnim korisnicima (koji nisu super korisnici - eng. "superusers") nije dozvoljen rad nad javnim šemama unutar baza podataka. Aplikacija koja se ovdje razmatra vrši promjene nad javnom šemom zadane baze podataka. Tako da, prije početka rada aplikacije, potrebno je dati prava pristupa nad njom korisniku kojeg koristi ova aplikacija.

Može se uraditi ručno, povezivanjem na server te izvršavanjem SQL upita prikazanog u kodu 4.9.

Program 4.8: Dodjeljivanje privilegija na šemu

```
GRANT ALL ON SCHEMA public TO "petclinic-user";
```

Drugi način jeste preko Helm vrijednosti unutar koda. U narednom bloku koda prikazan je primjer kako se može dodati novi korisnik u server te ga postaviti kao super korisnika nad određenom bazom. Kod je potrebno ubaciti u *values.yaml*:

Program 4.9: Dodjeljivanje privilegija kroz Helm

```
1 users:
2   - name: petclinic-user
3     databases:
4       - petclinic
5     options: 'SUPERUSER'
```

Aplikacija je sada konfigurisana da koristi bazu kreiranu u sekciji 4.5. Naredni korak jeste postavljanje aplikacije unutar klastera. Prvi korak je kreiranje Docker slike za istu što se postiže korištenjem naredne "Dockerfile" datoteke:

Program 4.10: Dockerfile za petclinic

```
1 FROM eclipse-temurin:21-jdk-jammy
2
3 WORKDIR /app
4
5 COPY .mvn/.mvn
6 COPY mvnw pom.xml .
7 RUN ./mvnw dependency:resolve
8
9 COPY src ./src
10
11 CMD ["./mvnw", "spring-boot:run", "-Dspring-boot.run.profiles=postgres"]
```

Ova Dockerfile datoteka koristi osnovnu sliku `eclipse-temurin:21-jdk-jammy` koja sadrži razvojni komplet za Javu (eng. "Java Development Kit" - JDK) verziju 21 na Ubuntu Jammy osnovi. Radni direktorij postavlja na `/app`, a zatim kopira Maven omotač i konfiguracijske fajlove (`.mvn`, `mvnw`, `pom.xml`) u ovaj direktorij. Maven je alat za upravljanje projektima i automatizaciju gradnje koji upravlja zavisnostima i procesom gradnje Java aplikacija te omogućava lakše upravljanje. Maven omotač (`mvnw`) je skripta koja omogućava pokretanje Maven komandi bez potrebe za instalacijom Mavena na sistemu, jer koristi lokalnu verziju Mavena unutar projekta. Dockerfile pokreće `dependency:resolve` za preuzimanje zavisnosti definisanih u `pom.xml` fajlu. Zatim kopira izvorni kod aplikacije iz lokalnog direktorija `src` u kontejner. Na kraju, postavlja komandnu liniju koja pokreće Maven omotač skriptu sa komandom `spring-boot:run`, koristeći profil `postgres` za konfiguraciju prilikom pokretanja aplikacije unutar Docker kontejnera. Dalje, pokretanjem komande `"docker build . -t petclinic"` dobije se potrebna Docker slika.

Naredni korak je kreiranje neophodnih Kubernetes resursa. Za rad aplikacije potrebna su samo dva, pod i servis. Za postavljanje aplikacije se može koristiti ArgoCD na sličan način.

Nakon postavljanja, aplikacija će se pokušati spojiti na bazu te izvršiti predefinirane upite nad istom. U blokovima koda 4.11 i 4.12 su prikazani isječci iz SQL skripti koje su dostupne na putanji `spring-petclinic-main/src/main/resources/db/postgres`.

Program 4.11: Kreiranje tabela

```
1 CREATE TABLE IF NOT EXISTS vets (
2   id          INT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
3   first_name TEXT,
4   last_name  TEXT
5 );
6 CREATE INDEX ON vets (last_name);
7
8 CREATE TABLE IF NOT EXISTS specialties (
9   id      INT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
10  name   TEXT
```

```

11 );
12 CREATE INDEX ON specialties (name);
13
14 CREATE TABLE IF NOT EXISTS vet_specialties (
15     vet_id          INT NOT NULL REFERENCES vets (id),
16     specialty_id   INT NOT NULL REFERENCES specialties (id),
17     UNIQUE (vet_id, specialty_id)
18 );
19 ...

```

Program 4.12: Popunjavanje tabela

```

1 INSERT INTO vets (first_name, last_name) SELECT 'James', 'Carter'
      WHERE NOT EXISTS (SELECT * FROM vets WHERE id=1);
2 INSERT INTO vets (first_name, last_name) SELECT 'Helen', 'Leary'
      WHERE NOT EXISTS (SELECT * FROM vets WHERE id=2);
3 ...
4 INSERT INTO specialties (name) SELECT 'radiology' WHERE NOT EXISTS (
      SELECT * FROM specialties WHERE name='radiology');
5 ...
6 INSERT INTO vet_specialties VALUES (2, 1) ON CONFLICT (vet_id,
      specialty_id) DO NOTHING;
7 INSERT INTO vet_specialties VALUES (3, 2) ON CONFLICT (vet_id,
      specialty_id) DO NOTHING;
8 ...

```

Novo stanje u bazi se može provjeriti pristupom kroz neki alat za rad sa bazama, te se mogu izvršiti upiti. Na slici 4.15

	id [PK] integer	first_name text	last_name text	address text	city text	telephone text
2	2	Betty	Davis	638 Cardinal Ave.	Sun Prairie	6085551749
3	3	Eduardo	Rodriguez	2693 Commerce St.	McFarland	6085558763
4	4	Harold	Davis	563 Friendly St.	Windsor	6085553198
5	5	Peter	McTavish	2387 S. Fair Way	Madison	6085552765
6	6	Jean	Coleman	105 N. Lake St.	Monona	6085552654
7	7	Jeff	Black	1450 Oak Blvd.	Monona	6085555387
8	8	Maria	Escobito	345 Maple St.	Madison	6085557683
9	9	David	Schroeder	2749 Blackhawk Trail	Madison	6085559435
10	10	Carlos	Estaban	2355 Independence Ln.	Waunakee	6085555487
11	11	Ahmed	Pasic	'Zmaj od Bosne' br. 23	Fojnica	1111111111

Slika 4.15: Pregled novog stanja u bazi izvršavanjem upita

Aplikacija također nudi i grafički interfejs preko kojeg je moguće dodati npr. novog vlasnika kućnog ljubimca što je vidljivo na slici 4.16, a na slici 4.17 se može vidjeti da je novi vlasnik zaista prisutan u tabeli unutar same baze.

The screenshot shows a web application interface for adding a new owner. At the top, there is a navigation bar with links for HOME, FIND OWNERS, VETERINARIANS, and ERROR. Below the navigation bar is a form titled "Owner". The form fields are as follows:

- First Name: Harry
- Last Name: Kane
- Address: "Adresa"
- City: Munchen
- Telephone: 2222222222

At the bottom of the form is a green "Add Owner" button.

spring by VMware Tanzu

Slika 4.16: Dodavanje vlasnika kroz grafički interfejs

The screenshot shows a database query tool interface. At the top, there are tabs for "Query" (which is selected) and "Query History". Below the tabs, the following SQL query is displayed:

```
1 ✓ SELECT * FROM owners
2 WHERE first_name LIKE 'Harry';
```

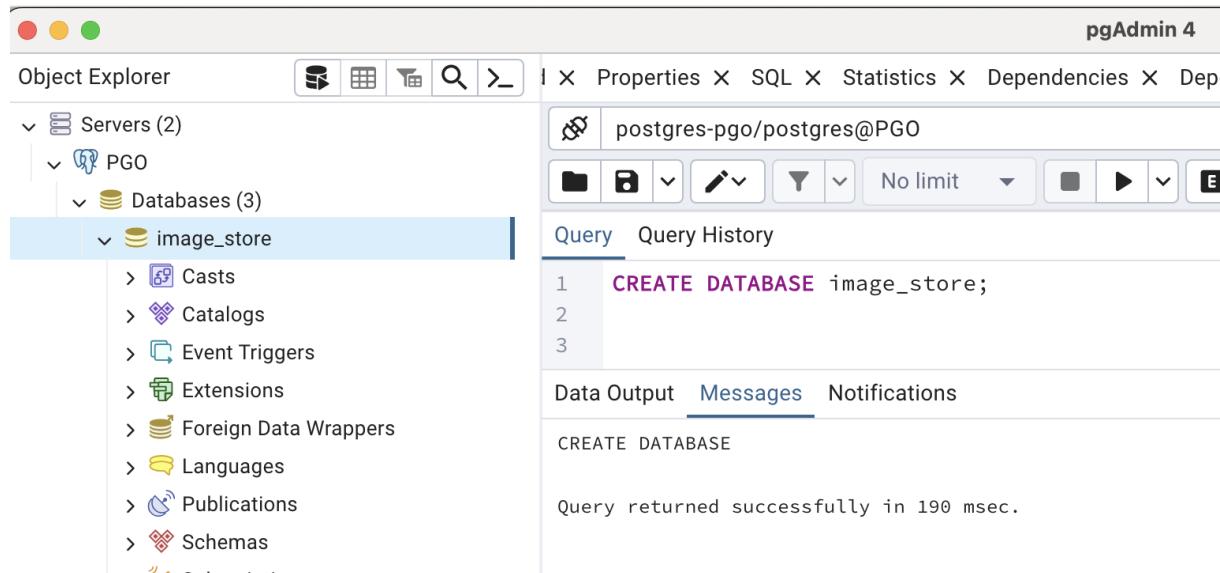
At the bottom of the interface, there are tabs for "Data Output", "Messages", and "Notifications". Below these tabs is a toolbar with various icons for managing data. A table below the toolbar displays the results of the query:

	id [PK] integer	first_name	last_name	address	city	telephone
1	12	Harry	Kane	"Adresa"	Munchen	2222222222

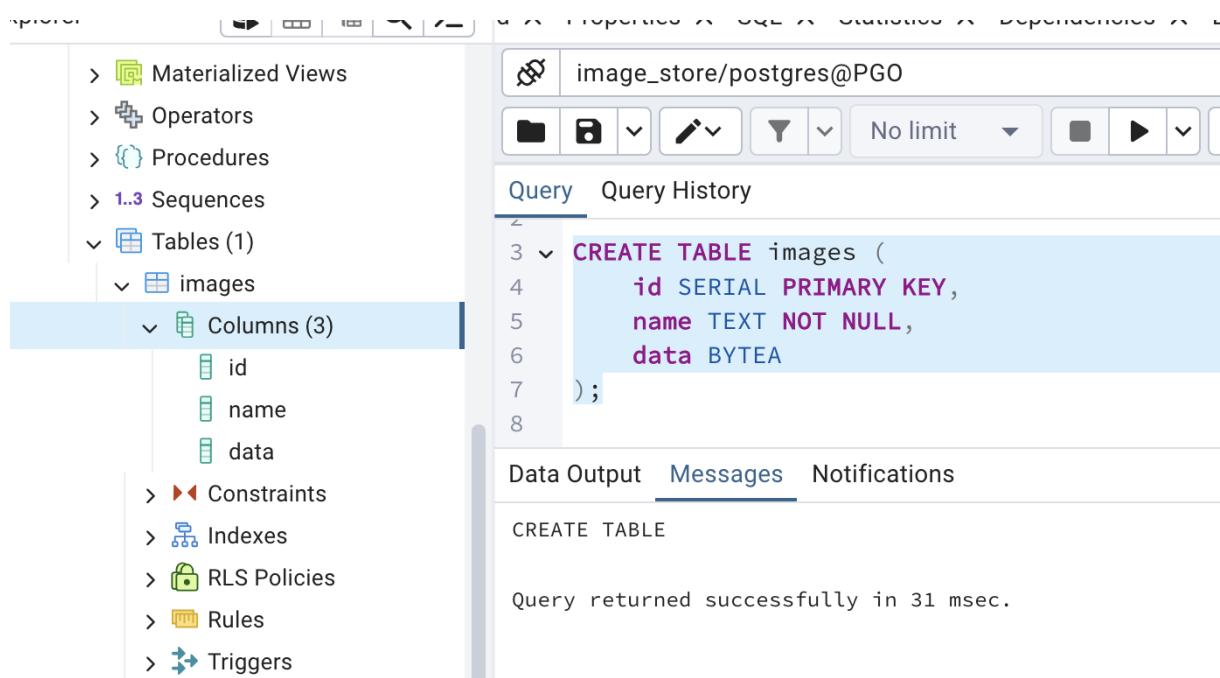
Slika 4.17: Provjera stanja u bazi nakon dodavanja vlasnika

4.6.2 Direktan rad nad bazom kroz pgAdmin4 alat

Naravno, više aplikacija bi moglo koristiti isti server, što bi i dalje uspješno radilo radi dobre mogućnosti skaliranja pri radu PostgreSQL operatora. U nastavku je prikazano kreiranje nove baze, nove tabele, te izvršavanje CRUD operacija nad istom.



Slika 4.18: Kreiranje nove baze podataka



Slika 4.19: Kreiranje tabele za čuvanje slika

Na slici 4.19 je prikazano kreiranje tabele za pohranu slika. Što se tiče dodavanja redova u ovu tabelu, preko aplikacija je vrlo jednostavno jer one čuvaju te podatke direktno u samoj memoriji aplikacije. Stvar je drugačija ako se vrši ručno dodavanje. Pri ručnom dodavanju, nprimjer preko pgAdmin alata, potrebno je specificirati putanju do slike, što bi značilo da se slika treba nalaziti u kontejneru u kojem je pokrenut DBMS. Način da se to postigne je povezivanje

poda sa postojanim volumenom pri čemu se razmatrana slika nalazi u tom volumenu. Još jedan, možda i jednostavniji način, je kopiranje datoteke sa lokalnog uređaja u kontejner na određenu putanju. To se može uraditi pomoću komande "kubectl cp".

Nakon izvršavanja jednog od ova dva načina, datoteka se može dodati, kao što je prikazano na slici 4.20.

```

    pgAdmin 4
    Object Explorer   Statistics  Dependencies  Dependents  Processes  postgres-pgo/post...  image_store/
    Publications
    Schemas (1)
    public
        Aggregates
        Collations
        Domains
        FTS Configurations
        FTS Dictionaries
        FTS Parsers
        FTS Templates
        Foreign Tables
        Functions
        Materialized Views
        Operators
        Procedures
        Sequences
    Tables (1)
        images
            Columns (3)
                id
                name
                data
            Constraints
            Indexes
    Query  Query History
    7 );
8
9 v INSERT INTO images (name, data)
10 VALUES ('example.png', pg_read_binary_file('/tmp/ahmed/harry_kane.png'));
11
12
Data Output  Messages  Notifications
INSERT 0 1
Query returned successfully in 70 msec.
  
```

Slika 4.20: Dodavanje slike u tabelu

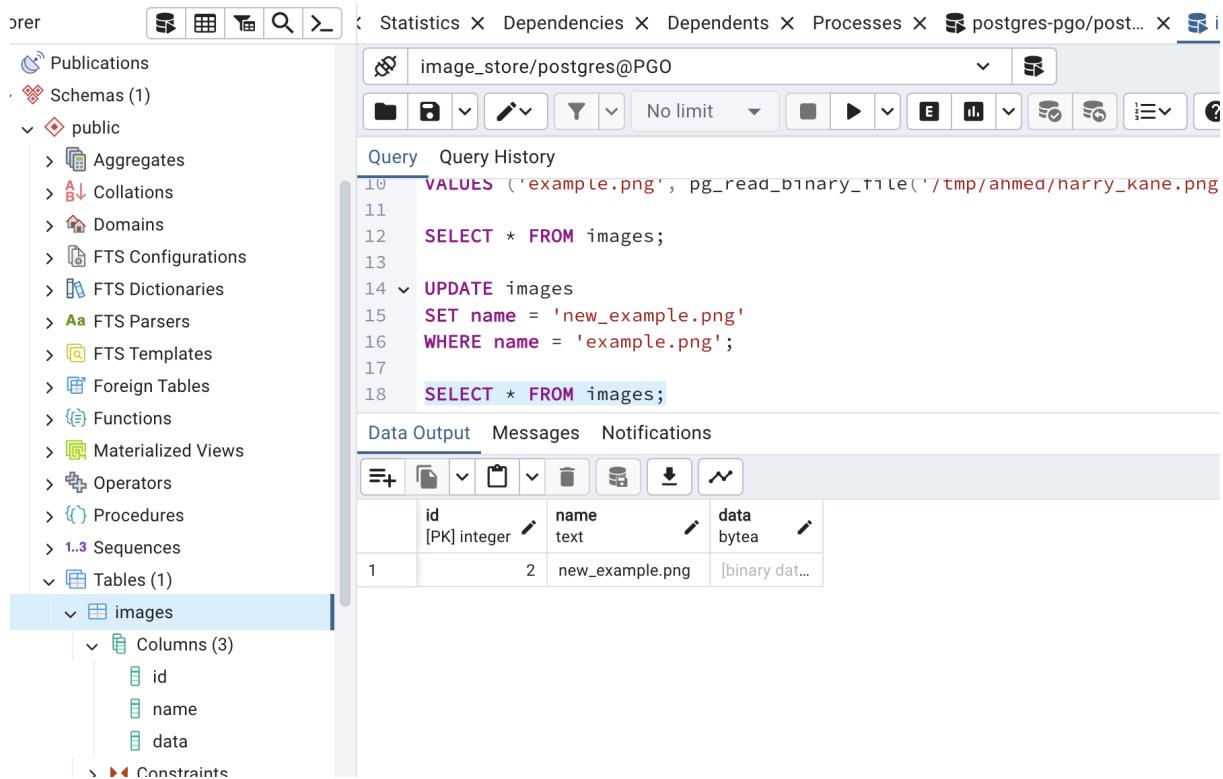
```

    pgAdmin 4
    Publications
    Schemas (1)
    public
        Aggregates
        Collations
        Domains
        FTS Configurations
        FTS Dictionaries
        FTS Parsers
        FTS Templates
        Foreign Tables
        Functions
        Materialized Views
        Operators
        Procedures
        Sequences
    Tables (1)
        images
            Columns (3)
                id
                name
                data
            Constraints
            Indexes
    Query  Query History
    5     name TEXT NOT NULL,
    6     data BYTEA
    7 );
8
9 v INSERT INTO images (name, data)
10 VALUES ('example.png', pg_read_binary_file('/tmp/ahmed/harry_kane.png'));
11
12 SELECT * FROM images;
13
Data Output  Messages  Notifications
  
```

	id [PK] integer	name text	data bytea
1	2	example.png	[binary dat...]

Slika 4.21: Prikaz sadržaja tabele "images"

Primjer korištenja kontejnerizacijskih i orkestracijskih tehnika za upravljanje DB serverom

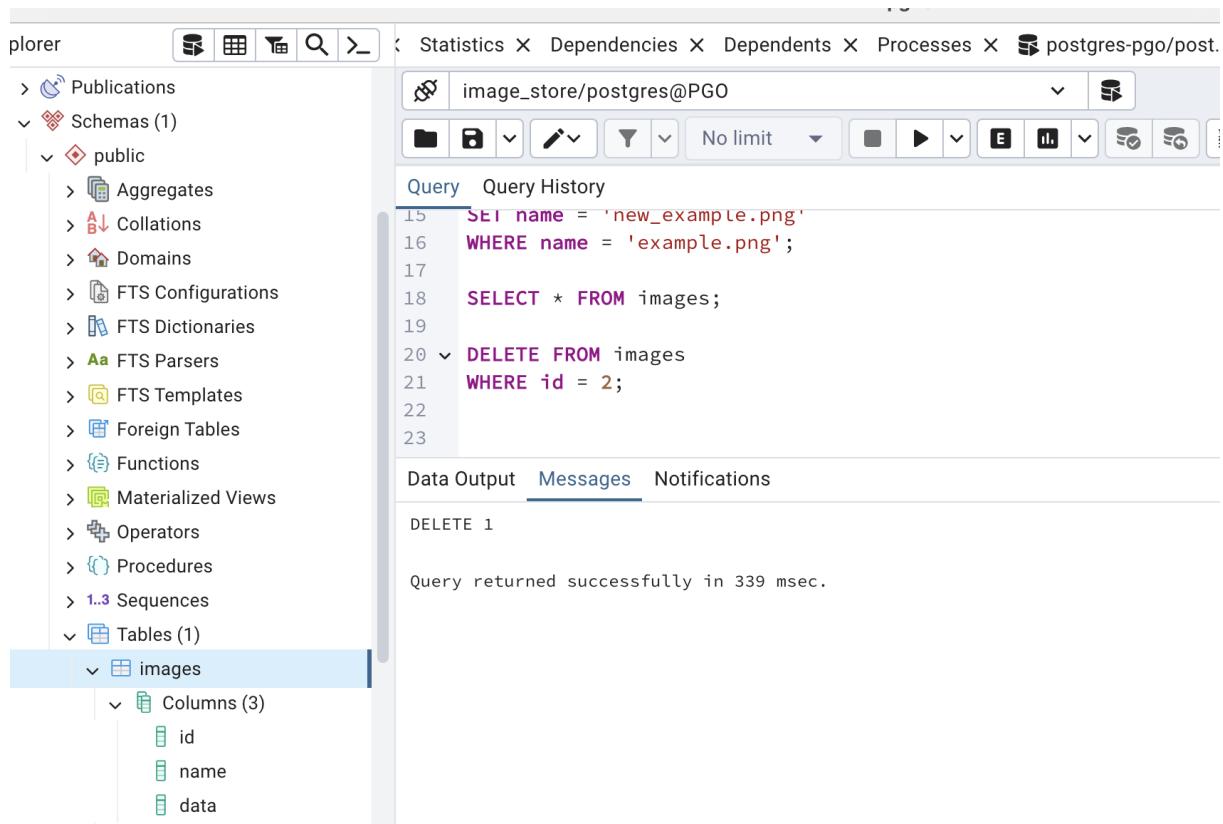


The screenshot shows the pgAdmin 4 interface. On the left, the Object Explorer tree shows the database structure, including the 'public' schema and its 'images' table. The 'images' table has three columns: 'id' (PK integer), 'name' (text), and 'data' (bytea). A context menu is open over the 'name' column of the first row. The main pane displays a SQL query window with the following code:

```
VALUES ('example.png', pg_read_binary_file('/tmp/anmed/harry_kane.png')
SELECT * FROM images;
UPDATE images
SET name = 'new_example.png'
WHERE name = 'example.png';
SELECT * FROM images;
```

The 'Data Output' tab shows the result of the query, which is a single row with id 2, name 'new_example.png', and data [binary data...].

Slika 4.22: Promjena reda u tabeli "images"



The screenshot shows the pgAdmin 4 interface. The Object Explorer tree is identical to the previous screenshot. The main pane displays a SQL query window with the following code:

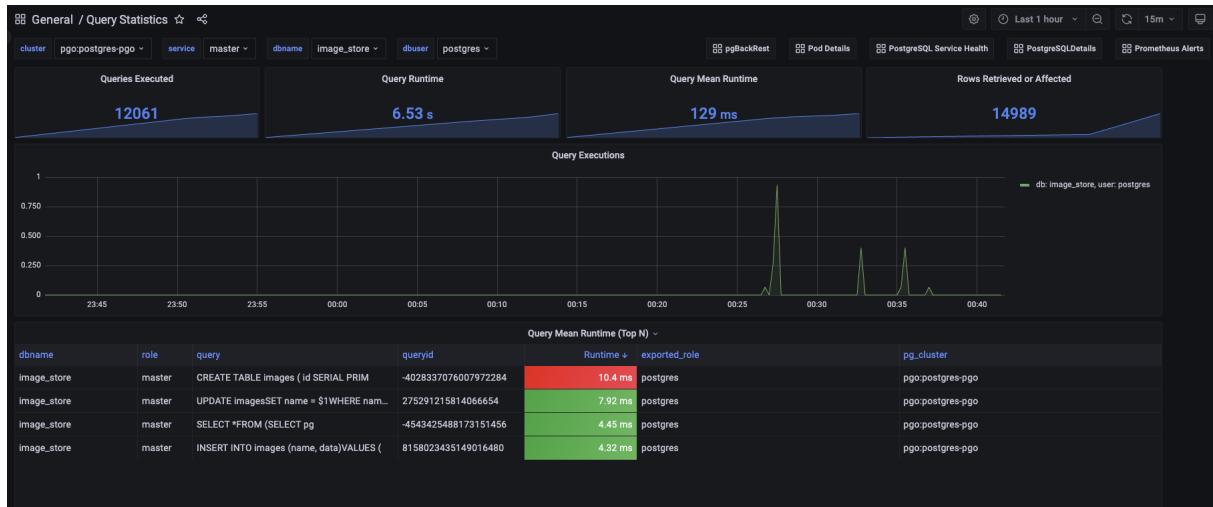
```
SET name = 'new_example.png'
WHERE name = 'example.png';
SELECT * FROM images;
DELETE FROM images
WHERE id = 2;
```

The 'Messages' tab shows the output of the DELETE command: 'DELETE 1'. Below it, a message states 'Query returned successfully in 339 msec.'

Slika 4.23: Brisanje reda iz tabele "images"

Izvršeni upiti su također vidljivi i praćeni unutar Grafane i za novokreiranu bazu. Na slici

4.24 su prikazani svi izvršeni upiti filtrirani po vremenu, bazi i korisniku koji ih je izvršio što može predstavljati jako bitne i korisne uvide pri radu sa stvarnim podacima u producijskim i osjetljivim okruženjima.



Slika 4.24: Izvršeni upiti nad bazom prikazani u Grafana panelu

* * *

Razmatrano poglavlje predstavlja demonstraciju praktične primjene DevOps principa i alata pri upravljanju bazama podataka. Dat je primjer kako se sistem za upravljanje bazama podataka, kao što je PostgreSQL, može podesiti u dinamičnom okruženju koristeći alate čija je primarna namjena bila za efemerne aplikacije i servise koji ne zahtijevaju očuvanje podataka (suprotno od baza podataka). Također, prikazano je i konkretno korištenje baze putem jednostavne aplikacije te i izvršavanje upita direktno nad samom bazom. U narednom poglavlju bit će opisani dobiveni rezultati, otkrivene prednosti i nedostaci u ovom pristupu i bit će upoređena dva navedena načina upravljanja bazama podataka u klasteru.

Poglavlje 5

Analiza postignutih rezultata

Prethodno poglavlje sadrži detaljan primjer kako je moguće upravljati bazama podataka koristeći alate za kontejnerizaciju i orkestraciju. Pri tome je dat uvid u dva načina: korištenjem običnog skupa sa stanjem i korištenjem operatora. U nastavku će biti opisane prednosti i mane oba ova načina.

5.1 Skup sa stanjem vs. Kubernetes operatori za baze podataka

Skupovi sa stanjem pružaju jednostavnost jer su dio Kubernetes jezgra i njihova upotreba ne zahtijeva dodatne komponente. Lahko ih je razumjeti i implementirati jer su dio osnovne Kubernetes funkcionalnosti. Pružaju podršku za PV, što znači da podaci ostaju sačuvani čak i nakon restarta ili premještanja podova. Svaki pod u skupu sa stanjem ima svoj jedinstveni identitet i povezani postojani volumen. Također omogućavaju kontrolirano kreiranje i skaliranje podova, čime se osigurava da se novi podovi kreiraju jedan po jedan i u određenom redoslijedu, što je korisno za baze podataka koje zahtijevaju specifičan redoslijed inicijalizacije ili pristupa. Međutim, skupovi sa stanjem pružaju ograničenu funkcionalnost jer ne nude napredne funkcionalnosti poput automatskog oporavka od grešaka, kreiranja rezervne kopije i vraćanja podataka, ili automatskog skaliranja. Većina operacija, kao što su nadogradnje i kreiranje rezervnih kopija, mora se obavljati ručno, što može biti složeno i podložno greškama, posebno u složenijim okruženjima. Pored toga, skupovi sa stanjem ne pružaju ugrađeni nadzor i metrike, pa je potrebno dodatno konfigurirati alate za nadzor kako bi se pratilo stanje i performanse baza podataka.

Kubernetes operatori automatski upravljaju složenim operacijama poput kreiranja, skaliranja, kreiranja rezervnih kopija, vraćanja podataka, nadogradnji i oporavka od grešaka, što značajno smanjuje ručni rad i rizik od grešaka. Oni su specifični za određene baze podataka i implementiraju najbolje prakse za njihovo upravljanje, što znači da su optimizirani za performanse, sigurnost i pouzdanost određenih baza podataka. Mnogi operatori dolaze sa ugrađenim alatima za nadzor i prikupljanje metrika, što olakšava praćenje stanja i performanse baza podataka. Ipak, operatori mogu biti složeni za razumijevanje i konfiguraciju, posebno za korisnike koji nisu upoznati sa Kubernetes operatorima i konceptima upravljanja životnim ciklusom aplikacija u Kubernetesu. Korištenje operatora često znači zavisnost od specifične implementacije treće strane, što može donijeti rizike u vezi sa podrškom, sigurnosnim ažuriranjima i kompatibilnošću sa budućim verzijama Kuberneta. Također, operatori mogu zahtijevati dodatne resurse za po-

kretanje svojih kontrolera i upravljanje bazama podataka, što može povećati ukupnu potrošnju resursa u klasteru.

Unatoč nabrojanim nedostacima operatora, njihove ogromne prednosti se ne mogu zanemariti te je očigledno da predstavljaju bolju opciju ukoliko se razmatra upravljanje bazama podataka unutar Kubernetes klastera. Bez operatora, zbog komplikacija koje mogu nastati skaliranjem, kreiranjem rezervnih kopija, nadziranjem i sl. Kubernetes ipak ne bi predstavljao ozbiljnu opciju kada je u pitanju rad sa bazama podataka.

5.2 Baza podataka u Kubernetesu?

Odgovor na pitanje da li je isplativo postaviti sistem za upravljanje bazama podataka unutar Kubernetes klastera nije izravan. U poređenju sa analognim servisima u oblaku, Kubernetes način ima određeni prednosti, ali i mane. Kubernetes nema očigledne prednosti u odnosu na oblak - oblak također nudi dosta fleksibilnosti, mnogo olakšano upravljanje gdje korisnik samo specificira parametre, željene resurse i način pristupa bazi. Generalno, upravljanje serverom baza podataka koji se nalazi u oblaku je jednostavniji. Korisniku je potrebno samo znanje o bazama podataka, dok mu ostalo znanje o kontejnerizaciji, orkestraciji i drugim DevOps principa nije potrebno.

Međutim, pristup demonstriran u ovom radu ima i neke svoje prednosti. Prva, možda i najznačajnija prednost, jeste ekonomičnost. U većini slučajeva, postavljanje baze u upotrebu unutar Kubernetes klastera će proizvesti manje troškove od posebnog resursa unutar oblaka (ne mora uвijek biti slučaj). Dalje, portabilnost također može biti od značaja. U situaciji da korisnik želi migrirati podatke iz jednog oblaka u drugi to je mnogo lakše ukoliko je u pitanju baza podataka u klasteru jer se na jednake načine može pokrenuti na bilo kojem klasteru neovisno od oblaka čiji se klaster kao servis koristi.

Dakle, pri odlučivanju da li izabrati ovakav način upravljanja bazama, potrebno je razmotriti prioritete i na osnovu istih donijeti odluku šta je najbolje za taj konkretan slučaj.

5.3 Najbolje prakse za DBMS na Kubernetesu

U nastavku su nabrojane najbolje prakse kojima se treba voditi pri upravljanju producijskim bazama u klasterskom okruženju:

1. Koristiti operatore
2. Imati pokrenutu više od jedne instance servera (bar jedna replika)
3. Koristiti PV
4. Izbjegavati ovaj način upravljanja ako je u pitanju prevelika baza podataka (npr. 50 TB)
5. Za klaster na kojem će biti serveri baza podataka bolje je izabrati nešto veće čvorove
6. Cijeli klaster ne smije biti samo na jednom fizičkom čvoru
7. Podesiti prioritete podova baze tako da imaju najveći prioritet unutar klastera radi što manjeg broja restarta
8. Server što više sakriti, ne izlagati previše vanjskoj sredini radi sigurnosti podataka

9. Omogućiti enkripciju podataka i pri mirovanju i pri kretanju
10. Kredencijale čuvati u Kubernetes tajnama
11. Obavezno raditi kreiranje rezervne kopije
12. Koristiti DBMS koji je pogodan za rad sa Kubernetesom (lahko omogućeno skaliranje i *sharding*)

* * *

Poglavlje je posvećeno analizi rezultata iz praktičnog dijela. Opisane su prednosti i mane oba načina upravljanja bazama podataka u Kubernetesu, zašto koristiti ovakav pristup umjesto tradicionalnih te su navedene i najbolje prakse u tom slučaju.

Zaključak

Ostvareni ciljevi rada/disertacije

U ovom radu istraženi su kontejnerizacijski i orkestracijski pristupi za upravljanje bazama podataka koristeći DevOps alate. Proveden je detaljan pregled stanja u oblasti kroz analiziranje referentnih radova. Kroz rad opisan je DevOps kao pristup razvoju softvera, zajedno sa svojim osnovnim principima kao što su kontejnerizacija i orkestracija. Također, opisani su i glavni alati za te principe, Docker i Kubernetes.

Dato je i poređenje te navođenje razloga žasto ili zašto ne koristiti navedene DevOps tehnike pri radu sa bazama podataka. Kroz praktičnu studiju slučaja prikazana su dva načina upravljanja bazama podataka koristeći kontejnerizaciju i orkestraciju (moguće su i drugi načini a ne samo opisani) te tehnologije i alate popularne u vrijeme pisanja ovog rada. Rezultat praktičnog rada predstavlja cjelokupno programsko rješenje koje implementirane sve bitne komponente za rad jednog sistema za upravljanje bazama podataka.

Na kraju, dato je nekoliko uputa za rad u ovakovom okruženju te kada je isplativo koristiti se ovim načinom rada sa bazom.

Smjernice za budući rad

Da li koristiti ili ne koristiti ovakav način upravljanja za baze podataka zavisi od konkretnе situacije, da li je bitna ekonomičnost, portabilnost itd. Jedan od načina korištenja bi bio za baze podataka koje se koriste u razvojnim okruženjima, a ne u proizvodnim.

U budućnosti, radovi u ovoj oblasti bi se mogli fokusirati na unapređenje rada operatora, omogućiti lakše korištenje, kreiranje novih baza za koje ne postoji itd. Prije nekoliko godina, upravljanje bazom podataka unutar klastera je bilo nezamislivo. Sada, postoje rješenja koja omogućavaju to upravljanje do te mjeru da se može vršiti poređenje da li je bolja opcija imati bazu u klasteru ili u nekom tradicionalnom okruženju.

Literatura

- [1] Atlassian, “Devops”, dostupno na: <https://www.atlassian.com/devops> 2023.
- [2] Dhaduk, H., “What is container orchestration? basics, benefits, tools, and best practices”, dostupno na: <https://www.simform.com/blog/container-orchestration/> Pristupljeno: 2024-06-11. 2024.
- [3] Documentation, K., “Cluster architecture”, dostupno na: <https://kubernetes.io/docs/concepts/architecture/> Pristupljeno: 2024-05-18. 2024.
- [4] Wang, Y.-T., Ma, S.-P., Lai, Y.-J., Liang, Y.-C., “Qualitative and quantitative comparison of spring cloud and kubernetes in migrating from a monolithic to a microservice architecture”, Service Oriented Computing and Applications, Vol. 17, 05 2023, str. 1-11.
- [5] Consulting, C. P., “Pgwatch2 documentation”, dostupno na: <https://pgwatch2.readthedocs.io/en/latest/README.html> Pristupljeno: 2024-06-09. 2020.
- [6] Insights, G. M. (2023) Devops market size by component, service, by deployment model, by organization, application global forecast, 2023 - 2032, dostupno na: <https://www.gminsights.com/industry-analysis/devops-market>
- [7] Carter, M. (2021) Docker index shows momentum in developer community activity, dostupno na: <https://www.docker.com/blog/docker-index-shows-surging-momentum-in-developer-community-activity-again/>
- [8] Bass, L., Weber, I., Zhu, L., DevOps: A Software Architect’s Perspective. Addison-Wesley Professional, 2015.
- [9] Shahin, M., Ali Babar, M., Zhu, L., “Continuous integration, delivery and deployment: A systematic review on approaches, tools, challenges and practices”, IEEE Access, Vol. 5, 2017, str. 3909-3943.
- [10] Rehmann, K.-T., Folkerts, E., “Performance of containerized database management systems”, in Proceedings of the Workshop on Testing Database Systems, ser. DBTest’18. New York, NY, USA: Association for Computing Machinery, 2018, dostupno na: <https://doi.org/10.1145/3209950.3209953>
- [11] Kousiouris, G., Kyriazis, D., “Enabling containerized, parametric and distributed database deployment and benchmarking as a service”, in Companion of the ACM/SPEC International Conference on Performance Engineering, ser. ICPE ’21. New York, NY, USA: Association for Computing Machinery, 2021, str. 77–80, dostupno na: <https://doi.org/10.1145/3447545.3451188>
- [12] Turnbull, J., The Art of Monitoring, 2014.

- [13] Humble, J., "Continuous integration, delivery and deployment: A systematic review on approaches, tools, challenges and practices", in IEEE Software Engineering Workshop (SEW), 2015.
- [14] Hildred, T., "The history of containers", dostupno na: <https://www.redhat.com/en/blog/history-containers> 2015.
- [15] Rad, B. B., Bhatti, H. J., Ahmadi, M., "An introduction to docker and analysis of its performance", International Journal of Computer Science and Network Security (IJCSNS), Vol. 17, No. 3, 2017, str. 228.
- [16] Sayfan, G., Mastering kubernetes. Packt Publishing Ltd, 2017.
- [17] Moravcik, M., Kontsek, M., "Overview of docker container orchestration tools", in 2020 18th International Conference on Emerging eLearning Technologies and Applications (ICETA). IEEE, 2020, str. 475–480.
- [18] Kubernetes-Authors, "Kubernetes pods", dostupno na: <https://kubernetes.io/docs/concepts/workloads/pods/> Pristupljeno: 2024-06-22. 2024.
- [19] Kubernetes-Authors, "Kubernetes services", dostupno na: <https://kubernetes.io/docs/concepts/services-networking/service/> Pristupljeno: 2024-06-22. 2024.
- [20] Kubernetes-Authors, "Kubernetes deployments", dostupno na: <https://kubernetes.io/docs/concepts/services-networking/service/> Pristupljeno: 2024-06-22. 2024.
- [21] Kubernetes-Authors, "Kubernetes ingress", dostupno na: <https://kubernetes.io/docs/concepts/services-networking/ingress/> Pristupljeno: 2024-06-22. 2024.
- [22] Kubernetes-Authors, "Kubernetes configmaps", dostupno na: <https://kubernetes.io/docs/concepts/configuration/configmap/> Pristupljeno: 2024-06-22. 2024.
- [23] Kubernetes-Authors, "Kubernetes secrets", dostupno na: <https://kubernetes.io/docs/concepts/configuration/secret/> Pristupljeno: 2024-06-22. 2024.
- [24] Kubernetes-Authors, "Kubernetes replicasesets", dostupno na: <https://kubernetes.io/docs/concepts/workloads/controllers/replicaset/> Pristupljeno: 2024-06-22. 2024.
- [25] Kubernetes-Authors, "Kubernetes statefulsets", dostupno na: <https://kubernetes.io/docs/concepts/workloads/controllers/statefulset/> Pristupljeno: 2024-06-22. 2024.
- [26] Kubernetes-Authors, "Kubernetes cronjob", dostupno na: <https://kubernetes.io/docs/concepts/workloads/controllers/cron-jobs/> Pristupljeno: 2024-06-22. 2024.
- [27] Kubernetes-Authors, "Kubernetes pv and pvc", dostupno na: <https://kubernetes.io/docs/concepts/storage/persistent-volumes/> Pristupljeno: 2024-06-22. 2024.
- [28] Kubernetes-Authors, "Kubernetes custom resources", dostupno na: <https://kubernetes.io/docs/concepts/extend-kubernetes/api-extension/custom-resources/> Pristupljeno: 2024-06-22. 2024.
- [29] Kubernetes-Authors, "Kubernetes namespaces", dostupno na: <https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/> Pristupljeno: 2024-06-22. 2024.

- [30] Kubernetes-Authors, “Using rbac authorization”, dostupno na: <https://kubernetes.io/docs/reference/access-authn-authz/rbac/> Pristupljeno: 2024-06-22. 2024.
- [31] Helm-Documentation, “Using rbac authorization”, dostupno na: <https://helm.sh/docs/> Pristupljeno: 2024-06-22. 2024.
- [32] Turnbull, J., The Docker book. James Turnbull, 2016.
- [33] Crunchy Data Solutions, I., “Crunchy data postgresql operator documentation”, dostupno na: <https://access.crunchydata.com/documentation/postgres-operator/latest> Pristupljeno: 2024-06-09. 2024.
- [34] community, S., “Spring petclinic”, dostupno na: <https://github.com/spring-projects/spring-petclinic> Pristupljeno: 2024-06-22. 2024.
- [35] Mueller, T., “H2 database documentation”, dostupno na: <https://h2database.com/html/main.html> Pristupljeno: 2024-06-22. 2023.