|  | **Univerzitet u Nišu, Elektronski fakultet**  **Katedra za računarstvo** |  |
| --- | --- | --- |

**Mihajlo Veljković**

**Korišćenje PostgreSQL DBMS-a u Cloud okruženjima**

**Seminarski rad**

**Niš, 2023**

Sadržaj

[Uvod 3](#_Toc138456065)

[PostgreSQL i kontejnerizacija 4](#_Toc138456066)

[PostgreSQL u Kuberenetesu 6](#_Toc138456067)

[Alati za automatsko upravljanje instancama 10](#_Toc138456068)

[PGO 10](#_Toc138456069)

[PostgreSQL kod Cloud provajdera 12](#_Toc138456070)

[Azure database 12](#_Toc138456071)

[Cloud SQL 13](#_Toc138456072)

[Prednosti i mane upravljanih instanci 13](#_Toc138456073)

[Zaključak 14](#_Toc138456074)

[Literatura 15](#_Toc138456075)

# Uvod

PostgreSQL je popularan i svestran RDBMS otvorenog koda. Poznat je po dobrim performansama, skalabilnosti, pouzdanosti i bezbednosti. Sa ekspanzijom aplikacija u Cloud okruženjima, pojavila su se i rešenja za korišćenje PostgreSQL-a unutar njih.

Postoje više razloga zašto je PostgreSQL dobar izbor za korišćenje u Cloud-u. Otvorenog je koda, čime se mogu izbeći troškovi licenciranja i održavanja, a sa sve većom popularnošću, pojavljuju se veliki broj ekstenzija i biblioteka za integraciju ovog RDBMS-a u aplikacije. PostgreSQL se dobro skalira, što ga čini idealnim za praćenje potražnje unutar Cloud sistema. Poznat je po pouzdanosti, a projektat postoji već 26 godina, pa je sigurno da će se nastaviti ažuriranje i unapređivanje mogućnosti sa pojavom novih tehnologija.

U ovom radu biće opisani načini korišćenja i integracija PostgreSQL RDBMS-a unutar Cloud ukruženja, uključujući instalaciju u Kubernetes klasterima, kao i korišćenje gotovih rešenja poput Azure instanciranih baza podataka ili Google Cloud SQL-a.

# PostgreSQL i kontejnerizacija

Kako bi omogućili pokretanje PostgreSQL DBMS-a u Cloud okruženjima, potrebno je imati jednostavan način za instalaciju i pokretanje. Trenutno aktuelno rešenje za ovakav problem je kontejnerizacija aplikacija. Postoji zvanična Docker slika za korišćenje PostgreSQL koja se redovno ažurira i održava.

Pokretanje ove slike je vrlo jednostavno, moguće je direktno kroz terminal pomoću Docker-a:

$ docker run --name some-postgres -e POSTGRES\_PASSWORD=example -d postgres

Alternativno, moguće je iskoristiti i docker-compose fajl:

# Use postgres/example user/password credentials

version: '3.1'

services:

  db:

    image: postgres

    restart: always

    environment:

      POSTGRES\_PASSWORD: example

Nakon kreiranja fajla, pokrećemo docker-compose up u direktorijumu u kome se nalazi fajl. Oba primera će kreirati instancu PostgreSQL-a sa korisnikom *postgres* i šifrom *example* i biće dostupni na portu 5432.

Naravno, potrebna su dodatna podešavanja za korišćenje PostgreSQL-a u okviru aplikacija. Ova slika nudi promenljive okruženja za podešavanja nekih parametara:

* POSTGRES\_PASSWORD – Obavezna promenljiva koja podešava šifru *superuser* korisnika. Ne sme biti prazna ili nedefinisana. Slika podrazumeva *trust* metodu autentifikacije unutar kontejnera (alati kao što su *psql* unutar kontejenera ne zahtevaju šifru), dok za sve konekcije van kontejnera zahtevaju šifru.
* POSTGRES\_USER – Opcionalna promenljiva koja podešava korisničko ime *superuser* korisnika. Slika će pri inicijalizaciji kreirati ovog korisnika i bazu sa istim imenom. Podrazumevana vrednost je *postgres*.
* POSTGRES\_DB – Opcionalna promenljiva koja definiše ime baze kreirane nakon inicijalizacija. Podrazumevana vrednost je vrednost promenljive POSTGRES\_USER.
* POSTGRES\_INITDB\_ARGS – Opcionalni argumenti za prosleđivanje *initdb* komandi, razdvojeni razmakom
* POSTGRES\_INITDB\_WALDIR – Opcinalna promenljiva lokacije direktorijuma za transakcione logove.
* POSTGRES\_HOST\_AUTH\_METHOD – Opcionalna promenljiva za podešavanje metoda autentifikacije za *host* konekcije za sve baze, korisnike i adrese. Podrazumevana vrednost je *scram-sha-256*.
* PGDATA – Opcionalna promenljiva za promenu lokacije skladištenja podataka iz baze.

Ako ne želimo da prosleđujemo poverljive informacije direktno, za POSTGRES\_PASSWORD, POSTGRES\_USER, POSTGRES\_DB i POSTGRES\_INITDB\_ARGS promenljive je moguće proslediti i putanju do fajla unutar kontejnera.

Direktorijume i fajlove van kontejnera možemo zakačiti za kontejner pomoću *mount* komande ili skraćeno pomoću *-v*:

--mount 'type=volume,src=<VOLUME-NAME>,dst=<CONTAINER-PATH>,volume-driver=local,volume-opt=type=nfs,volume-opt=device=<nfs-server>:<nfs-path>,"volume-opt=o=addr=<nfs-address>,vers=4,soft,timeo=180,bg,tcp,rw"'

-v src:dst:options

Pomoću ovih komandi je moguće koristiti fajlove iz host operativnog sistema unutar kontejnera. Takođe, moguće je zameniti konfiguracione fajlove PostgreSQL-a direktno (npr. *postgresql.conf* i *pg\_hba.conf* fajlove). Pri modifikaciji konfiguracionih fajlova, treba obratiti pažnju na to da je DBMS unutar kontejnera, i da uz to postoje određenja ograničenja. Najčešći problemi su ne podešavanje opcije *listen\_addresses = '\*'* u *postgresql.conf* fajlu i pogrešno podešavanje *pg\_hba.conf* fajla.

Moguće je primeniti i skripte pri inicijalizaciji baze dodavanjem *.sql*, *.sql.gz* ili *.sh* skripti unutar */docker-entrypoint-initdb.d* direktorijuma. Nakon izvršavanja *initdb*, slika pokreće skripte iz direktorijuma ukoliko je direktorijum sa podacima prazan. Još jedan čest problem pri konfiguraciji je da ukoliko skripte za inicijalizaciju nisu izvršene uspešno i kontejner se restartuje, a direktorijum sa podacima nije resetovan, slika neće ponovo izvršiti skripte niti će nastaviti izvršavanje od prethodnog pokušaja. Sve skripte se pokreću pod *postgres* korisnikom unutar kontejnera.

Još jedna napomena pri pokretanju slike je da ukoliko ne postoje podaci sa bazom, podrazumeva se da će se izvršiti inicijalizacija baze, pri čemu će DBMS odbijati sve konekcije. Potrebno je uračunati ovo ponašanje, pogotovo pri korišćenju *docker-compose* alata sa više servisa. Takođe, podrazumevana količina deljene memorije unutar kontejnera je 64MB, pa ukoliko se desi da kontejner potroši svu memoriju, potrebno je drugačije pokrenuti kontejner i povećati količinu deljene memorije.

Sliku je moguće nadograditi sa svim željenim ekstenzijama i modulima, neki moduli poput *PostGIS*-a zahtevaju samo instalaciju paketa, dok neki moduli zahtevaju kompajliranje celog projekta, pa vrlo često postoje i gotovo slike sa već instaliranim ekstenzijama.

# PostgreSQL u Kuberenetesu

Kubernetes je prenosiva i proširiva platforma otvorenog koda za upravljanje kontejnerizovanim jedinicama rada i servisima, koja podržava i deklarativnu konfiguraciju i automatizaciju. Uz pomoć Kubernetesa, možemo napraviti Database-as-a-service instancu PostgreSQL-a bez podešavanja potrebne infrastrukture. Polazimo od pretpostavke da postoji Kubernetes klaster koji možemo da modifikujemo (nebitno da li je bare metal ili podešen pomoću nekog provajdera). Korišćenjem Kubernetesa možemo da omogućimo lako skaliranje baze podataka (jer PostgreSQL ima dobru podršku za replikaciju i skaliranje) i time omogućiti dobar vremenski odziv zahtevima. Kubernetes takođe lako omogućava upravljanjem instanci PostgreSQL-a. Kontrolna ravan Kubernetesa omogućava praćenje statusa podova, pa u slučaju greške (npr. pucanja poda), klaster će automatski napraviti novi pod da preuzme posao prethodnog. Resursi definisani u .*yaml* fajlovima možemo opet instancirati na nekom drugom klasteru, pa i migrirati postojeću bazu pomoću funkcija bekapovanja i vraćanja podataka ugrađenih u PostgreSQL.

Kubernetes podovi (osnovne jedinice izvršavanja) koriste Docker slike u pozadini, pa nakon nalaženja odgovarajuće slike možemo definisati *Deployment* objekte koji će upravljati podovima i osigurati postojanje onoliko podova koliko je definisano u objektu. Za definisanje objekta, potrebno nam je skladištenje podataka i konfiguracioni fajlovi koje će slika pročitati. Podatke čuvamo u volumenu definisanom pomoću *PersisentVolume* objekta:

apiVersion: v1

# Kind for volume chain

kind: PersistentVolume

metadata:

  # Name the persistent chain

  name: postgresdb-persistent-volume

  # Labels for identifying PV

  labels:

    type: local

    app: postgresdb

spec:

  storageClassName: manual

  capacity:

    # PV Storage capacity

    storage: 8Gi

  # A db can write and read from volumes to multiple pods

  accessModes:

    - ReadWriteMany

  # Specify the path to persistent the volumes

  hostPath:

    path: "/data/db"

Kako bi klaster mogao da rezerviše volumen, potrebno je definisati i PersistentVolumeClaim objekat:

apiVersion: v1

# define a resource for volume chain

kind: PersistentVolumeClaim

metadata:

  # Name the volume chain

  name: db-persistent-volume-claim

spec:

  storageClassName: manual

  accessModes:

    # Allow ReadWrite to multiple pods

    - ReadWriteMany

  # PVC requesting resources

  resources:

    requests:

      # the PVC storage

      storage: 8Gi

Konfiguracioni fajl definišemo pomoću ConfigMap objekta:

apiVersion: v1

# Kind for kubernets ConfigMap

kind: ConfigMap

metadata:

  # Name your ConfigMap

  name: db-secret-credentials

  labels:

    app: postgresdb

data:

  # User DB

  POSTGRES\_DB: testDB

  # Db user

  POSTGRES\_USER: testUser

  # Db password

  POSTGRES\_PASSWORD: testPassword

U ovom slučaju definišemo samo promenljive okruženja, ali moguće je definisati i ceo *postgresql.conf* fajl u *data* polju:

apiVersion: v1

# Kind for kubernets ConfigMap

kind: ConfigMap

metadata:

  # Name your ConfigMap

  name: db-config

  labels:

    app: postgresdb

data:

  postgresql.conf:

      # This is a comment

    log\_connections = yes

    log\_destination = 'syslog'

    search\_path = '"$user", public'

    shared\_buffers = 128MB

Definišemo Deployment objekat koji će koristiti naš volumen i konfiguracioni fajl:

  # Kubernetes API version

apiVersion: apps/v1

# Deployment object

kind: Deployment

metadata:

  # The name of the Deployment

  name: postgresdb

spec:

  # Replicas for this Deployment

  replicas: 3

  selector:

    # labels the pods

    matchLabels:

      app: postgresdb

  template:

    metadata:

      labels:

        # The label the pods created from the pod template should have

        app: postgresdb

    spec:

      containers:

        # The container name to execute pods

        - name: postgresdb

          # pull postgresimage from docker hub

          image: postgres

          ports:

            # Assign ports to expose container

            - containerPort: 5432

          envFrom:

            # Load the environment variables/PostgresSQL credentials

            - configMapRef:

                # This should be the ConfigMap name created ealier

                name: db-secret-credentials

          volumeMounts:

            # The volume mounts  for the container

            - mountPath: /var/lib/postgres/data

              name: db-data

      # Volumes attached to the pod

      volumes:

        - name: db-data

          persistentVolumeClaim:

            # reference the PersistentVolumeClaim

            claimName: db-persistent-volume-claim

U slučaju da smo želeli ceo konfiguracioni fajl, umesto definisanja konfiguracije u polju *envFrom*, koristili bi volumen u *volumeMounts* uz definiciju volumena u *volumes* polju:

volumeMounts:

        - name: config-volume

          mountPath: /etc/postgresql/

      volumes:

        - name: config-volume

          configMap:

        # Provide the name of the ConfigMap containing the files you want

        # to add to the container

            name: db-config

Potrebno je definisati i *Service* objekat kako bi omogućili pristup podu preko mreže:

apiVersion: v1

# Kind for service

kind: Service

metadata:

  # Name your service

  name: postgresdb

  labels:

    app: postgresdb

spec:

  # Choose how to expose your service

  type: NodePort

  ports:

    # The port number to expose the service

    - port: 5432

  # Pod to route service traffic

  selector:

    app: postgresdb

Sve definicije možemo spojiti u jedan .*yaml* fajl (spajamo ih pomoću --- karaktera između definicija) i pokretanjem:

kubectl apply -f <ime\_fajla>.yaml

Kubernetes će kreirati sve potrebne resurse i pokrenuti podove.

## Alati za automatsko upravljanje instancama

Ručno definisanje .*yaml* fajlova može biti naporan posao. Definisanje svih objekta svaki put kada želimo novu instancu baze ili promenu nad postojećom instancom nije efikasno rešenje. Kao rešenje ovog problema uvodi se koncept *operatora*, softvera koji kontroliše kompleksne aplikacije umesto administratora i vodi računa o očuvanju podataka, replikaciji, skaliranju postojećih podova kao i olakšavanje instanciranja novih. Jedan od popularnijih operatora za PostgreSQL je Crunchy Data PGO.

### PGO

Cilj PGO alata je deklarativno rešenje za PostgreSQL koje automatski upravlja klasterima. Jedini korak potreban pre instalacije ovog alata je omogućavanje dinamičkog kreiranja volumena. „*Bare metal*“ klasteri vrlo često nemaju podršku za dinamičke volumene, pa je potrebno instalirati neku vrstu snabdevača. Za te potrebe može se iskoristiti NFS server i NFS snabdevač. U slučaju da NFS server nije instaliran, potrebno je pokrenuti:

apt-get install nfs-utils nfs-utils-lib

Nakon instalacije, potrebno je konfigurisati server kreiranjem fajla */etc/exports* sa željenim direktorijumom na host mašini:

/srv/nfs/pg \*(rw,sync,no\_subtree\_check,no\_root\_squash,insecure)

Kao snabdevača možemo koristiti <https://github.com/kubernetes-sigs/nfs-subdir-external-provisioner>. Instalaciju vršimo komandom uz promenu odgovarajućih vrednosti:

helm install my-release nfs-subdir-external-provisioner/nfs-subdir-external-provisioner \

    --set nfs.server=x.x.x.x \

    --set nfs.path=/exported/path

Instalaciju PGO alata vršimo forkovanjem [https://github.com/CrunchyData/postgres-operator-](https://github.com/CrunchyData/postgres-operator-examples)examples i modifikacijom *kustomize/postgres/postgres.yaml* fajla. Potrebno je dodati polje:

storageClassName: "nfs-storageclass"

Nakon modifikacije, pokrećemo instalaciju komandom:

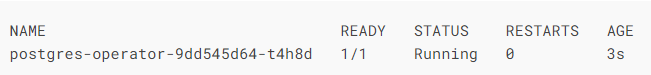
kubectl apply -k kustomize/install/namespace

Proveravamo da li je instalacija uspešna pomoću:

kubectl -n postgres-operator get pods \

--selector=postgres-operator.crunchydata.com/control-plane=postgres-operator --field-selector=status.phase=Running

U slučaju uspešne instalacije, trebalo bi da vidimo:



Nakon instalacije, možemo instancirati PostgreSQL na sledeći način:

kubectl apply -k kustomize/postgres

PGO će iskoristiti prethodno modifikovani *postgres.yaml* fajl za instanciranje, pa sve neophodne promene možemo upisati u taj fajl, uključujući promenu konfiguracija, volumena, promenljiva, pa i broj podova. Možemo i izmeniti postojeću instancu baze ako ne menjamo ime resursa, već samo menjamo konfiguraciju koristeći istu komandu kao za kreiranje instance. Brisanje instance se vrši pomoću komande:

kubectl delete -k kustomize/postgres

PGO u pozadini kreira sve potrebne resurse za korišćenje instance baze. Kredencijale baze možemo naći u *Secret* objektu sa šablonskim imenom:

<clusterName>-pguser-<userName>

U okviru objekta postoje sledeća polja:

* user – korisničko ime
* password – šifra
* dbname – ime podrazumevane baze podataka
* host – identifikator hosta
* port – broj porta
* uri – konekcioni string u formatu PostgreSQL-a (postgresql://[userspec@][hostspec][/dbname][?paramspec])
* jdbc-uri – konekcioni string u formatu za korišćenje pomoću JBDC drajvera

Ove vrednosti možemo korisiti direktno unutar aplikacija u klasteru pomoću referenciranja na *Secret* objekat ili ih pročitati i ručno uneti u konfiguracione fajlove aplikacija.

Prednost korišćenja PGO-a je što alat vodi računa o svim resursima i u slučaju greške bilo kog pokušava brzi oporavak i vraćanje u traženo stanje. U primeru ručnog kreiranja resursa, da smo obrisali bilo koji od objekata (npr. *Service* objekat), Kubernetes platforma ne bi ni obavestila administatora o problemu, već bi aplikacije dobijale greške pristupa bazi. PGO bi odmah rekreirao objekat, ili u slučaju greške, prijavio problem sa instancom baze kako bi administrator mogao da reši problem.

# PostgreSQL kod Cloud provajdera

U slučaju da korisnik ne želi da prolazi kroz proces instalacije Kubernetesa ili podešavanja operatora, Cloud provajderi nude i opciju upravljanih instanci PostgreSQL DBMS-a. Korisnik može da izabere provajdera, veličinu instance i dobija gotovu instancu spremnu za korišćenje unutar aplikacija.

## Azure database

Microsoft Azure platforma nudi upravljane instance PostgreSQL-a sa sledećim mogućnostima:

* Ugrađena velika dostupnost
* Zaštita podataka i povratak unazad do 35 dana
* Automatsko održavanje infrastrukture (ažuriranje sistema i hardvera)
* Stabilne, predvidljive performanse
* Brzo skaliranje
* Veliki nivo sigurnosti i industrijski standardi bezbednosti podataka
* Praćenje instanci i automatizacija operacija
* Brza i kvalitetna podrška

Postoje dva modela instanci: pojedinačni server i fleksibilni server. Pojedinačni server je namenjen kao minimalna instanca sa jednostavnim opcijama za konfiguraciju. Podržava verzije 9.5, 9.6, 10 i 11 i nudi 99.99% dostupnosti. Korisnik bira nivo hardvera u zavisnosti od potrebe aplikacije:

* Basic – osnovni nivo sa 1-2 vCPU i do 1 TB podataka
* General purpose – do 64 vCPU i 16TB podataka
* Memory optimized – do 32 vCPU i 16 TB podataka sa većim protokom podataka

Korisnik bira korisničko ime i šifru i nakon instanciranja baze podataka, ne mora da vodi računa o infrastrukturi i može direktno koristiti DBMS u aplikaciji. Treba napomenuti da su sve mašine sa Windows operativnim sistemima za ovu vrstu servera.

Fleksibilni server je nova generacija instanci PostgreSQL-a koja nudi novije verzije (11-15). Mašine koriste Linux operativni sistem i imaju ugrađeni *PgBouncer* alat za keširanje konekcija. Podržan je i veći broj konekcija (do 5000). Ne postoje nivoi servera, već korisnik samo bira hardver na kome će biti instanca. Azure nudi predefinisane opcije za izbor hardvera, ali je moguće promeniti virtuelne mašine na kojima se izvršava instanca baze. Ova vrsta servera nudi i opciju velike dostupnosti nad više geografskih zona i podršku za logičku replikaciju. Cene servera u nekim slučajevima mogu biti jeftinije za pojedinačne servera, dok u nekim drugim slučajevima su fleksibilni serveri jeftinija opcija. Na korisniku je izbor vrste servera u zavisnosti od potreba aplikacije i očekivanog broja korisnika.

## Cloud SQL

Google nudi Cloud SQL platformu za instanciranje PostgreSQL-a sa sledećim mogućnostima:

* Serverske mašine sa do 624 GB RAM memorije i 96 CPU-a
* Do 64 TB podataka
* Migracija podataka pomoću DMS (Database Migration Service)
* Enkripciju podataka
* Zaštićene konekcije za Cloud SQL Auth ili SSL/TLS protokolom
* Replikacija podataka između geografskih zona
* Dampovanje podataka
* Automatsko čuvanje rezervnih kopija
* Kloniranje instanci
* Logička replikacija

Cloud SQL ne dozvoljava bilo koje operacije koje zahtevaju *superuser* privilegije (osim kreiranje podržanih ekstenzija), ne podržava pozadinske radnike, komande sa rekonekcijom i nisu podržane sve opcije za konfiguraciju.

Cloud SQL nudi više opcija za konfiguraciju instance, ali isto tako za neke korisnike može biti i previše opcija. Kao i sa Azure instancama, moguće je podesiti korisnike, korisničko ime i šifru, pa je integracija u okviru aplikacija jednostavna. Na korisniku je odabir mašine i konfiguracije promenljiva baze podataka.

Cloud SQL se pokazao kao jedan od bržih provajdera PostgreSQL instanci.

Postoje još provajdera instanci podataka, uključujući Amazon AWS, DigitalOcean, IBM itd. Pored velikih Cloud provajdera, postoje i specijalizovani provajderi koji služe samo za instanciranje baza podataka ili njihovih infrastruktura (npr. Aiven ili Database Labs).

## Prednosti i mane upravljanih instanci

Upravljane instance baza olakšavaju posao korisniku i smanjuju proces i vreme potrebno do spremne baze podataka. Na korisniku je da proceni koliko resursa bi trošila njegova instanca i uvek je u mogućnosti da proširi ili smanji svoju instancu od zavisnosti od potreba. Problem se javlja kada je korisniku potrebno više pristupa nad podešavanjima nego što provajderi dozvoljavaju. Ekstenzije su vrlo često ograničene, a u nekim slučajevima čak i zabranjene. Određena podešavanja su takođe zabranjena, pa za specifične slučajeve korišćenja korisnik ipak treba da odabere hostovanje na svom klasteru, ili ako nema potrebe za brzim skaliranjem, korisnik može odlučiti i hostovanje direktno na svojim serverima. Provajderi ponekad mogu biti spori za ažuriranje određenih bezbednosnih problema ili problema sa konkretnom verzijom PostgreSQL-a, a vrlo često su i ograničene verzije, pa treba i to uzeti u obzir prilikom odluke. Zbog prirode SQL baza podataka, kao i dobrih opcija za bekapovanje PostgreSQL baza, čak i pri pogrešnom odabiru, korisnik nije zaključan na određenog provajdera, pa može lako izabrati drugog ili preći na sopstveno rešenje.

# Zaključak

Zbog razvoja kontejnerizacije i dobre podrške PostgreSQL tima za kreiranje slika i korišćenje njihove aplikacije unutar kontejnera, videli smo da ovaj DBMS može da bude pogodan za korišćenje u Cloud okruženjima. Najveće usko grlo kod ovakvih slučajeva korišćenja je brzina diskova, pa treba obratiti pažnju prilikom odabira rešenja na performanse diskova unutar mašina koje će ih hostovati.

Kubernetes klasteri omogućavaju veću kontrolu i potencijalno veće performanse instanci, ali zahtevaju stručno lice za održavanje i podešavanje opcija. Pored toga, potrebno je imati i Kubernetes klaster koji korisnik mora hostovati na svojoj mašini ili opet iznajmljivati od nekog Cloud provajdera. Operatori olakšavaju posao kreiranja i održavanja instanci, ali nisu savršeni i moraju često da se ažuriraju i ne eliminišu potrebu za stručnim licem.

Upravljane instance nude najbrža rešenja, ali su i najviše ograničena pa samim tim nisu univerzalna rešenja. Potrebno je razumevanje PostgreSQL-a za izvlačenje maksimalnih performansi, a u nekim slučajevim to nije ni moguće zbog limitacija provajdera.

Pored svih faktora, najveći uticaj na odabir rešenja su: veličina i kompleksnost aplikacije, budžet projekta kao i nivo stručnosti lica koji upravljaju bazom. Sva rešenja napomenuta u ovom radu mogu vršiti funkciju proste baze podataka, ali je potrebno znanje i iskustvo za biranje pravog rešenja i optimizaciju istog.

# Literatura

<https://github.com/docker-library/docs/blob/master/postgres/README.md>

<https://cloudnativenow.com/kubecon-cnc-eu-2022/why-run-postgres-in-kubernetes/>

<https://sweetcode.io/how-to-deploy-postgresql-instance-to-kubernetes/>

<https://portworx.com/blog/choosing-a-kubernetes-operator-for-postgresql/>

<https://access.crunchydata.com/documentation/postgres-operator/v5/>

<https://www.prisma.io/dataguide/postgresql/5-ways-to-host-postgresql#databases-managed-by-cloud-providers>

<https://learn.microsoft.com/en-us/azure/postgresql/single-server/overview>

<https://cloud.google.com/sql/postgresql>

<https://aiven-io.medium.com/benchmarking-postgresql-across-the-cloud-648921d5fab9>