

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 903

**PROTOTIP SUSTAVA ZA LOKALNO UPRAVLJANJE
INFRASTRUKTUROM APLIKACIJA**

Karlo Josip Kardum

Zagreb, srpanj 2025.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 903

**PROTOTIP SUSTAVA ZA LOKALNO UPRAVLJANJE
INFRASTRUKTUROM APLIKACIJA**

Karlo Josip Kardum

Zagreb, srpanj 2025.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 3. ožujka 2025.

DIPLOMSKI ZADATAK br. 903

Pristupnik: **Karlo Josip Kardum (0036529821)**
Studij: Računarstvo
Profil: Programsко инжењерство и информациски системи
Mentor: prof. dr. sc. Igor Mekterović

Zadatak: **Prototip sustava za lokalno upravljanje infrastrukturom aplikacija**

Opis zadatka:

Softverski sustavi se sve češće oslanjaju na infrastrukturu u oblaku (engl. cloud infrastructure) kako bi osigurali skalabilnost, automatizaciju i pouzdano upravljanje resursima poput baza podataka, web-aplikacija i posrednika poruka. No, korištenje komercijalnih platformi može biti financijski neisplativo za manje projekte, laboratorijske postavke ili edukacijske scenarije, dok je njihova ručna implementacija često prezahtjevna za timove s ograničenim resursima. Kako bi se riješio taj problem, ovaj rad istražuje mogućnost razvoja samostalnog okruženja s vlastitim poslužiteljem (engl. self-hosted) koje pruža funkcionalnosti nalik uslugama u oblaku, ali uz potpunu kontrolu nad infrastrukturom na vlastitom hardveru ili virtualnom privatnom poslužitelju (engl. VPS). Fokus je na jednostavnosti postavljanja i održavanja, čime se omogućuje brži prijelaz iz razvojne faze u fazu produkcije bez potrebe za dubinskim poznavanjem tehnologija koje stoje u pozadini. Cilj rada je razviti sustav temeljen na Docker tehnologiji koji omogućuje kontejnerizaciju i automatizirano upravljanje softverskim resursima. Ključne funkcionalnosti uključuju: jednostavno postavljanje i upravljanje resursima poput baza podataka, web aplikacija i posrednika poruka kroz unaprijed definirane konfiguracije. Dodatno, potrebno je ostvariti centralizirano upravljanje koje omogućuje praćenje metrika i provjeru zdravlja sustava te potencijalnu optimizaciju performansi. Osmisliti model "upravljanja resursa", gdje sustav automatski optimizira i integrira standardizirane komponente u cijelovitu infrastrukturu. Koristiti modularnu arhitekturu koja olakšava prilagodbu različitim potrebama korisnika i omogućuje nadogradnju sustava. Za implementaciju razmotriti .NET Whale paket za rad s Docker API-jem. Osim same implementacije, analizirati praktičnu primjenu ovakvog rješenja u različitim scenarijima i donijeti ocjenu ostvarenog pristupa kroz testiranje na nekoliko pokaznih sustava.

Rok za predaju rada: 4. srpnja 2025.

Sadržaj

| | |
|--|----|
| Uvod | 1 |
| 1. Upravljanje infrastrukturom web-aplikacija..... | 2 |
| 1.1. Komponente sustava za upravljanje infrastrukturom | 2 |
| 1.1.1. Izolacija | 2 |
| 1.1.2. Skaliranje..... | 3 |
| 1.1.3. Unutarnje umrežavanje..... | 4 |
| 1.1.4. Otvaranje prema internetu | 4 |
| 1.1.5. Telemetrija..... | 5 |
| 1.2. Pristupi oblikovanja sustava za upravljanje infrastrukturom | 5 |
| 1.2.1. Neupravljeni sustavi | 6 |
| 1.2.2. Upravljeni sustavi..... | 7 |
| 2. Idejno rješenje: sustav za upravljanje vlastitom infrastrukturom - Cloudy | 9 |
| 2.1. Resursi – funkcionalni zahtjevi | 10 |
| 2.1.1. Web-aplikacije..... | 12 |
| 2.1.2. Baze podataka..... | 19 |
| 2.1.3. Posrednici poruka | 24 |
| 2.1.4. Virtualne mreže | 26 |
| 2.2. Telemetrija..... | 27 |
| 2.2.1. Agregacija telemetrijskih podataka | 27 |
| 2.2.2. Prikaz i analiza telemetrijskih podataka | 28 |
| 2.3. Otvaranje prema internetu | 28 |
| 2.4. Virtualni resurs domaćina..... | 30 |
| 2.5. Ostali zahtjevi..... | 33 |
| 3. Implementacija | 36 |
| 3.1. Arhitektura sustava..... | 36 |

| | | |
|--------|--|----|
| 3.2. | Pregled modela podataka..... | 37 |
| 3.3. | Korišteni razvojni okviri i biblioteke..... | 38 |
| 3.3.1. | ASP.NET Core | 39 |
| 3.3.2. | MediatR i FluentValidation..... | 39 |
| 3.3.3. | Entity Framework Core | 39 |
| 3.3.4. | React i Vite s TypeScriptom..... | 40 |
| 3.3.5. | Mantine i Mantine Datatables..... | 41 |
| 3.3.6. | Redux i RTK Query..... | 41 |
| 3.3.7. | React Chart.js 2..... | 41 |
| 3.4. | Integrirani alati | 42 |
| 3.4.1. | Docker | 42 |
| 3.4.2. | Alati za telemetriju | 43 |
| 3.4.3. | Caddy..... | 44 |
| 3.4.4. | Postgres..... | 45 |
| 3.4.5. | Kafka | 45 |
| 3.5. | Kontejnerizacija sustava..... | 45 |
| 3.6. | Upravljanje sa izoliranim resursima..... | 47 |
| 3.7. | Umrežavanje kontejnera iza resursa | 48 |
| 4. | Testiranje sustava | 50 |
| 4.1. | Projekt DevToolsDash..... | 50 |
| 4.2. | Projekt Alpinity | 53 |
| 4.3. | Laboratorijski primjer – posrednici poruka | 54 |
| 5. | Budući rad..... | 57 |
| 6. | Zaključak | 59 |
| 7. | Literatura | 60 |

Uvod

Softverski sustavi se sve češće oslanjaju na infrastrukturu u oblaku (engl. *Cloud Infrastructure*) kako bi osigurali skalabilnost, automatizaciju i pouzdano upravljanje resursima poput baza podataka, web-aplikacija i posrednika poruka. Međutim te platforme su finansijski zahtjevne, posebno kada su u pitanju manji projekti, laboratorijski uvjeti ili edukacijski scenariji. Iz cjenovnih razloga u njima se ponekad upotrebljava vlastita infrastruktura, bili to virtualni privatni poslužitelji (engl. *Virtual Private Server - VPS*) ili infrastruktura na vlastitom hardveru (engl. *self-hosted*) koji zahtijevaju više vremena i poznavanja potrebnih tehnologija i alata za održavanje.

Kako bi se riješio taj problem, ovaj rad istražuje spajanje tih dvaju pristupa kroz sustav koji pruža funkcionalnosti nalik onima u oblaku, ali uz potpunu kontrolu nad infrastrukturom. Fokus rada je na jednostavnosti postavljanja samog sustava, a zatim na jednostavnosti postavljanja i održavanja komponenti infrastrukture čime se omogućuje brži prijelaz iz faze razvoja u fazu produkcije.

Razvijen sustav ima cilj omogućiti jednostavno postavljanje i upravljanje web-aplikacijama, bazama podataka i posrednicima poruka. Dodatno cilj je osmisliti model „upravljanih resursa“ koji dolaze sa preporučenim unaprijed definiranim konfiguracijama, automatskom i centraliziranom telemetrijom te pomoćnim alatima koji pojednostavljaju njihovo postavljanje, optimiziranje, ispravljanje pogrešaka kao i spajanje u cjelinu sustava kojem pripadaju.

Implementacija sustava temeljena je na Docker tehnologiji. Sustav pomoću paketa Docker.DotNet (prethodno *Whale*) koji apstrahira Docker API upravlja svim dijelovima infrastrukture. Uz implementaciju je provedeno testiranje sustava u nekoliko različitih scenarija kako bi se pokazala njegova učinkovitost i uočile prilike za budući razvoj.

1. Upravljanje infrastrukturom web-aplikacija

Upravljanje infrastrukturom odnosi se na skup procesa i alata za kontrolu, optimizaciju i održavanje resursa potrebnih za izvođenje softverskih sustava. U kontekstu web-aplikacija, to uključuje upravljanje poslužiteljima, bazama podataka, mrežnim komponentama i drugim elementima koji omogućuju funkcioniranje aplikacije. Svaka instanca jednog od tih elemenata predstavlja jedan resurs u sklopu sustava. Sustav za upravljanje infrastrukturom centralizira ove aktivnosti, nudeći alate za automatizaciju, praćenje i kontrolu resursa, čime se smanjuje operativni napor i povećava pouzdanost. U nastavku se analiziraju glavne komponente sustava, te se uspoređuju pristupi dizajniranja sustava.

1.1. Komponente sustava za upravljanje infrastrukturom

Sustavi za upravljanje infrastrukturom sastoje se od međusobno povezanih komponenti koje rješavaju specifične izazove operacijskog okruženja. Glavne od njih su

- izolacija – predstavlja osiguranje da jedan dio sustava ne može nekontrolirano utjecati na drugi
- skaliranje – povećanjem snage ili količine dostupnog hardvera povećamo performanse sustava
- unutarnje umrežavanje – omogućuje sigurnu komunikaciju između izoliranih resursa
- otvaranje prema internetu – proslijeduje zahtjeve vanjskih korisnika ciljnim resursima (na temelju domene, mrežnog priključka, ...)
- telemetrija – prikupljanje podataka o radu i zdravlju sustava kroz vrijeme

Osim toga, sustav često sadržava i standardne funkcionalnosti poput prikupljanja dnevničkih zapisu i metrika, autentifikacije, autorizacije, dnevnika korištenja i drugih generalnih funkcionalnosti softverskih rješenja.

Svaka od ključnih funkcionalnosti sustava za upravljanje infrastrukturom je u nastavku detaljnije opisana:

1.1.1. Izolacija

Izolacija unutar sustava za upravljanje infrastrukturom osigurava da pojedini resursi ili aplikacije rade neovisno jedni o drugima. Na taj način sprječava se mogućnost da greške,

preopterećenja ili sigurnosni incidenti unutar jednog dijela sustava utječu na ostale komponente. Izolacija se najčešće ostvaruje korištenjem virtualizacijskih tehnologija poput virtualnih strojeva (engl. *virtual machine*) ili kontejnera (engl. *container*), pri čemu svaki resurs dobiva vlastito izolirano okruženje s ograničenim pristupom zajedničkim resursima [1].

Stariji sustavi za upravljanje infrastrukturom, poput cPanel sustava, ne nude izolaciju na razini pojedinih resursa te ona pripada modernijim komponentama sustava za upravljanje infrastrukturom i često se izvodi pomoću kontejnerizacije [2].

Kontejnerizacija, kao suvremeni pristup izolaciji, omogućuje pokretanje više aplikacija na istom operacijskom sustavu, pri čemu je svaka aplikacija smještena u vlastiti kontejner s definiranim resursima i mrežnim pravilima. Time se postiže visoka razina sigurnosti i predvidljivosti ponašanja sustava, a istovremeno se olakšava održavanje i nadogradnja pojedinih komponenti bez utjecaja na ostatak infrastrukture.

Dodatno, izolacija omogućuje precizno upravljanje resursima, poput procesorske snage, memorije i prostora za pohranu, čime se smanjuje mogućnost međusobnog ometanja aplikacija.

1.1.2. Skaliranje

Skaliranje je mogućnost prilagodbe kapaciteta resursa prema trenutnim potrebama aplikacija i korisnika. U kontekstu web-aplikacija, skaliranje podrazumijeva mogućnost povećanja ili smanjenja broja instanci aplikacija, baze podataka ili drugih servisa, kao i dodjelu dodatnih računalnih, memorijskih ili mrežnih resursa.

Postoje dvije osnovne vrste skaliranja: vertikalno skaliranje (engl. *vertical scaling*) i horizontalno skaliranje (engl. *horizontal scaling*). Vertikalno skaliranje ostvaruje se povećanjem resursa unutar jedne instance, uglavnom dodjelom više procesorske snage ili memorije virtualnom stroju ili kontejneru. Horizontalno skaliranje, s druge strane, uključuje dodavanje novih instanci iste aplikacije ili servisa, čime se omogućuje ravnomjerna raspodjela opterećenja i povećava dostupnost sustava.

Suvremeni sustavi za upravljanje infrastrukturom često implementiraju automatizirane mehanizme za skaliranje, koji na temelju prikupljenih metrika (npr. opterećenja procesora, broja aktivnih korisnika ili vremena odziva) automatski prilagođavaju broj ili veličinu resursa.

1.1.3. Unutarnje umrežavanje

Unutarnje umrežavanje predstavlja skup mehanizama i pravila kojima se omogućuje sigurna komunikacija između pojedinih resursa unutar infrastrukture. U sustavima za upravljanje infrastrukturom, unutarnje umrežavanje omogućuje povezivanje servisa na način koji sprječava neautorizirani pristup i smanjuje mogućnost neželjenih interferencija.

Implementacija unutarnjeg umrežavanja najčešće se ostvaruje korištenjem virtualnih mreža (engl. *virtual network*) i mrežnih pravila koja definiraju dozvoljene komunikacijske putove između resursa. Svaki resurs može biti smješten u zasebnu mrežnu domenu, čime se dodatno povećava razina izolacije i sigurnosti. Korištenjem mrežnih politika moguće je precizno odrediti koji servisi mogu međusobno komunicirati, a koji su odvojeni.

Osim sigurnosti, unutarnje umrežavanje omogućuje optimizaciju performansi putem lokalizacije prometa.

Unutarnje umrežavanje čini temelj za izgradnju složenijih arhitektura, poput mikroservisnih sustava, gdje je potrebno osigurati kontroliranu razmjenu podataka između velikog broja izoliranih komponenti.

1.1.4. Otvaranje prema internetu

Otvaranje prema internetu odnosi se na skup tehnika i alata kojima se omogućuje pristup resursima unutar infrastrukture iz vanjskih mreža, uz istovremeno očuvanje sigurnosti i kontrole nad prometom. U sustavima za upravljanje infrastrukturom, ova funkcionalnost obuhvaća prosljeđivanje zahtjeva korisnika prema odgovarajućim servisima, filtriranje prometa te provedbu sigurnosnih politika.

Glavne komponente za otvaranje prema internetu uključuju *reverse proxy* poslužitelje, *API gateway* sustave i alate za upravljanje pristupom poput *front door* servisa. *Reverse proxy* (npr. *Nginx*, *Caddy*) preuzima zahtjeve s interneta i prosljeđuje ih odgovarajućim internim resursima, omogućujući centralizirano upravljanje prometom, balansiranje opterećenja i provedbu sigurnosnih pravila. *API gateway* dodatno omogućuje autentifikaciju, autorizaciju i ograničavanje prometa prema aplikacijskim sučeljima (engl. *Application Programming Interface – API*).

Za zaštitu podataka u prijenosu standardno se koristi HTTPS protokol (engl. *Hypertext Transfer Protocol Secure*), koji omogućuje enkripciju komunikacije između korisnika i

servisa kao proširenje HTTP protokola s TLS/SSL enkripcijom. Korištenje HTTPS-a danas je obavezno iz sigurnosne perspektive, no izdavanje certifikata može predstavljati dodatni trošak. Popularno rješenje za besplatne TLS certifikate je *Let's Encrypt*. Zbog kratkog trajanja takvih certifikata, kao i radi unaprjeđenja jednostavnosti korištenja, automatsko upravljanje TLS/SSL certifikatima često je integrirano u *reverse proxy* rješenja, čime se dodatno pojednostavljuje implementacija sigurnih veza.

1.1.5. Telemetrija

Telemetrija predstavlja skup procesa i alata za prikupljanje, pohranu i analizu podataka o radu i stanju sustava tijekom vremena. U sustavima za upravljanje infrastrukturom, telemetrija omogućuje kontinuirano praćenje performansi, dostupnosti i sigurnosti svih resursa, što je od velike važnosti za pravovremeno otkrivanje problema, optimizaciju rada i donošenje odluka o dalnjem razvoju infrastrukture.

Osnovni elementi telemetrije uključuju prikupljanje metrika (npr. opterećenje procesora, iskorištenost memorije, broj zahtjeva), agregaciju i pohranu dnevničkih zapisa (engl. *log*), te praćenje događaja i upozorenja. Moderni sustavi često koriste specijalizirane alate za telemetriju, poput Prometheusa [3] za prikupljanje i pohranu metrika, Grafana [4] za vizualizaciju, te Lokija [5] ili sličnih sustava za centralizirano upravljanje dnevničkih zapisa.

Telemetrija omogućuje i implementaciju naprednih funkcionalnosti, kao što su automatsko skaliranje resursa na temelju stvarnog opterećenja, detekcija anomalija, te izrada izvještaja o korištenju i performansama. U suvremenim okruženjima, telemetrija je često integrirana s ostalim komponentama sustava, omogućujući centralizirani pregled stanja i ubrzavajući reakciju na incidente.

Telemetrija implementirana zasebno ovisno o vrstama resursa te automatski dostupna prilikom postavljanja resursa predstavlja često veliku prednostrješenja u oblaku naspram ručnih postavljanja rješenja u vlastitoj infrastrukturi.

1.2. Pristupi oblikovanja sustava za upravljanje infrastrukturom

Sustavi za upravljanje infrastrukturom mogu se klasificirati prema stupnju automatizacije i razini kontrole koju nude korisniku. U ovom radu izdvajaju se dvije glavne kategorije:

neupravljeni sustavi i upravljeni sustavi. Ova podjela omogućuje jasnije razumijevanje razlika u pristupu upravljanju resursima, kao i identifikaciju prednosti i ograničenja svake skupine. Razumijevanje razlika između njih te proučavanje primjera dovodi nas do istraživanja potrebe za rješenjem opisanom u radu.

1.2.1. Neupravljeni sustavi

Neupravljeni sustavi (engl. *unmanaged systems*) predstavljaju pristup u kojem korisnik ima potpunu kontrolu nad infrastrukturom, ali i potpunu odgovornost za njezino postavljanje, održavanje i sigurnost. Ovakvi sustavi obično nude generičko sučelje za upravljanje resursima, bez specifične optimizacije ili automatizacije za pojedine vrste resursa. Ključna karakteristika neupravljenih sustava je minimalna razina apstrakcije – korisnik samostalno definira konfiguracije, prati performanse i brine o skaliranju i sigurnosti.

Primjeri neupravljenih sustava uključuju:

- Portainer [6] – grafičko sučelje za upravljanje Docker kontejnerima, koje omogućuje osnovne operacije nad kontejnerima, mrežama i volumenima, ali ne pruža napredne mogućnosti specifične za pojedine tipove aplikacija ili baza podataka.
- Dokku [7] – platforma za jednostavno postavljanje aplikacija na vlastitu infrastrukturu, koja uz pomoć proširenja (engl. *plugin*) omogućuje osnovnu integraciju baza podataka i drugih servisa, ali bez centralizirane telemetrije i naprednih konfiguracijskih sučelja.
- CapRover [8] – rješenje koje pojednostavljuje postavljanje i upravljanje kontejnerskim aplikacijama, ali sve resurse tretira na jednak način, bez specijaliziranih alata za pojedine tipove resursa i bez napredne telemetrije.
- DigitalOcean Droplets [9] i slični VPS servisi – korisnik samostalno postavlja i održava sve servise, bez dodatnih upravljačkih slojeva. Iako DigitalOcean nudi i upravljana rješenja, osnovni Droplet model spada u neupravljane sustave.

Neupravljeni sustavi često su privlačni naprednim korisnicima i timovima koji žele potpunu fleksibilnost i kontrolu nad infrastrukturom, ali zahtijevaju značajno tehničko znanje i dodatno vrijeme za održavanje.

1.2.2. Upravljeni sustavi

Upravljeni sustavi (engl. *managed systems*) predstavljaju pristup u kojem je infrastruktura unaprijed konfiguirana, optimizirana i automatizirana od strane pružatelja usluge ili specijaliziranog softverskog rješenja. Korisniku se omogućuje korištenje resursa bez potrebe za dubinskim tehničkim znanjem ili ručnim održavanjem, dok se operativni zadaci, sigurnost i skaliranje obavljaju automatski ili kroz centralizirano sučelje.

Najpoznatiji primjeri upravljenih sustava su veliki servisi u oblaku :

- *Amazon Web Services* (AWS), *Microsoft Azure*, *Google Cloud Platform* (GCP) – nude širok spektar upravljenih servisa, uključujući baze podataka (npr. *AWS RDS*), aplikacijske platforme (*App Service*, *App Engine*), posrednike poruka i druge specijalizirane resurse. Ovi sustavi integriraju napredne alate za telemetriju, automatsko skaliranje, sigurnosne politike i centralizirano upravljanje.
- *Vercel*, *Render* – platforme za automatsko postavljanje i skaliranje web-aplikacija, koje korisniku omogućuju jednostavno upravljanje bez potrebe za ručnim konfiguriranjem infrastrukture.
- *DigitalOcean Managed Databases* i slične usluge – omogućuju korištenje baza podataka bez brige o održavanju, sigurnosnim kopijama ili skaliranju.

Osim komercijalnih servisa u oblaku, dostupna su i *self-hosted* rješenja koja nastoje približiti funkcionalnosti upravljenih sustava korisnicima koji žele zadržati potpunu kontrolu nad vlastitom infrastrukturom:

- *Coolify* – suvremenii *self-hosted* sustav koji omogućuje postavljanje i upravljanje aplikacijama, bazama podataka i drugim resursima kroz centralizirano sučelje, uz podršku za automatsko skaliranje, telemetriju i integraciju s popularnim alatima.
 - Ovaj alat predstavlja jedno od najnaprednijih *self-hosted* rješenja, nudeći širok spektar upravljačkih mogućnosti za postavljanje resursa s kvalitetnim zadanim konfiguracijama
 - Postoje unaprijed definirani 'recepti' za razne sustave koji se mogu postaviti kao resursi, međutim oni ne sadrže specijalizirane pomoćne alate i sučelja prilagođena pojedinim vrstama resursa. Time je razina upravljanja i optimizacije za specifične resurse ipak ograničena u odnosu na vodeće komercijalne platforme.

- *cPanel* – jedno od najpoznatijih *self-hosted* rješenja za upravljanje web-hosting infrastrukturom. Iako cPanel omogućuje upravljanje web-aplikacijama, bazama podataka i e-mail servisima kroz grafičko sučelje, ima nekoliko značajnih ograničenja iz perspektive suvremenih potreba:
 - Visoka cijena licenciranja može predstavljati značajan trošak, osobito za manje projekte ili edukacijske svrhe.
 - Unatoč grafičkom sučelju, mnoge napredne funkcionalnosti zahtijevaju dodatnu ručnu konfiguraciju i intervencije..
 - Zastarjela arhitektura – cPanel ne podržava nativnu kontejnerizaciju, naprednu telemetriju niti suvremene metode automatizacije, što ga čini manje pogodnim za moderne aplikacijske arhitekture i dinamička okruženja.

Upravljeni sustavi omogućuju korisnicima da se fokusiraju na razvoj i održavanje aplikacija, dok se tehnički detalji vezani uz infrastrukturu automatiziraju i apstrahiraju. Većina takvih rješenja dolazi u obliku komercijalnih servisa u oblaku, koja sa sobom nose visoku cijenu u usporedbi s neupravljenim sustavima.

Zbog navedenih prednosti, ali i ograničenja postojećih rješenja, u sljedećem poglavlju bit će predstavljeno idejno rješenje sustava za upravljanje vlastitom infrastrukturom, koje kombinira prednosti upravljenih sustava s potpunom kontrolom nad resursima na vlastitom hardveru ili virtualnim poslužiteljima.

2. Idejno rješenje: sustav za upravljanje vlastitom infrastrukturom - Cloudy

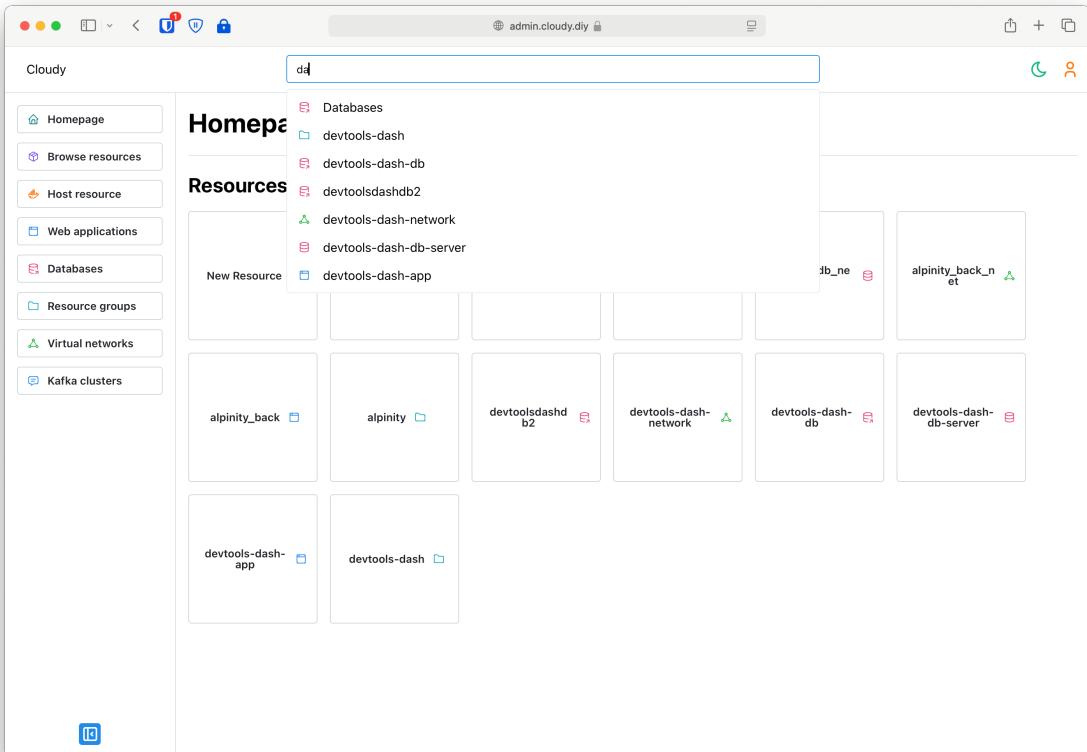
Kao što je istaknuto u prethodnom poglavlju, korištenje komercijalnih platformi u oblaku može biti financijski neisplativo za manje projekte, laboratorijske postavke ili edukacijske scenarije. S druge strane, ručna implementacija i održavanje infrastrukture na vlastitom hardveru često je prezahtjevna za timove s ograničenim resursima i zahtijeva angažman specijaliziranih *DevOps* inženjera, što dodatno povećava troškove.

Kao odgovor na navedene izazove, u sklopu ovog rada razvijen je sustav za upravljanje vlastitom infrastrukturom pod nazivom **Cloudy**. Osnovna ideja sustava je pružiti funkcionalnosti nalik onima na platformama u oblaku (engl. *cloud-like experience*), ali na vlastitom hardveru ili virtualnom privatnom poslužitelju, uz potpunu kontrolu nad podacima i troškovima.

Sustav Cloudy temelji se na nekoliko ključnih principa:

- **Upravljeni resursi** (engl. *Managed Resources*) – središnji koncept sustava je model „upravljanih resursa“. Umjesto da korisnik samostalno konfigurira svaki detalj, sustav nudi unaprijed definirane i optimizirane resurse. Cloudy se brine o pozadinskim procesima poput mrežne izolacije, sigurnosnih postavki i osnovne konfiguracije, čime korisniku isporučuje resurs spremjan za korištenje.
- **Jednostavnost i prilagođeno sučelje** – korisniku se ne izlaže kompleksnost Docker tehnologije i drugih alata koji se nalazi u pozadini. Umjesto toga, nudi mu se pojednostavljeni i intuitivno sučelje (engl. *curated access*) koje prikazuje samo relevantne opcije. Postavljeni su pametni predefinirani parametri (engl. *smart defaults*) koji odgovaraju većini scenarija, uz jednostavne alate za njihovu prilagodbu.
- **Integrirana telemetrija** – za razliku od neupravljenih sustava gdje je prikupljanje telemetrijskih podataka često naknadni i složen zadatak, u sustavu Cloudy telemetrija je automatski integrirana za svaki stvoreni resurs. Time se osigurava preglednost nad radom cijelog sustava bez dodatne konfiguracije.

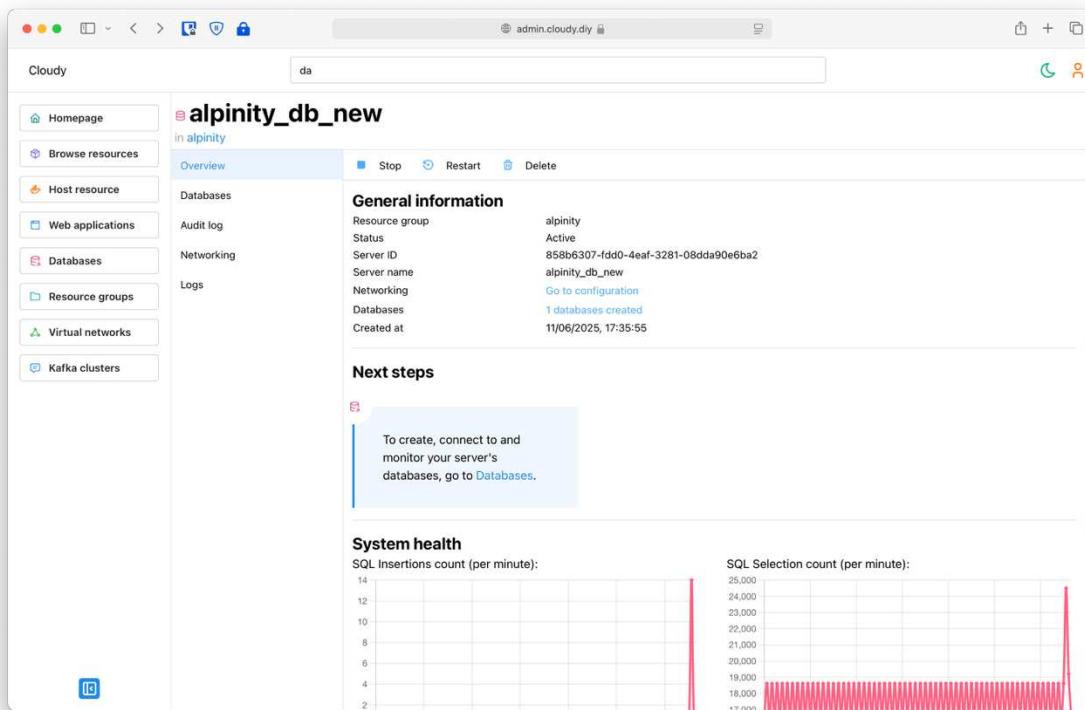
Sustav korisniku nudi moderno i pregledno grafičko sučelje, čija je početna stranica prikazana u nastavku (Slika 2.1)



Slika 2.1. Početna stranica upravljačkog panela sustava Cloudy

2.1. Resursi – funkcionalni zahtjevi

Glavni entitet sustava Cloudy je **resurs**. Resurs predstavlja bilo koju komponentu korisničkog sustava koja se postavlja i upravlja putem Cloudy sučelja, poput web-aplikacije, baze podataka ili dodatnih elemenata infrastrukturne arhitekture. Kroz upravljačko sučelje moguće je jednostavno stvarati, uređivati i uklanjati resurse. Svi resursi definiraju se unosom nekoliko osnovnih, nepromjenjivih podataka, nakon čega se otvara unificirano upravljačko sučelje prilagođeno tom tipu resursa. Iako svaka vrsta resursa nudi specifične funkcionalnosti, njihov raspored i osnovna struktura su generalizirani radi modularnosti i preglednosti (Slika 2.2).



Slika 2.2. Upravljačko sučelje Postgres poslužitelja

Kako bi se olakšalo upravljanje većim brojem resursa i omogućila organizacija prema projektima ili funkcionalnim cjelinama, sustav omogućuje grupiranje resursa putem entiteta **grupa resursa** (engl. *Resource group*). Grupa resursa omogućuje povezivanje više resursa koji pripadaju istoj aplikaciji, projektu ili korisničkoj domeni, čime se olakšava administracija i preglednost sustava (Slika 2.3).

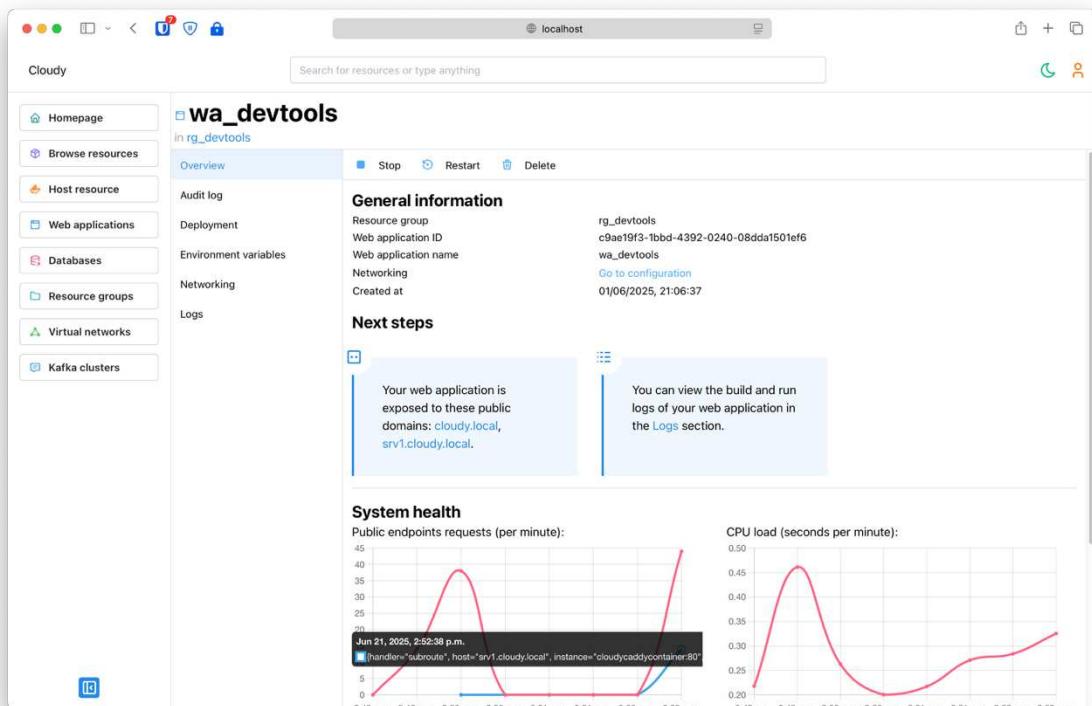
Slika 2.3. Prikaz grupe resursa u sučelju

2.1.1. Web-aplikacije

Web-aplikacije predstavljaju jedan od glavnih tipova resursa unutar svakog sustava za upravljanje infrastrukturom. Proces postavljanja nove web-aplikacije u sustavu Cloudy započinje ispunjavanjem forme za stvaranje resursa, u kojoj je potrebno odabrati grupu resursa kojoj aplikacija pripada, unijeti naziv resursa te odabrati metodu postavljanja (engl. *deployment method*). Trenutno je podržana metoda kloniranja javnog Git repozitorija (engl. *public Git clone*), pri čemu se unosi URL repozitorija iz kojeg će se aplikacija preuzeti i izgraditi.

Nakon stvaranja resursa, korisniku je dostupno upravljačko sučelje web-aplikacije koje je podijeljeno u nekoliko funkcionalnih panela:

- **Pregled** (engl. *Overview*) – prikazuje osnovne informacije o aplikaciji, uključujući naziv, grupu resursa i status (Slika 2.4). Osim toga, sadrži kartice s preporukama za sljedeće korake te odjeljak sa zdravljem sustava, gdje su prikazani grafovi metrika za zadnjih sat vremena. Ovdje su dostupne opće metrike poput opterećenja procesora i memorije, kao i specifična metrika, koju imaju samo resursi web-aplikacije, za broj zahtjeva prema aplikaciji putem interneta.



Slika 2.4. Pregled resursa web-aplikacije

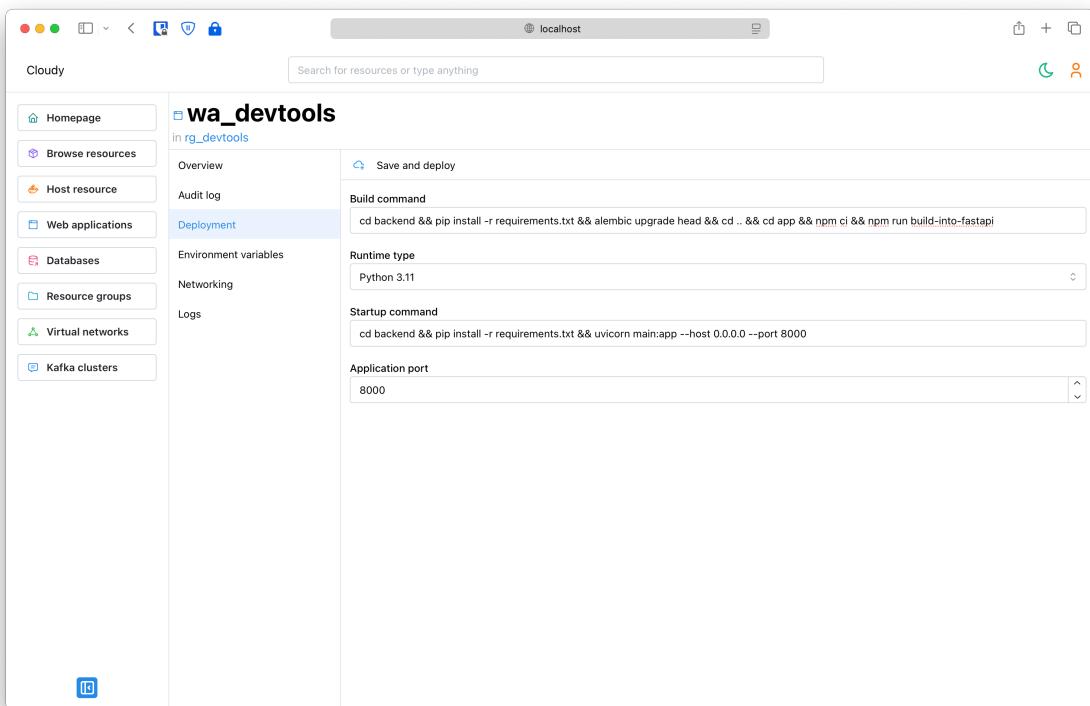
- **Dnevnik radnji** (engl. *audit log*) – omogućuje pregled svih akcija provedenih nad resursom kroz sučelje, uključujući opis, datum i vrijeme svake radnje (Slika 2.5). Ova funkcionalnost olakšava praćenje promjena i doprinosi transparentnosti upravljanja.

The screenshot shows a web-based management interface for a cloud service named 'Cloudy'. The URL in the address bar is 'localhost'. On the left, there's a sidebar with various navigation options: Homepage, Browse resources, Host resource, Web applications, Databases, Resource groups, Virtual networks, and Kafka clusters. The 'Audit log' option is selected. The main content area is titled 'wa_devtools' and shows an 'Overview' section with tabs for Deployment, Environment variables, Networking, and Logs. The 'Logs' tab is active, displaying a table of audit logs. The table has columns for Action, Timestamp, and a detailed log message. The logs show deployment attempts for the 'wa_devtools' application, with one entry failing due to a Docker API conflict.

| Action | Timestamp |
|--|----------------------|
| Successfully deployed git repo configuration to wa_devtools | 14/06/2025, 15:03:58 |
| Deploying git repo configuration to wa_devtools | 14/06/2025, 15:01:56 |
| Update deployment configuration for Web application wa_devtools | 14/06/2025, 15:01:56 |
| Failed to deploy git repo configuration to wa_devtools: Docker API responded with status code=Conflict, response={"message":"Conflict. The container name '/cloudy:9ae19f31bbd4392024008dd1501ef6containerbuilder' is already in use by container ('f114fa2c9301b224992565a430dbe94c0c2c32bb1ce3287c9bec3cef730d74)'. You have to remove (or rename) that container to be able to reuse that name."} | 14/06/2025, 14:59:37 |
| Deploying git repo configuration to wa_devtools | 14/06/2025, 14:59:30 |
| Update deployment configuration for Web application wa_devtools | 14/06/2025, 14:59:30 |
| Successfully deployed git repo configuration to wa_devtools | 14/06/2025, 14:58:25 |
| Deploying git repo configuration to wa_devtools | 14/06/2025, 14:57:48 |
| Update deployment configuration for Web application wa_devtools | 14/06/2025, 14:57:48 |
| Deploying git repo configuration to wa_devtools | 14/06/2025, 14:57:43 |
| Update deployment configuration for Web application wa_devtools | 14/06/2025, |

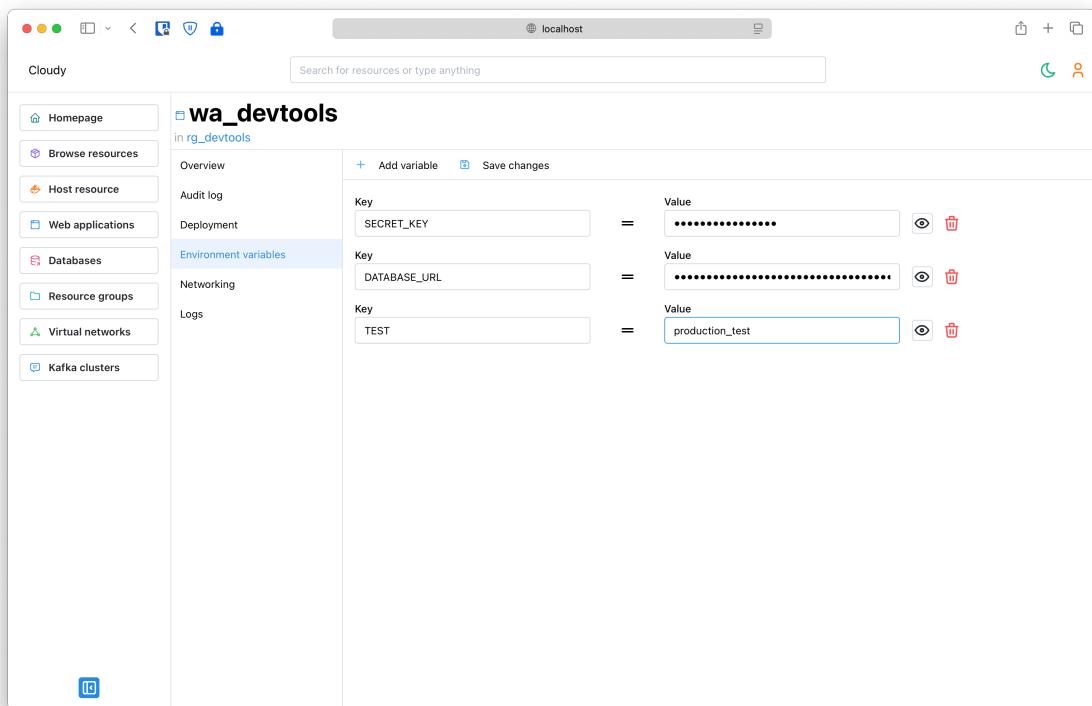
Slika 2.5. Dnevnik radnji web-aplikacije

- **Postavljanje** (engl. *Deployment*) – omogućuje definiranje i uređivanje naredbi za izgradnju (engl. *build*) i pokretanje (engl. *startup*) aplikacije (Slika 2.6). Također je moguće odabrati okruženje u kojem će se aplikacija izvoditi (Node, Python ili .NET), čime se osigurava fleksibilnost i proširivost sustava. Zadnje polje u ovom panelu odnosi se na aplikacijski port na kojem aplikacija očekuje dolazne zahtjeve. Na vrhu panela nalazi se tipka za spremanje i pokretanje procesa postavljanja (engl. *save and deploy*).



Slika 2.6. Postavljanje web-aplikacije

- **Varijable okruženja** (engl. *environment variables*) – omogućuje definiranje i uređivanje varijabli koje su dostupne tijekom izgradnje i izvođenja aplikacije (Slika 2.7).



Slika 2.7. Varijable okruženja web-aplikacije

- **Umrežavanje** (engl. *networking*) – sadrži pod-stranice za konfiguraciju javnih i privatnih mreža. Aplikacija može biti spojena na javnu mrežu preko više domena (Slika 2.8). Spajanjem na javnu mrežu, aplikacija se povezuje s *reverse proxy* komponentom, pri čemu se konfigurira prosljeđivanje određene domene na aplikaciju uz mogućnost aktivacije HTTPS protokola putem jednostavnog prekidača (Slika 2.9). U okviru privatnih virtualnih mreža, aplikacija se može povezati s resursom virtualne mreže radi komunikacije s drugim resursima unutar iste mreže (Slika 2.10).

The screenshot shows the Cloudy web interface with the URL `localhost`. On the left, there's a sidebar with various resource categories like Homepage, Browse resources, Host resource, Web applications, Databases, Resource groups, Virtual networks, and Kafka clusters. The main area is titled `wa_devtools` and shows the `Networking` section. It has a table with columns: Domain, HTTPS, and Id. Two entries are listed: `cloudy.local` and `srv1.cloudy.local`, both with 'HTTPS' set to 'No'. There are also tabs for 'Public access' and 'Private virtual networks', and a 'Logs' section at the bottom.

Slika 2.8. Lista spojenih javnih domena na web-aplikaciju

This screenshot shows the same Cloudy interface as above, but with a modal dialog open. The dialog is titled 'Add public network domain' and contains a single input field with the value 'test.cloudyyyyyy.diy'. Below the input field is a checkbox labeled 'Use HTTPS (will automatically generate and refresh certificates for you using Let's Encrypt)'. A warning message is displayed in a yellow box: 'You are currently accessing this interface over different IP than the one used by domain test.cloudyyyyyy.diy'. It also notes that this might be fine if you expect it (e.g., have a gateway, VPN, or proxy). At the bottom of the dialog is a blue 'Connect' button.

Slika 2.9. Spajanje web-aplikacije na internet

| Virtual network | Id |
|-----------------|--------------------------------------|
| vnet_devtools | cc5c0a25-d9c0-4903-0241-08dda1501ef6 |

Slika 2.10. Lista spojenih virtualnih mreža na web-aplikaciju

- **Dnevnički zapisi** (engl. *logs*) prikazuje listu logova aplikacije, za vrijeme izgradnje i za vrijeme rada, s mogućnošću automatskog osvježavanja (Slika 2.11). Ova mogućnost je posebno korisna tijekom izgradnje aplikacije za praćenje eventualnih pogrešaka, kao i za nadzor ponašanja aplikacije nakon postavljanja.

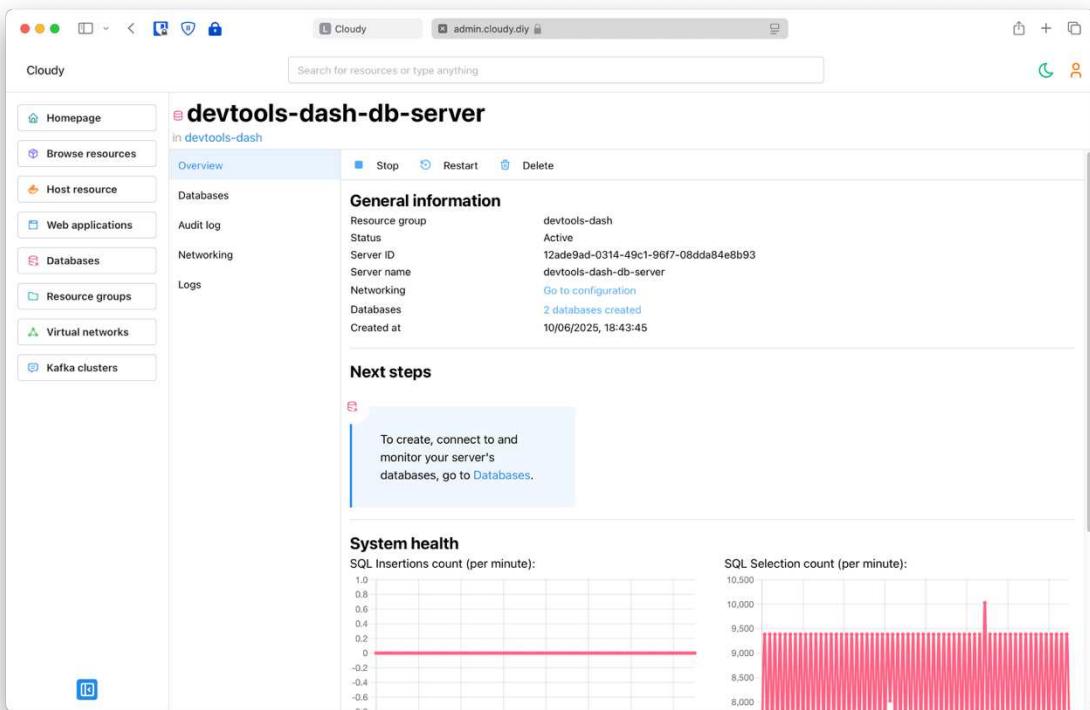
| Level | Message | Timestamp |
|---------|--|----------------------|
| UNKNOWN | access_token is None | 21/06/2025, 14:52:23 |
| INFO | INFO: 172.22.0.4:58484 - "GET /api/user/me HTTP/1.1" 401 Unauthorized | 21/06/2025, 14:52:23 |
| INFO | INFO: 172.22.0.4:58484 - "GET /openapi.json HTTP/1.1" 200 OK | 21/06/2025, 14:52:23 |
| INFO | INFO: 172.22.0.4:58484 - "GET /login HTTP/1.1" 200 OK | 21/06/2025, 14:52:23 |
| UNKNOWN | access_token is None | 21/06/2025, 14:52:22 |
| INFO | INFO: 172.22.0.4:58484 - "GET /api/user/me HTTP/1.1" 401 Unauthorized | 21/06/2025, 14:52:22 |
| UNKNOWN | access_token is None | 21/06/2025, 14:52:22 |
| INFO | INFO: 172.22.0.4:58484 - "GET /api/level/categories HTTP/1.1" 401 Unauthorized | 21/06/2025, 14:52:22 |
| INFO | INFO: 172.22.0.4:58484 - "GET /openapi.json HTTP/1.1" 200 OK | 21/06/2025, 14:52:22 |
| INFO | INFO: 172.22.0.4:58484 - "GET / HTTP/1.1" 200 OK | 21/06/2025, 14:52:22 |
| UNKNOWN | access_token is None | 21/06/2025, 14:52:20 |
| INFO | INFO: 172.22.0.4:33802 - "GET /api/user/me HTTP/1.1" 401 Unauthorized | 21/06/2025, 14:52:20 |
| INFO | INFO: 172.22.0.4:33802 - "GET /openapi.json HTTP/1.1" 200 OK | 21/06/2025, 14:52:20 |
| INFO | INFO: 172.22.0.4:33802 - "GET /login HTTP/1.1" 200 OK | 21/06/2025, 14:52:20 |
| UNKNOWN | access_token is None | 21/06/2025, 14:52:20 |
| INFO | INFO: 172.22.0.4:33802 - "GET /api/user/me HTTP/1.1" 401 Unauthorized | 21/06/2025, 14:52:20 |

Slika 2.11. Log lista web-aplikacije

2.1.2. Baze podataka

Za resurs baze podataka u sustavu Cloudy odabran je **PostgreSQL** kao najpopularniji sustav baza podataka otvorenog koda, uz mogućnost proširenja sustava na druge baze poput Microsoft SQL Servera, MySQLa i slične. Upravljanje bazama podataka u sustavu podijeljeno je na dva tipa resursa: **poslužitelj baze podataka** (engl. *Database Server*) i **baza podataka** (engl. *Database*).

Prilikom stvaranja nove baze podataka korisnik može odabrati postojeći poslužitelj ili kreirati novi. Korištenje istog poslužitelja za više baza omogućuje dijeljenje hardverskih resursa, metrika i mrežnih postavki u jednostavnijim scenarijima, dok je za kompleksnije scenarije moguće odvojiti svaku bazu na vlastiti poslužitelj. Prilikom kreiranja poslužitelja, potrebno je unijeti naziv i grupu resursa. Za svaki poslužitelj automatski se generira administratorski korisnik kojem krajnji korisnik nema pristup, a koji služi za interno upravljanje i sigurnost.



Slika 2.12. Upravljačko sučelje PostgreSQL poslužitelja

Pregled poslužitelja baze podataka (Slika 2.12) sadrži:

- **Pregled** – Proširen sa metrikama specifičnim za baze podataka kao što su broj pročitanih/zapisanih SQL redova

- **Lista baza podataka** – prikazuje baze podataka stvorene na poslužitelju te nudi mogućnost dodavanja novih (Slika 2.13)
- **Dnevnik radnji**
- **Umrežavanje**
- **Logovi**

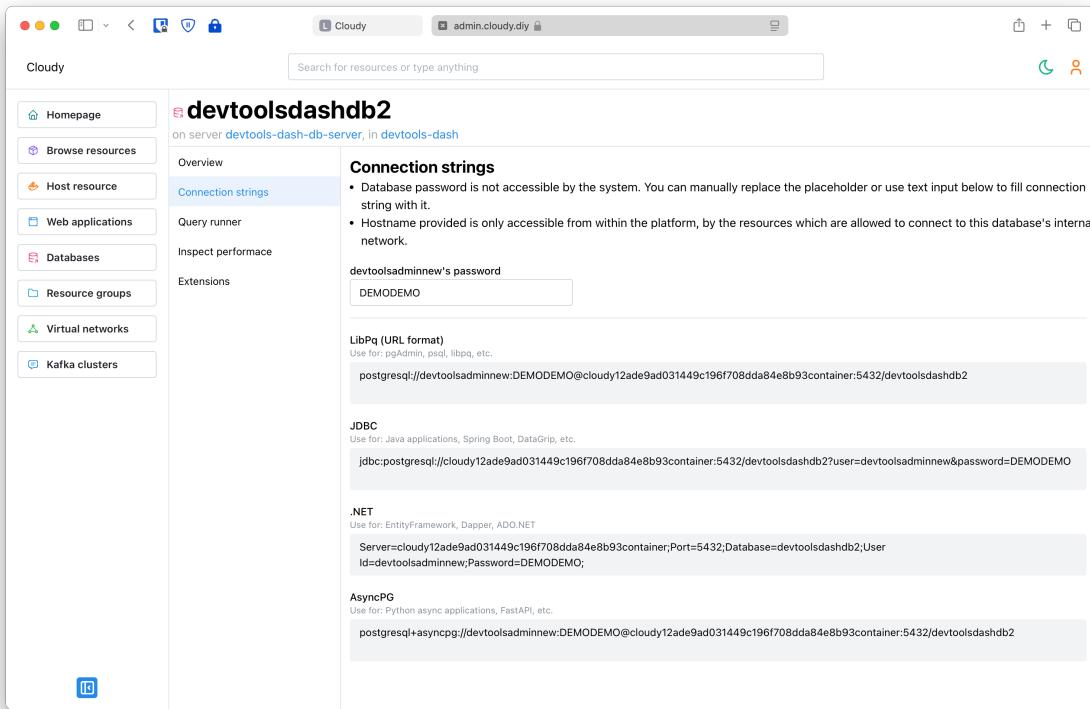
| Database name | Admin username | Created at | Updated at |
|------------------|------------------|----------------------|----------------------|
| devtools-dash-db | devtoolsadmin | 10/06/2025, 20:45:17 | 10/06/2025, 20:45:17 |
| devtoolsdashdb2 | devtoolsadminnew | 10/06/2025, 21:17:49 | 10/06/2025, 21:17:49 |

Slika 2.13. Pregled baza podataka na poslužitelju

Ove funkcionalnosti su u osnovi jednake onima dostupnima za resurse web-aplikacija, uz iznimku specifičnih opcija za baze podataka.

Pregled pojedine baze podataka proširen je dodatnim funkcionalnostima:

- Tekstualni nizovi za povezivanje – stranica na kojoj su pripremljeni tekstualni nizovi za povezivanje (engl. *connection string*) s bazom podataka za različite biblioteke i programske jezike. Nizovi ne sadrže lozinku administratora koju je korisnik definirao, ali ako ju korisnik želi dodati bez da ručno mijenja niz, može to učiniti unosom šifre u polje na vrhu stranice (Slika 2.14).



Slika 2.14. Tekstualni nizovi za povezivanje na bazu podataka

- **Izvođač upita (engl. Query runner)** – alat za izravnu interakciju s bazom podataka kroz sučelje sustava, bez potrebe za otvaranjem baze prema internetu (Slika 2.15). Komunikacija se ostvaruje posredno, preko sustava, korištenjem *Docker exec* nad odgovarajućim kontejnerom poslužitelja. Cilj izvođača upita nije da zamijeni potpuni alat za upravljanje bazom podataka, nego za brzu provjeru stanja podataka ili otklanjanje pogrešaka. Izvođač upita se spaja na bazu kao interni glavni administrator te zatim smanjuje svoje ovlasti predstavljanjem kao administrator baze. Uvedeno je pročišćavanje upita kako bi se izbjeglo podizanje ovlasti.

The screenshot shows the Cloudy UI interface. On the left is a sidebar with various navigation options: Homepage, Browse resources, Host resource, Web applications, Databases, Resource groups, Virtual networks, and Kafka clusters. The 'Databases' option is selected. The main area has a title 'devtoolsdashdb2' and a subtitle 'on server devtools-dash-db-server, in devtools-dash'. Below this is a 'Query runner' section with a code editor containing a SQL query:

```
1 -- Write your SQL query here --
2 -- example for viewing all tables:
3 SELECT * FROM pg_tables WHERE schemaname = 'devtoolsdashdb2'
```

Below the code editor is a table with the following data:

| schemaname | tablename | tableowner | tablespace | hasindexes | hasrules | hastriggers | rowsecurity |
|-----------------|-----------------|------------------|------------|------------|----------|-------------|-------------|
| devtoolsdashdb2 | alembic_version | devtoolsadminnew | | t | f | f | f |
| devtoolsdashdb2 | user | devtoolsadminnew | | t | f | t | f |
| devtoolsdashdb2 | level_session | devtoolsadminnew | | t | f | t | f |

Slika 2.15. Izvođač upita nad bazom podataka

- **Pregled performansi (engl. Inspect performance)** – stranica koja prikazuje najvažnije statistike za bazu podataka, poput liste najsporijih upita, najvećih upita, upita koji najviše koriste disk zbog nedostatka dobrih indeksa i slično (Slika 2.16).

devtoolsdashdb2
on server devtools-dash-db-server, in devtools-dash

| query | executions | disk_reads | cache_hits | avg_execution_ms | total_execution_ms |
|---|------------|------------|------------|------------------|--------------------|
| SELECT query, calls AS executions, cast(mean_exec_time as DECIMAL(18, 2)) AS avg_execution_ms, cast(total_exec_time as DECIMAL(18, 2)) AS total_execution_ms FROM public.pg_stat_statements WHERE queryid is not null ORDER BY total_exec_time DESC LIMIT \$1 | 1 | 2 | 3 | 3.23 | 3.23 |
| INSERT INTO alembic_version (version_num) VALUES (\$1) RETURNING alembic_version.version_num | 1 | 1 | 3 | 0.42 | 0.42 |
| UPDATE "user" SET last_login=\$1::TIMESTAMP WITHOUT TIME ZONE WHERE "user".id = \$2::INTEGER | 33 | 0 | 173 | 0.40 | 13.24 |
| INSERT INTO "user" (nickname, password_hash, last_login) VALUES (\$1::VARCHAR, \$2::VARCHAR, \$3::TIMESTAMP WITHOUT TIME ZONE) RETURNING "user".id | 41 | 0 | 290 | 0.36 | 14.60 |
| UPDATE level_session SET try_count=\$1::INTEGER WHERE level_session.id = \$2::INTEGER | 236 | 0 | 1255 | 0.19 | 44.25 |

Slika 2.16. Pregled performansi baze podataka

- **Ekstenzije (engl. Extensions)** – stranica za pregled i dodavanje ekstenzija u PostgreSQL bazu podataka kroz grafičko sučelje. Poslužitelj baze podataka dolazi s nekoliko najpopularnijih ekstenzija (*postgis, vector*) unaprijed instaliranih, koje se mogu aktivirati jednim klikom (Slika 2.17).

alpinity_psql_new
on server alpinity_db_new, in alpinity

| | |
|--------------------------------|-------|
| postgis_tiger_geocoder | |
| address_standardizer | |
| address_standardizer_data_us_3 | |
| address_standardizer-3 | |
| postgis_raster | |
| postgis | 3.6.3 |
| postgis_topology-3 | |
| postgis_tiger_geocoder- | |
| vector | |
| address_standardizer_data_us | |

Slika 2.17. Upravljanje ekstenzijama baze podataka

Važno je napomenuti da je u trenutnoj fazi razvoja sustava mrežna funkcionalnost za baze podataka ograničena zbog ograničenja implementacije *reverse proxy* sustava isključivo na na protokol HTTP(S), dok otvaranje baze prema internetu zahtjeva običan TCP/UDP promet. Baze podataka moguće je za sada povezivati isključivo s internim servisima putem virtualnih mreža.

Prilikom kreiranja baze podataka korisnik definira naziv baze, kao i korisničko ime i lozinku administratora te baze. Nakon stvaranja, korisnik je preusmjeren na pogled koji sadrži navedene alate.

2.1.3. Posrednici poruka

Sustav Cloudy nudi mogućnost postavljanja i upravljanja *Apache Kafka* klasterom kao resursom za posredovanje porukama?. Trenutna implementacija koristi pojednostavljenu konfiguraciju s jednim kombiniranim brokerom i kontrolerom, što omogućuje stabilan rad uz minimalne zahtjeve za konfiguracijom korisnika.

Istraživanjem provedenim u sklopu ovog rada razmatrala se složenija arhitektura s autentifikacijom, odvojenim kontrolerom i skaliranim brojem Kafka kontejnera. Zbog složenosti implementacije prilikom prototipiranja, odabrana je jednostavnija verzija jednog kombiniranog brokera koja ne zahtijeva dodatnu konfiguraciju pristupa, a teme se automatski kreiraju prema potrebi (*auto-create topics*).

Prilikom stvaranja Kafka resursa potrebno je unijeti samo naziv i grupu resursa, dok ostale postavke automatski konfigurira sustav.

Kao i kod ostalih resursa, Kafka resurs nudi standardne funkcionalnosti poput pregleda, dnevnika radnji, umrežavanja i dnevničkih zapisa. Zbog TCP protokola koji Kafka koristi, umrežavanje je ograničeno na virtualne mreže unutar sustava.

Posebna funkcionalnost Kafka resursa je upravljanje temama (engl. *topics*) (Slika 2.18). Korisnik može pregledavati listu trenutno dostupnih tema, otvoriti pojedinu temu za detaljan pregled te izravno iz preglednika slušati poruke na toj temi (Slika 2.19). Također je omogućeno slanje poruka na otvorenu temu putem sučelja.

| Name | Topic ID | Partition count | Replication factor |
|-----------|----------|-----------------|--------------------|
| demotopic | | 1 | 1 |
| tema1 | | 1 | 1 |

Slika 2.18. Pregled Kafka tema

```

Selected Message
{
  "name": "sample-1-4k-square.png",
  "isFolder": false,
  "childCount": 0,
  "createdAt": "2024-12-10T12:32:03Z",
  "lastModified": "2024-12-10T12:32:12Z",
  "size": 13735317,
  "path":
    "core/tenantUsers/1/sample-1-4k-square.png",
  "children": []
}
  
```

Slika 2.19. Pregled i slušanje na temu

Iako je trenutna implementacija sustava dovoljna za neke manje laboratorijske scenarije i manja ili testna okruženja, kao prilika za budući razvoj ističe se proširenje klastera na više od jednog brokera, te autentifikacija i autorizacija koji bi omogućili preciznu kontrolu pristupa i povećanu sigurnost.

2.1.4. Virtualne mreže

Virtualne mreže omogućuju sigurnu i fleksibilnu komunikaciju između različitih resursa unutar sustava bez potrebe za izlaganjem tih resursa javnoj mreži. U sustavu Cloudy, stvaranje virtualne mreže zahtijeva unos imena mreže i pripadajuće grupe resursa.

Kroz panel za umrežavanje svakog resursa moguće ga je jednostavno dodijeliti jednoj ili više virtualnih mreža. Resursi koji se nalaze unutar iste virtualne mreže mogu međusobno komunicirati koristeći svoja unutarnja imena domaćina (engl. *hostname*), što omogućuje sigurno povezivanje, primjerice, web-aplikacije i baze podataka bez potrebe za otvaranjem baze prema internetu. Ovakav pristup posebno je koristan u scenarijima gdje je potrebno omogućiti komunikaciju između mikroservisa isključivo putem privatnih mreža, dok se pristup prema vanjskom svijetu ostvaruje kroz pristupne (engl. *gateway*) servise.

Resurs virtualne mreže nudi pregled svih resursa koji su joj pridruženi, čime se olakšava administracija i praćenje mrežnih veza (Slika 2.20). Uz to, dostupan je dnevnik radnji koji omogućuje praćenje svih promjena i aktivnosti vezanih uz njeno umrežavanje.

Osim resursa virtualne mreže, sustav po potrebi stvara dodatne pažljivo definirane mreže nevidljive korisniku sustava namijenjene za nesmetan rad i proširivanje funkcionalnosti sustava.

Slika 2.20. Pregled virtualne mreže i pridruženih resursa

2.2. Telemetrija

Telemetrija predstavlja jedan od ključnih elemenata suvremenih sustava za upravljanje infrastrukturom, te je jedna od najvažnijih prednosti opisanog sustava u odnosu na dosad dostupna rješenja. Povećana integracija telemetrije s konceptom upravljanja resursa omogućuje detaljan i sveobuhvatan uvid u rad svih dijelova sustava. Cilj sustava je omogućiti da svaki dio infrastrukture može biti promatran i nadziran bez potrebe za dodatnim alatima ili konfiguracijama, čak i kada su u pitanju metrike koje su specifično vezane uz tip resursa.

2.2.1. Agregacija telemetrijskih podataka

Svi servisi koji čine resurse u sustavu izvode se unutar kontejnera, što omogućuje automatsku agregaciju dnevničkih zapisa i metrika. Dnevnički zapisi svakog resursa prikupljaju se centralizirano i dostupni su za pregled kroz upravljačko sučelje na konzistentan način, bez potrebe za ručnim pristupom pojedinim kontejnerima ili dodatnim alatima.

Osim osnovnih dnevničkih zapisa, sustav kontinuirano prikuplja hardverske metrike poput iskorištenosti procesora, zauzeća memorije, mrežnog prometa i korištenja diska. Ove informacije prikazuju se u pregledima resursa, a dodatne, detaljnije metrike prikupljaju se ovisno o tipu resursa. Primjerice, za baze podataka prate se operacije čitanja i pisanja, za posrednike poruka poput Kafka brojevi članova u grupama potrošača i pomaci grupa, dok se za web-aplikacije bilježi broj pristupa po domeni.

Agregacija svih ovih podataka odvija se automatski, čime se korisniku omogućuje potpuna preglednost i nadzor bez dodatnih konfiguracija.

2.2.2. Prikaz i analiza telemetrijskih podataka

Pregled telemetrijskih podataka u sustavu omogućen je kroz dva osnovna pristupa. S jedne strane, najvažniji dnevnički zapisi i osnovne metrike dostupni su izravno u upravljačkim panelima svakog resursa, što omogućuje brz uvid u stanje i performanse bez napuštanja sučelja.

Za detaljniju analizu i napredni nadzor, sustav nudi mogućnost jednostavnog postavljanja alata **Grafana**. Grafana je odmah povezana s izvorima dnevničkih zapisa i metrika, te omogućuje korisnicima pregled širokog spektra podataka, izradu prilagođenih nadzornih ploča i praćenje stanja sustava u realnom vremenu.

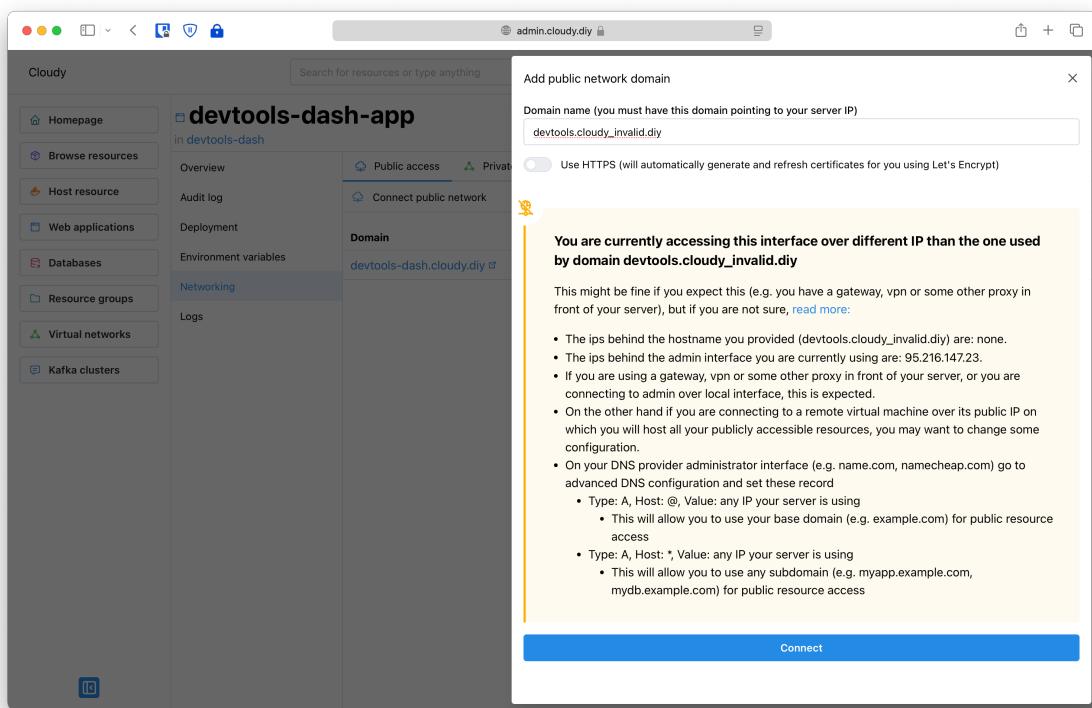
Ovakva integracija je odabrana zbog vrlo velike količine različitih metrika i oslanjanja na postojeće industrijske standarde za telemetriju, kako bi se korisniku omogućio pristup i rad s njima kada su potrebni, a zadržavanje jednostavnosti kada nisu.

2.3. Otvaranje prema internetu

Otvaranje sustava prema internetu podrazumijeva omogućavanje korisnicima sustava ili postavljenih resursa pristup putem interneta na unificiran način, te bi bez ove komponente cijeli sustav bio izoliran unutar lokalne mreže jednog računala. U sustavu Cloudy, prilikom prvog pokretanja automatski se postavlja *reverse proxy* koji služi kao ulazna točka za sav internetski promet prema sustavu. Ova komponenta upravlja pristupom upravljačkom panelu, web-aplikacijama i integriranoj Grafani, osiguravajući centraliziranu kontrolu prometa.

Sustav omogućuje jednostavno postavljanje i održavanje HTTPS certifikata jednim klikom prilikom spajanja resursa na mrežu. Time se korisniku olakšava implementacija sigurnih veza bez potrebe za ručnim generiranjem ili konfiguiranjem TLS/SSL certifikata.

Za povezivanje resursa s internetom dovoljno je unijeti željenu (pod) domenu koju korisnik želi povezati s određenim resursom. Očekuje se da su DNS zapisi za odabranu domenu prethodno postavljeni tako da upućuju na IP adresu sustava. Kako bi se pojednostavio ovaj postupak i smanjila mogućnost pogreške, sustav automatski provjerava pokazuje li unesena domena na istu IP adresu kao ona kojom se pristupa upravljačkom panelu. Ako je provjera uspješna, HTTPS se automatski postavlja kao preporučena značajka. U slučaju neuspješne provjere, HTTPS se isključuje, a korisniku se prikazuje informacija o rizicima povezanim s nepravilno konfiguiranom domenom, uz jasne upute o DNS zapisima koje je potrebno postaviti kako bi sustav ispravno funkcionirao (Slika 2.21).



Slika 2.21. Uputa za postavljanje DNS zapisa domene

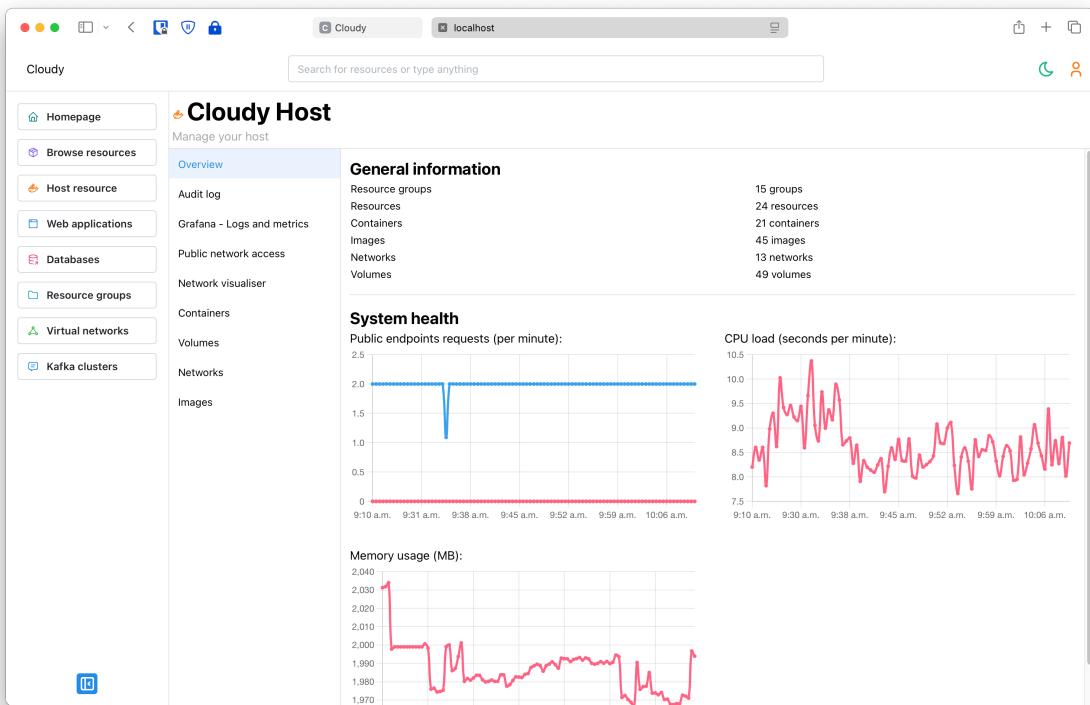
Ovakav pristup omogućuje korisnicima sigurno, jednostavno i pouzdano otvaranje resursa prema internetu, uz minimalan broj koraka i bez potrebe za dodatnim tehničkim znanjem ili vanjskim alatima.

2.4. Virtualni resurs domaćina

Do sada su opisani resursi koji se mogu postaviti i kojima se može upravljati kroz sustav, no za cijelovito upravljanje infrastrukturom potrebno je omogućiti i nadzor samog sustava, odnosno poslužitelja na kojem Cloudy radi. Kako bi i ovo upravljanje bilo dosljedno ostalim dijelovima sustava, domaćin (engl. *host*) je predstavljen kao „virtualni“ resurs. Ovaj resurs nije moguće stvoriti niti je dio modela podataka, već se prikazuje na listi resursa kao fiksni, uvijek prisutan entitet koji unutar sebe sadrži korisne kontrole i informacije o domaćinu.

Prikaz virtualnog resursa domaćina strukturiran je na način sličan ostalim resursima:

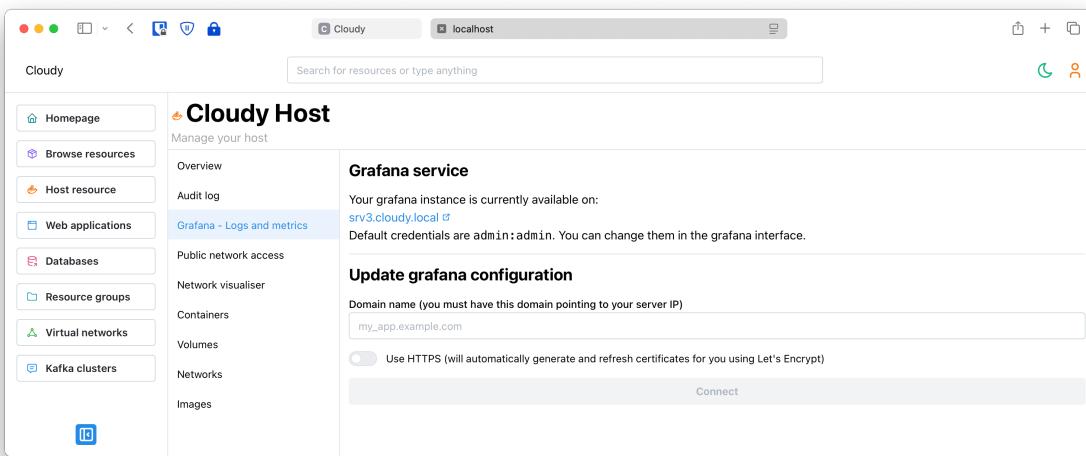
- **Pregled** - prikazuje osnovne informacije o sustavu, uključujući broj resursa, broj grupa resursa, broj kontejnera, mreža i drugih entiteta sustava (Slika 2.22). U ovom pregledu dostupne su i globalne metrike, poput ukupnog broja zahtjeva putem interneta, globalnog stanja memorije i procesora.



Slika 2.22. Pregled virtualnog resursa domaćina

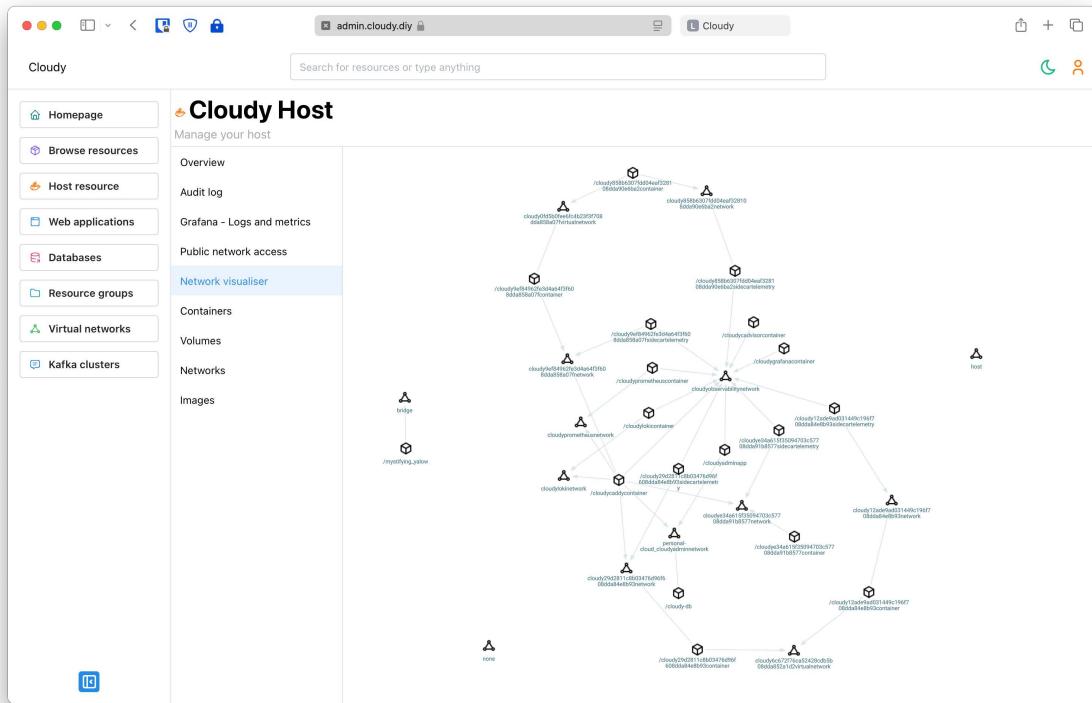
- **Dnevnik radnji** - sadrži agregiranu listu svih radnji provedenih nad sustavom, uz prikaz identifikatora resursa na kojem je radnja zabilježena. Time se omogućuje centralizirano praćenje aktivnosti u cijelom sustavu.

- **Grafana – dnevnički zapisi i metrike** - ova podstranica omogućuje postavljanje Grafane unosom podataka za pristup putem javnog interneta (domena i HTTPS opcija) (Slika 2.23). Prilikom spremanja, Grafana se postavlja ili, ako već postoji, premješta na unesenu domenu. Detalji o ovoj funkcionalnosti opisani su u prethodnom poglavlju. Ako je Grafana već postavljena, na vrhu stranice nalazi se poveznica na nju zajedno s uputama za prepostavljene vjerodajnice (engl. *credentials*).



Slika 2.23. Konfiguriranje alata Grafana

- **Javni mrežni pristup** – služi za uređivanje postavki javnog pristupa prema upravljačkom panelu sustava. Zadano svaka domena koja nije zauzeta (ili izravna IP adresa) vodi prema panelu putem HTTP protokola, što omogućuje prvi pristup nakon postavljanja sustava. Radi povećanja sigurnosti, ovdje se može definirati domena preko koje će se omogućiti pristup panelu putem HTTPS protokola.
- **Vizualizator mreže** - Prikazuje graf kontejnera i mreža te njihovih međusobnih poveznica unutar sustava (Slika 2.24), čime se omogućuje generalni pregled infrastrukture i lakše razumijevanje odnosa između resursa .



Slika 2.24. Vizualizator mreže sustava

- **Kontejneri, volumeni, mreže, slike** - ove četiri podstranice pružaju pregled trenutno stvorenih entiteta u Docker okruženju koje sustav koristi, zajedno s osnovnim informacijama o njihovom statusu (Slika 2.25). Ovi pregledi služe kao pomoć u dijagnostici i otklanjanju pogrešaka, omogućujući korisniku uvid u najvažnije tehničke detalje izravno kroz sučelje sustava.

The screenshot shows the Cloudy Host interface with the 'Containers' tab selected. A table lists the running Docker containers. The columns are: Name, Container ID, Running, Paused, and Restarting. The table contains four rows of data:

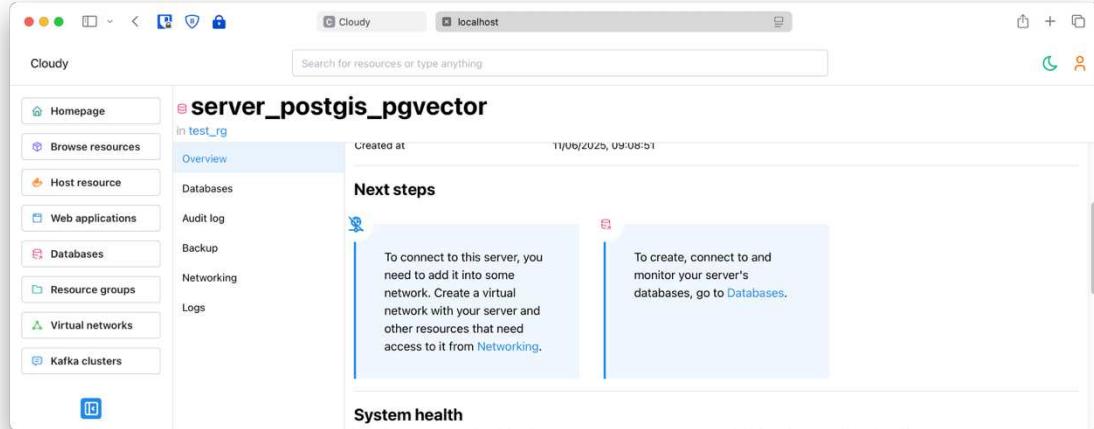
| Name | Container ID | Running | Paused | Restarting |
|--|--|---------|--------|------------|
| /cloudy9ef84962fe3d4a64f3f608dda858a07fcontainer | 5f464dff40fc4eabec4507dedabbabaac22a2c09b48a014d8c5dd03a402b804c | Yes | No | No |
| /cloudy34a615f35094703c57708dda91b8577container | 4acdba2fdf4ecacadd40a2a55d5e209e6918bf215822c73a9bec46957fc45b | Yes | No | No |
| /cloudy29d281c8b03476d96f608da84e8b93container | 3108cd60fcecc0999ca7c86cc8a1ef9b2c8133d1b8bd13bd146bfa1b82cd1e | Yes | No | No |

Slika 2.25. Prikaz kontejnera u sustavu

2.5. Ostali zahtjevi

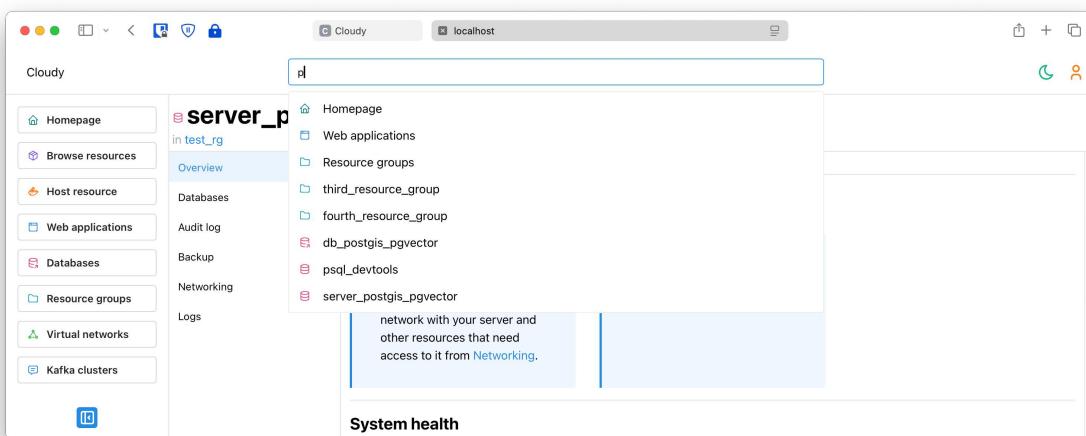
Osim opisanih glavnih zahtjeva sustava, postoje mnogi manji zahtjevi koji doprinose kvaliteti korištenja sustava u obliku jednostavnosti ili sigurnosti upravljanja sustavom.

- **Jednostavnost postavljanja samog sustava** - kako bi sustav u potpunosti ispunio svoj cilj pomoći korisnicima i timovima s manjim iskustvom u upravljanju infrastrukturom, nužna je jednostavnost postavljanja. Osim što njime upravlja, sustav Cloudy također se pokreće unutar *Docker* okruženja, što omogućuje da se cijeli sustav postavi instalacijom *Docker enginea* s repozitorija za macOS ili Linux distribucije poslužitelja kojeg korisnik koristi, kloniranjem repozitorija sustava te pokretanjem pomoću *Docker compose* komande. Idealno, ovaj postupak traje svega nekoliko minuta.
- **Jednostavnost postavljanja resursa** - forme za stvaranje resursa dizajnirane su tako da od korisnika traže minimalne potrebne podatke, uz dobro definirane zadane konfiguracije. Time se omogućuje postavljanje resursa bez potrebe za razumijevanjem tehnologija koje se nalaze ispod apstrakcije sustava.
- **Pomoć u upravljanju postavljenim resursima kroz preporuke za sljedeće korake** - različiti resursi u svojim pregledima sadrže odjeljak sljedeći koraci (engl. *next steps*) koji korisniku pruža statične i dinamične preporuke. Statični koraci predstavljaju najvažnije načine korištenja sustava, dok se dinamični pojavljuju ili nestaju ovisno o trenutnoj konfiguraciji resursa. Primjeri uključuju upute za umrežavanje poslužitelja baze podataka u virtualnu mrežu ako nije spojen (Slika 2.26), preporuke za definiranje naredbi za izgradnju i pokretanje web-aplikacije i slično. Ove preporuke nastale su na temelju povratnih informacija korisnika tijekom nadziranog testiranja, gdje su korisnici istaknuli potrebu za jasnim smjernicama u postavljanju aplikacija.



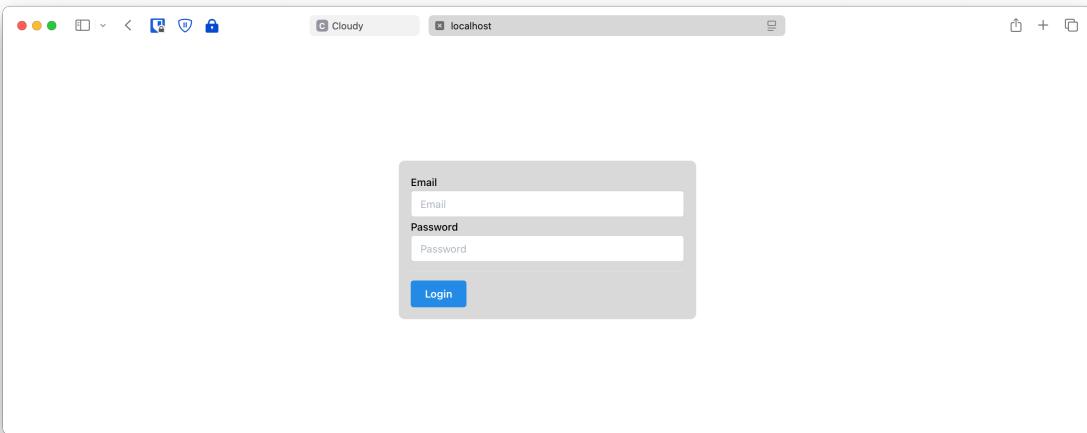
Slika 2.26. Uputa za umrežavanje Postgres poslužitelja baze podataka

- **Globalna tražilica** - sustav sadrži globalnu tražilicu koja omogućuje pretraživanje bilo kojeg resursa ili stranice unutar sustava (Slika 2.27), čime se olakšava navigacija i ubrzava pronađazak potrebnih informacija.



Slika 2.27. Globalna tražilica sustava

- **Autentifikacija** - budući da je upravljački panel dostupan putem interneta, autentifikacija je neizostavan dio sustava. Za bilo kakvu interakciju sa sustavom potrebno je imati autenticiranog korisnika (Slika 2.28). Iako autorizacija po ulogama nije implementirana zbog vremenskih ograničenja tijekom izrade rada, njeni bi implementaciji omogućila višekorisnički rad s različitim privilegijama, te se ističe kao prilika za budući razvoj.



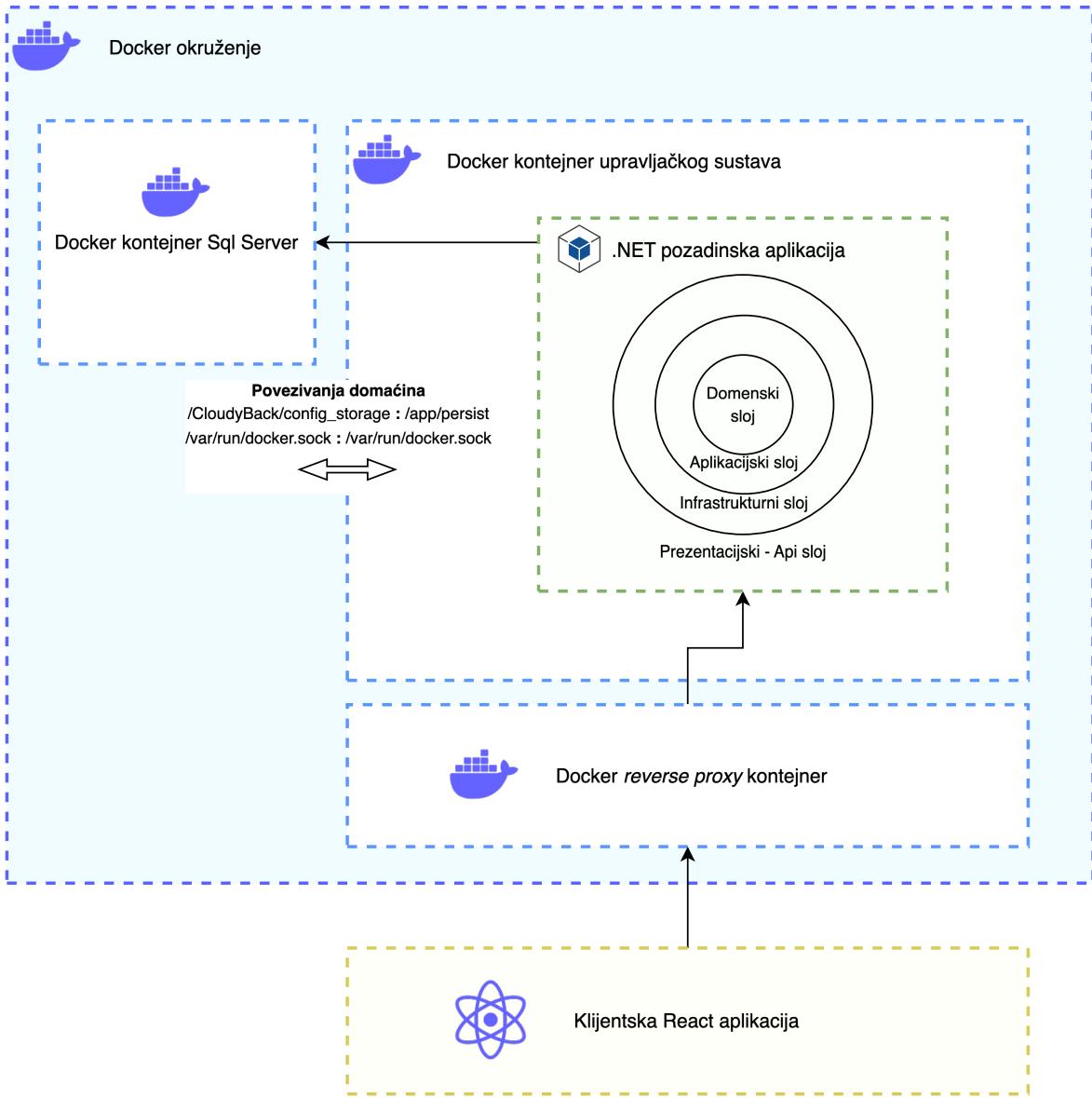
Slika 2.28. Forma za prijavu u sustav

- **Dnevnik radnji** - kao i većina većih sustava za upravljanje infrastrukturom, opisani sustav sadrži nepromjenjivi dnevnik radnji koji služi za provjeru što se i kada dogodilo u sustavu. U budućnosti, uz implementaciju autorizacije, dnevnik radnji može se koristiti za nadzor korisničkih aktivnosti i sprječavanje zlouporaba.

3. Implementacija

3.1. Arhitektura sustava

Opisani sustav ima standardnu klijentsko poslužiteljsku arhitekturu, gdje se na klijentskom računalu nalazi jednostranična web-aplikacija za upravljanje koja komunicira sa pozadinskim REST API-jem. Ova povezanost je proširena time da poslužiteljska strana sustava živi unutar Docker okruženja, kojeg kontrolira preko Docker API-ja proslijeđenog priklučka. Poslužiteljskoj strani klijent pristupa preko *reverse proxy* među-kontejnera koja otvara sustav kao i sve postavljene resurse prema internetu (Slika 3.1). Poslužiteljska strana sustava Cloudy strukturirana je po uzoru na arhitekturu lukovice, gdje su u sredini domenski entiteti. Iznad njih je aplikacijski sloj koji sadrži sučelja korištena inverzijom ovisnosti za komunikacijom sa vanjskim dijelovima infrastrukture. Iznad aplikacijskog sloja nalazi se infrastruktura u kojoj se nalaze klijenti za komunikaciju s Docker API-jem pomoću *Docker.DotNet* biblioteke kao i repozitoriji za pristup podacima baze podataka pomoću *Entity Framework* biblioteke. Na vrhu je prezentacijski sloj u obliku web API-ja na kojeg se spaja krajnja klijentska aplikacija.

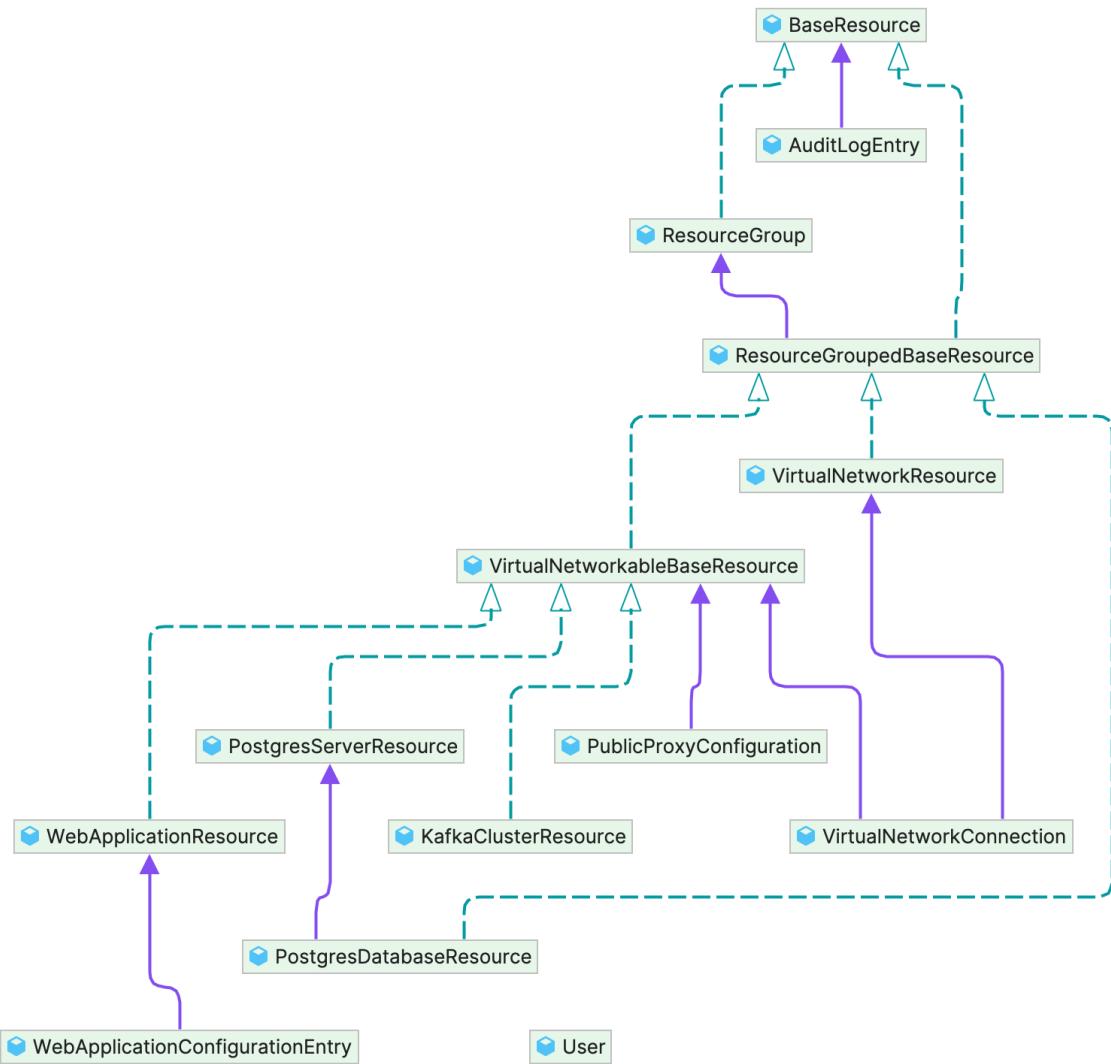


Slika 3.1. Pregled arhitekture sustava

3.2. Pregled modela podataka

Podaci u sustavu modelirani su objektno orijentiranim pristupom. Glavni tip podatka je bazni resurs, koji može biti ili grupa resursa ili neki konkretni grupirani resurs. Grupirani resursi mogu biti web-aplikacija, poslužitelj baze podataka, baza podataka, Kafka klaster i virtualna mreža. Osim toga neki od grupiranih resursa mogu spadati u resurse umrežavane virtualnim mrežama. Opisana struktura izvedena je klasnim nasljeđivanjem, što omogućuje stvaranje poveznica na generičke resurse. Na primjer, grupa resursa se sastoji od liste grupiranih resursa umjesto po jedne liste svakog tipa resursa. Sam po sebi ovaj model podataka je uobičajen, međutim posebnost u ovom slučaju dolazi od toga da se isti preslikava na model

baze podataka. Zbog tog razloga, za mnoge poveznice se ne koriste strani ključevi nego samo indeksirana polja za identifikatore. Detalji o načinu pretvorbe u model baze su opisani u sljedećim poglavljima. Dijagram modela podataka, sa označenim nasljeđivanjima i povratnim tipovima vidljiv je na slici u nastavku (Slika 3.2).



Slika 3.2. Model podataka sustava Cloudy

3.3. Korišteni razvojni okviri i biblioteke

Implementacija sustava Cloudy temelji se na tehnologijama .NET ekosustava, koje omogućuju modularnost, održivost i visoke performanse. Sljedeći odjeljci detaljno opisuju ključne komponente korištene u razvoju.

3.3.1. ASP.NET Core

ASP.NET Core [10] je višeplatformski razvojni okvir razvijen od strane Microsofta za izradu web-aplikacija u C# jeziku. Bazira se na kontrolerima ili minimalnim API-jem od kojih su u radu izabrani kontroleri kao ulazne točke. Pomoću nativnih funkcionalnosti i popularnih biblioteka omogućuje lagan i brz razvoj web-aplikacija baziranih na čistoj arhitekturi.

3.3.2. MediatR i FluentValidation

MediatR je biblioteka za implementaciju obrasca posrednika [11]. Kao takav omogućuje razdvajanje poslovne logike od kontrolera proslijedući komande i upite pravilnim izvođačima. Biblioteka također nudi proširivanje funkcionalnosti MediatR cjevovodom obrade zahtjeva. Tako se prije obrade zahtjeva može uvesti korak validacije, autorizacije, pokretanja i završavanja transakcija ili stvaranje dnevničkih zapisa. U sklopu ovog rada, biblioteka se koristila zajedno sa FluentValidation [12] bibliotekom koja nudi vrlo jednostavan pristup provjere ispravnosti stanja objekata. FluentValidation provjera ispravnosti uvedena je kao korak cjevovoda koji bi odbio komande i upite koji se nisu pridržavali zahtjeva. Primjer takvog zahtjeva je da naziv baze podataka smije sadržavati samo alfanumeričke znakove i crticu.

3.3.3. Entity Framework Core

Entity Framework Core (*EF Core* ili ukratko *EF*) je razvojni okvir i skup biblioteka koji služe za objektno relacijsko mapiranje. Vrlo je popularan zbog mogućnosti prijevoda C# LINQ sintakse u čisti SQL u upitima što ga čini vrlo jednostavnim za korištenje [13]. Osim toga, ovdje je izabran zbog odličnog mapiranja između hijerarhijske strukture domenskog modela i modela baze podataka. Za mapiranje hijerarhije EF nudi nekoliko različitih strategija [14]:

- **Table-per-hierarchy (TPH)** – pristup u kojem je cijela hijerarhija zapisana kao jedna tablica. Takva tablica sadrži opcionalna kolumna za svojstva svake klase, te sadrži polje diskriminatora u koji upisuje koji tip iz hijerarhije treba biti upotrijebљen. Predstavlja zadanu opciju, te je također učinkovitiji u slučajevima kada većina upita cilja baznu klasu.
- **Table-per-type (TPT)** – pristup u kojem svaki tip ima tablicu koja sadrži samo svojstva koja pripadaju isključivo tom tipu, dok za svojstva baznog tipa sadrži strani

ključ na tablicu baznog tipa. Ovakav pristup stvara normaliziranu bazu podataka nauštrb performansi te se zato ne preporučuje osim kada drugi nisu opcija

- **Table-per-concrete-type (TPC)** – pristup u kojem svaki tip ima tablicu, ali sadrži sva svojstva svojeg i baznog tipa. Predstavlja uglavnom učinkovitiju alternativu za TPT. Preporuča se koristiti kada su upiti prema konkretnim tipovima češći od upita prema baznim klasama.

U ovom radu korištena je TPC strategija, te je u nastavku prikazan dijagram modela baze podataka stvoren na temelju modela podataka sustava (Slika 3.3).



Slika 3.3. Model baze podataka temeljen na modelu sustava

3.3.4. React i Vite s TypeScriptom

Korisničko sučelje sustava Cloudy razvijeno je kao jednostranična web-aplikacija (engl. SPA - *Single Page Application*) koristeći **React** u kombinaciji s **TypeScriptom** i razvojnom

okolinom **Vite** [15]. React omogućuje deklarativni pristup izgradnji sučelja, komponentnu arhitekturu i učinkovito upravljanje stanjem aplikacije. Korištenje TypeScripta dodatno povećava sigurnost i održivost koda kroz statičku provjeru tipova, dok Vite osigurava brzu lokalnu izradu i razvoj s minimalnim vremenom pokretanja, automatskim osvježavanjem stranica te velikim ekosustavom dodataka od kojih je u radu korišten *proxy* dodatak za prosljeđivanje */api* zahtjeva prema poslužiteljskom dijelu za vrijeme izrade.

3.3.5. Mantine i Mantine Datatables

Za izradu vizualnog identiteta i korisničkog iskustva koristi se **Mantine** – moderna React biblioteka komponenti koja omogućuje brzu izradu responzivnih i prilagodljivih sučelja [16]. Mantine nudi bogat skup gotovih komponenti i React kuka (engl. *hook*) uz podršku za prilagodbu tema i stilova. React kuke dostupne kroz Mantine predstavljaju glavnu razliku u usporedbi s drugim bibliotekama komponenti omogućujući pojednostavljeni implementiranje čestih zahtjeva web-aplikacija.

S obzirom da Mantine biblioteka ne dolazi sa komponentom tablica podataka, za prikaz i upravljanje tabličnim podacima koristi se **mantine-datatables** [17], proširenje koje omogućuje sortiranje, filtriranje i paginaciju podataka na jednostavan i integriran način.

3.3.6. Redux i RTK Query

Za pomoć u upravljanju globalnim stanjem aplikacije, kao i komunikaciju s *backend* API-jem koristi se **Redux Toolkit** s **RTK Query** [18]. Redux omogućuje centralizirano spremanje stanja i predvidljivo upravljanje promjenama, dok RTK Query pojednostavljuje dohvati, korištenje predmemorije i sinkronizaciju podataka s API-jem. Ovaj pristup značajno smanjuje količinu potrebnog koda za rad s API-jima uz pomoć RTK Query generacije API-ja na temelju OpenAPI sheme.

3.3.7. React Chart.js 2

Za vizualizaciju metrika u obliku grafova koristi se **react-chartjs-2**, To je React omotač za popularnu biblioteku Chart.js [19]. Ova biblioteka omogućuje prikaz raznih tipova grafova (linijski, stupčasti, tortni i dr.) uz jednostavnu integraciju i prilagodbu prema potrebama aplikacije. Korištenjem biblioteke značajno je olakšan rad s grafovima u React okruženju.

3.4. Integrirani alati

Sustav Cloudy se oslanja na različite alate, posebno u području alata otvorenog koda, kako bi osigurao potpunu funkcionalnost upravljanja infrastrukturom. Sljedeći odjeljci detaljno opisuju ključne komponente i njihovu ulogu u sustavu.

3.4.1. Docker

Docker, kao alat za kontejnerizaciju, je odabran kao temeljna tehnologija zbog svoje široke prihvaćenosti, bogatog ekosustava i mogućnosti jednostavne integracije s ostalim alatima potrebnima za izgradnju suvremenih infrastrukturnih rješenja.

Docker pruža mogućnost pakiranja i pokretanja aplikacije u labavo izoliranom okruženju nazvanom kontejner. Izolacija i sigurnost omogućuju istovremeno pokretanje mnogih kontejnera na jednom domaćinu. Kontejneri su lagani i sadrže sve što je potrebno za pokretanje aplikacije, pa nije potrebno oslanjati se na ono što je instalirano na domaćinu [1].

U sustavu Cloudy svaki resurs koji pokreće neku aplikaciju ili servis koristi Docker kontejnere kao osnovu za izolaciju i upravljanje.

Za njihovu integraciju koriste se Docker mreže koje omogućuju sigurnu i fleksibilnu komunikaciju između kontejnera, bez potrebe za pristupom mreži domaćina. Za čuvanje stalnih podataka servisa koriste se Docker volumeni, s obzirom da Docker kontejneri ne zadržavaju podatke nakon gašenja.

Docker također nudi *exec* alat koji služi za pozivanje komandi unutar pokrenutog kontejnera. Na taj način domaćin može nadgledati i upravljati kontejnerom bez da su međusobno umreženi. Ovaj alat je ključan za upravljanje resursima, što je opisano u sljedećim poglavljima.

Osim navedenih osnovnih funkcionalnosti, Docker nudi i proširenje **Swarm mode**, koje nije istraženo u okviru ovog rada, ali predstavlja potencijalno proširenje sustava na više računala radi jednostavnog horizontalnog skaliranja. Swarm omogućuje povezivanje više Docker procesa u jedan klaster te pokretanje skaliranih resursa na različite čvorove uz automatsko balansiranje opterećenja. Iako Docker Swarm ne funkcioniра identično kao osnovni Docker Engine, oba alata dijele iste ili slične koncepte, pa bi proširenje na Swarm zahtijevalo generalne ali manje izmjene u implementaciji upravljanja resursima.

3.4.2. Alati za telemetriju

Telemetrijski sloj sustava Cloudy integrira moderne alate otvorenog koda kako bi omogućio gotovo potpunu preglednost infrastrukture bez ručne konfiguracije. Ključni alati uključuju:

- **OpenTelemetry Collector (s contrib proširenjima)** [20] – alat koji pruža platformu za prihvatanje, obradu i izvoz telemetrijskih podataka. U Cloudy sustavu koristi se kroz obrazac bočne prikolice (engl. *sidecar*) gdje se uz servis, na primjer bazu podataka, postavlja dodatni servis sakupljača metrika koji se spaja na tu bazu i sakuplja metrike specifične za taj tip resursa. Sakupljene metrike se zatim šalju prema alatu Prometheus. *Contrib* proširenje alata sadrži dodatna proširenja za specifična nadziranja raznih alata te je kao takav korišten u radu.
- **Prometheus** – alat za spremanje i obradu metričkih podataka. Alat periodično sakuplja metrike sa krajnjih web točaka servisa te ih sprema u vremensku bazu podataka. Iz baze se zatim mogu čitati pomoću PromQL jezika za upite. Alat u sustavu sakuplja metrike sa svih *OpenTelemetry Collector* instanci te sa cAdvisor alata zaduženog za sakupljanje drugih generičnih metrika.
- **CAdvisor** – Skraćenica za *Container Advisor*, predstavlja alat razvijen od strane Googlea za nadziranje kontejnerskih okruženja. Alat prikuplja metrike poput onih o statusu procesora, memorije i diska na razini cijelog sustava ali i detaljnije na razini svakog kontejnera. Te metrike se koriste za nadziranje aktivnosti resursa kroz upravljački panel.
- **Loki** – alat inspiriran alatom **Prometheus**, no za razliku od njega služi za agregiranje i čitanje dnevničkih zapisa. Svi dnevnički zapisi u sustavu se čuvaju u njemu. Podržava čitanje zapisa sličnim API-jem kao i Prometheus za metrike.
- **Loki driver za Docker** – Proširenje za Docker koje mijenja zadani upravljač dnevničkih zapisa s upravljačem koji ih šalje prema loki servisu. U sustavu je instaliran u Docker prilikom prvog pokretanja a zatim kontejneri za svaki resurs postavljaju upravljač zapisa na Loki proširenje.
- **Grafana** – Centralizirano sučelje za vizualizaciju dnevničkih zapisa, metrika i tragova. Uglavnom se uparuje sa Prometheus i Loki alatima, te tako stvara popularan paket (engl. *stack*) alata za telemetriju sustava. U Cloudy sustavu koristi se za omogućavanje korisniku detaljnijeg uvida u telemetrijske podatke.

3.4.3. Caddy

U procesu odabira *reverse proxy* rješenja za sustav Cloudy razmatrane su četiri glavne opcije: Nginx, Nginx Proxy Manager (NPM), Traefik i Caddy. Veliki fokus je bio na jednostavnoj integraciji, proširivosti za budući razvoj te rješavanju problema dobave certifikata za siguran promet.

- **Nginx** [21] – jedan od dugo najpopularnijih alata za *reverse proxy*, pomno razmatran, nije odabran zbog nedostatka nativne podrške za Let's Encrypt, što bi znatno zakompliciralo održavanje HTTPS certifikata i povećalo operativni napor.
- **Nginx Proxy Manager (NPM)** [22]- predstavlja proširenje Nginxa s grafičkim sučeljem za olakšano upravljanje, kao i nativnom podrškom za Let's Encrypt. Alat se pokazao previše fokusiranim na GUI upravljanje, dok je uređivanje konfiguracijskih datoteka i dalje složeno, što ga čini manje pogodnim za automatizirano upravljanje konfiguracijama.
- **Traefik** [23] – alat koji nudi potpunu podršku za Let's Encrypt i predstavlja najbližu alternativu Caddy alatu. Međutim, poznat je kao složeniji i komplikiraniji *reverse proxy* manager, što može otežati upravljanje i održavanje.
- **Caddy** je odabran zbog svoje inherentne podrške za HTTPS ("Caddy is the first and only web server to use HTTPS automatically and by default" [24]), minimalne potrebne konfiguracije i dobrih prepostavljenih postavki koje omogućuju jednostavnu i pouzdanu implementaciju sigurnih veza.

Caddy koristi jednostavan i čitljiv format konfiguracije nazvan **Caddyfile**, koji omogućuje brzo i intuitivno definiranje pravila. Za složenije i automatizirane scenarije, uključujući onaj u sustavu Cloudy, koristi se **Caddy JSON konfiguracija**, koja ima strogu, ali kompleksniju shemu, pogodnu za generiranje i upravljanje konfiguracijom putem alata.

Iako Caddy služi prvenstveno za HTTP prosljeđivanje prometa, alat podržava proširenja poput l4 koja omogućuju korištenje funkcionalnosti za prosljeđivanje TCP prometa. To otvara mogućnosti za budući razvoj sustava, uključujući prosljeđivanje prometa baza podataka i posrednika poruka koji se ne zasnivaju na HTTP protokolu.

3.4.4. Postgres

PostgreSQL je odabran kao osnovni sustav za upravljanje bazama podataka zbog svoje robusnosti i fleksibilnosti, te kao takav najpopularniji sustav za upravljanje bazom podataka otvorenog koda. U sustavu Cloudy se koristi prilagođena Docker slika koja dodatno proširuje mogućnosti standardnog PostgreSQL-a. Slika sadrži predinstalirane sljedeće često korištene ekstenzije:

- **PostGIS** [25] – ekstenzija koja omogućuje pohranu i obradu geoprostornih podataka unutar PostgreSQL baze podataka.
- **Pgvector** [26] – ekstenzija namijenjena za rad s vektorskim podacima i pretraživanje po sličnosti u visokodimenzionalnim prostorima

Dodatna funkcionalnost Postgres baza podataka, koja je uključena na svakoj bazi stvorenoj kroz sustav je automatska aktivacija ekstenzije pg_stat_statements. Ova ekstenzija sakuplja statistiku o svim upitima na sustavu te se može koristiti za istraživanje upita sa najvećim utjecajima na performanse.

3.4.5. Kafka

Apache Kafka je distribuirana platforma za pohranu događaja i obradu tokova podataka u stvarnom vremenu (engl. *event streaming platform*) [27]. Razvijen je kao rješenje otvorenog koda koje omogućuje izgradnju podatkovnih cjevovoda visokih performansi, analitiku tokova i pouzdane sustave za prijenos poruka.

Osnovne značajke Kafke kao i svakog drugog posrednika poruka su objavljivanje i preplate na tokove zapisa, visoku propusnost i nisku latenciju. Kafka se razlikuje od drugih sličnih alata jer se jednom pročitane poruke spremaju te mogu ponovno čitati, ovisno o postavljenoj vrijednosti pomaka koju Kafka čuva za svakog slušatelja.

3.5. Kontejnerizacija sustava

Sustav Cloudy u potpunosti je implementiran kao skup Docker kontejnera. To uključuje i glavnu aplikaciju i bazu podataka, uz sve pomoćne servise koji se pokreću unutar zasebnih kontejnera, čime se osigurava izolacija i dosljednost okruženja bez potrebe za instalacijom pojedinačnih komponenti poput .NET okruženja za pokretanje na domaćinu. Docker API proslijeden je aplikaciji putem vezane putanje domaćina na docker.sock priključak, što

omogućuje upravljanje svim resursima i servisima iz same aplikacije koja i sama živi u istom Docker okruženju.

Za pohranu podataka koriste se Docker volumeni i mapirani direktoriji s domaćina, čime se omogućuje trajnost podataka i jednostavno upravljanje datotekama između kontejnera i sustava domaćina. Datoteke, poput raznih konfiguracija, koje su potrebne drugim kontejnerima (Odsječak 1), kopiraju se u mapu u kontejneru `/app/persist` koja je povezana na stvarnu mapu na domaćinu. Zatim se tekst putanje uređuje i `/app/persist` dio putanje se mijenja sa stvarnom putanjom koja se nalazi na domaćinu te joj drugi kontejneri mogu pristupiti.

Kontejnerizacija također omogućuje jednostavno umrežavanje upravljačkog sustava kroz Docker virtualne mreže, tako da drugi resursi komuniciraju sa sustavom izolirano od vanjske mreže. Instalacija i pokretanje cijelog sustava svodi se na kloniranje repozitorija i pokretanje jedne Docker Compose naredbe, bez potrebe za ručnom instalacijom .NET okruženja, baze podataka ili drugih alata na poslužitelju.

```
receivers:  
  postgresql:  
    endpoint: "${DATA_SOURCE_URI}"  
    username: "${DATA_SOURCE_USER}"  
    password: "${DATA_SOURCE_PASS}"  
    exclude_databases:  
      - "postgres"  
      - "CloudyAdmin"  
    collection_interval: 30s  
    tls:  
      insecure: true  
    query_sample_collection:  
      enabled: true  
    top_query_collection:  
      enabled: true  
      top_n_query: 10  
      max_rows_per_query: 10  
  metrics:  
    postgresql.tup_inserted:  
      enabled: true  
    postgresql.tup_returned:  
      enabled: true  
    postgresql.tup_updated:  
      enabled: true  
    postgresql.tup_deleted:  
      enabled: true  
  connection_pool:  
    max_idle_time: 45s  
    max_lifetime: 1m
```

```

    max_idle: 5
    max_open: 10

processors:
  batch:
  resource:
    attributes:
      - key: service.name
        value: "${DATA_SOURCE_NAME}"
        action: upsert
exporters:
  debug:
    verbosity: detailed
  prometheus:
    endpoint: "0.0.0.0:8889"
  otelhttp:
    endpoint: "${OTLP_ENDPOINT}"

service:
  telemetry:
    logs:
      level: "debug"
  pipelines:
    logs:
      receivers: [postgresql]
      processors: [batch, resource]
      exporters: [otelhttp, debug]
    metrics:
      receivers: [postgresql]
      processors: [batch]
      exporters: [prometheus]

```

Odsječak 1. Primjer konfiguracije za kontejner sakupljača Postgres telemetrije

3.6. Upravljanje sa izoliranim resursima

Upravljanje resursima u sustavu Cloudy temelji se na korištenju **Docker exec** mehanizma za izvršavanje naredbi unutar pokrenutih kontejnera. Skoro sve administrativne i korisničke operacije, uključujući rad s bazama podataka, posrednicima poruka i web-aplikacijama, provode se kroz CLI alate unutar izoliranih okruženja. Na taj način nije potrebno otvarati dodatne portove prema domaćinu ili internetu, čime se povećava sigurnost i smanjuje površina napada.

Izvođenje upita nad bazom podataka omogućeno je putem posebnog sučelja koje koristi *Docker exec* za pokretanje SQL klijenata unutar kontejnera baze, bez izravnog izlaganja

servisa vanjskoj mreži. Slično, za posrednike poruka implementirani su alati za proizvodnju i konzumaciju poruka koji se također izvršavaju unutar kontejnera putem *Docker exec*.

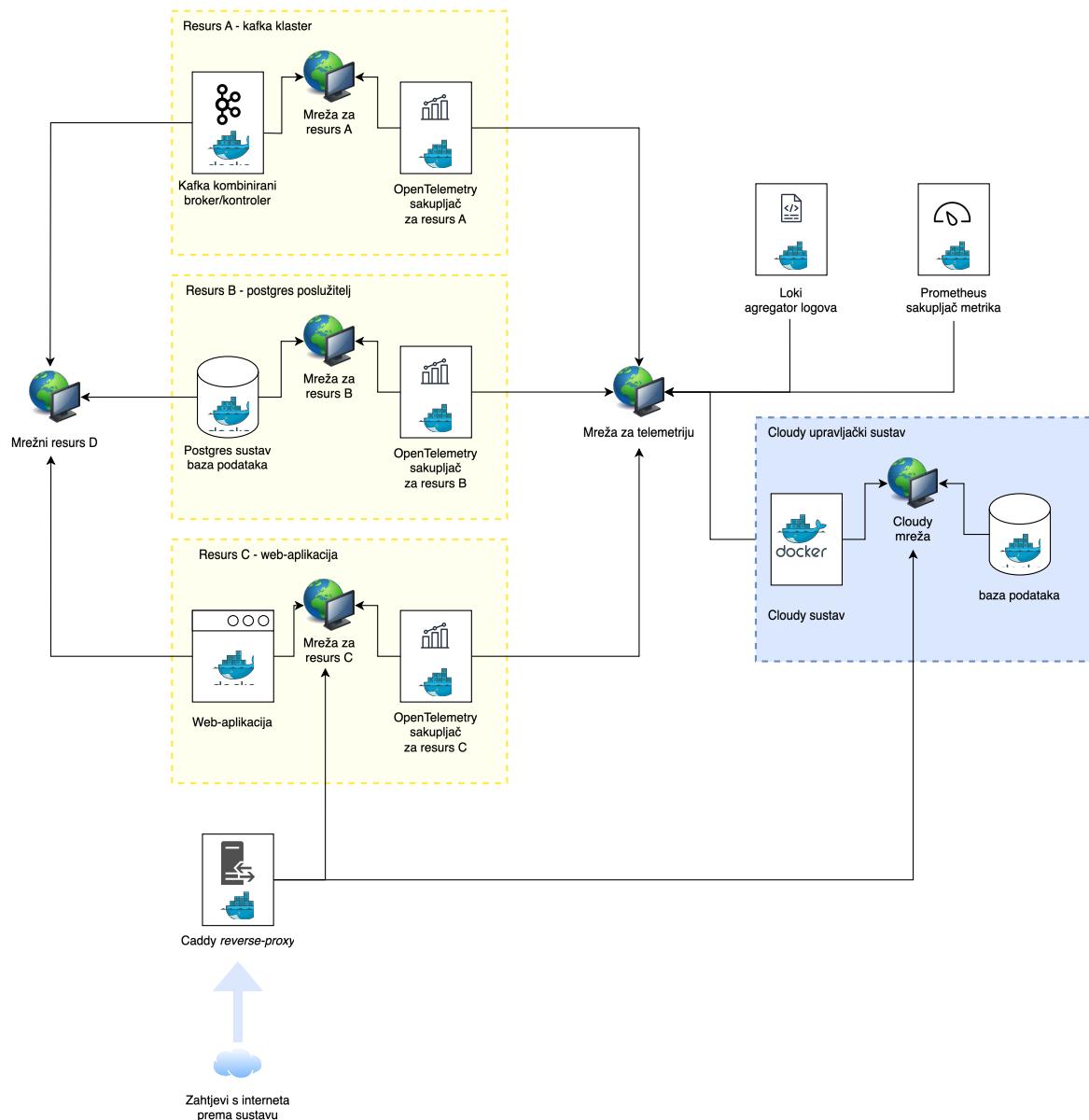
Iznimke za ovakav pristup su alati koji ne pripadaju resursu, a nude jednostavan API pristup poput alata za telemetriju. S njima je upravljački sustav umrežen u istu virtualnu mrežu i poziva API-je nedostupne domaćinu.

3.7. Umrežavanje kontejnera iza resursa

Umrežavanje kontejnera u sustavu temelji se na principu izolacije i precizne kontrole komunikacije između resursa. Većina resursa implementirana je tako da koristi vlastitu Docker mrežu u kojoj se nalaze dva kontejnera: glavni kontejner s aplikacijom ili servisom, te dodatni kontejner za telemetriju. Ovaj dodatni kontejner, prema obrascu bočne prikolice, spaja se na glavni kontejner i prikuplja metrike i dnevničke zapise specifične za taj resurs. Istovremeno je povezan na zajedničku mrežu za telemetriju (na koju glavni kontejner nije povezan), gdje se nalaze alati Prometheus i Loki, što omogućuje centralizirano prikupljanje i obradu podataka bez slučajnog izlaganja resursa mrežama sa drugim resursima.

Kada se resurs izlaže internetu, u njegovu mrežu se dinamički dodaje *reverse proxy* kontejner (Caddy), koji preuzima promet i prosledjuje ga isključivo ciljanom resursu. Alternativno je istraživan model u kojem bi svi resursi koji trebaju biti javno dostupni bili povezani na zajedničku mrežu u kojoj se nalazi i kojom upravlja *reverse proxy*, no takav pristup bi omogućio i internu komunikaciju između svih tih resursa, što nije poželjno iz sigurnosnih razloga.

Opisana struktura prikazana je na manjem primjeru u nastavku (Slika 3.4). Slika prikazuje web-aplikaciju dostupnu putem interneta koja interno komunicira sa dva resursa, bazom podataka i posrednikom poruka. Osim toga pokazan je dio telemetrijske mreže u koju su spojeni isključivo telemetrijski alati, te mreža za *reverse proxy* ulaznu točku sustava.



Slika 3.4. Primjer umreženja sustava

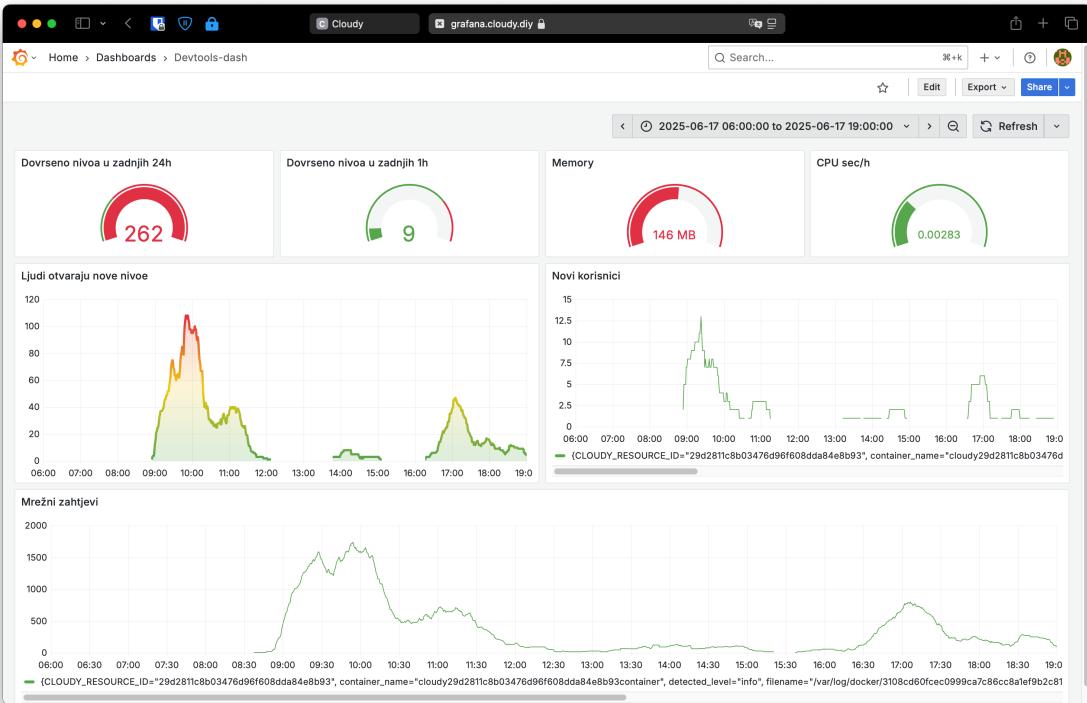
4. Testiranje sustava

Testiranje sustava obavljeno je dijelom u suradnji sa testnim korisnicima, a dijelom sa prezentacijskim primjerima. U nastavku je kratki opis svakog sustava

- Edukativna igra **DevToolsDash** – web-aplikacija izgrađena uz pomoć Python FastAPI i React.js razvojnih okvira uparena s Postgres bazom podataka. Tematski, radi se o edukativnoj igri kojoj je cilj približiti alat *Chrome DevTools* i njegove naprednije mogućnosti. Igra je razvijena u sklopu FER diplomskog rada računarskog smjera Tereze Bilić [28]
- Digitalni penjački vodič **Alpinity** – web-aplikacija izrađena u klijentsko poslužiteljskoj arhitekturi dvije pod-aplikacije: Next.js korisnička i ASP.NET poslužiteljska aplikacija. Poslužitelj je također zadužen za posluživanje API-ja mobilnoj aplikaciji. Aplikacija je uparena sa Postgres bazom podataka uz proširenje za geopodatke postgis. Aplikacija korisniku nudi istraživanje penjačkih lokacija i njihovih detalja, kao i praćenje osobnog napretka u sportu penjanja. Aplikacija je razvijena u sklopu FER diplomskog rada računarskog smjera Adriana Cvijanovića [29]
- Laboratorijski primjer **umreženih mikroservisa** koji za komunikaciju koriste posrednik poruka – sustav prikazuje cjevovod prijenosa i održavanja podataka između nekoliko servisa u simulaciji malog mikroservisnog okruženja. Sastoji se od 3 Python servisa: proizvođača, potrošača i API-ja. Također, sustav zahtjeva dvije baze podataka između kojih se sinkroniziraju podaci.

4.1. Projekt DevToolsDash

Projekt predstavlja edukativnu igru koja se sastoji od poslužitelja koji se oslanja na FastAPI razvojni okvir i klijenta koji koristi React.js i Vite biblioteke. Za vrijeme izgradnje web aplikacije, klijentska aplikacija se zapakira u poslužiteljsku te je tako potrebno samo jednu kombiniranu aplikaciju postaviti na poslužitelj. Nakon postavljanja igra je testirana sa pravim korisnicima, čija se aktivnost uz zdravlje sustava pratila kroz Grafana nadzornu ploču (Slika 4.1).



Slika 4.1. Nadzor aplikacije DevToolsDash u stvarnom vremenu

Proces postavljanja aplikacije je bio nadziran uz minimalnu pomoć u obliku odgovaranja osnovnih pitanja o sustavu. Na kraju, sustav je bio uspješno postavljen, te je njegova slika ekrana prikaza je u nastavku (Slika 4.2). Povratne informacije ukazuju da, iako je sustav jednostavan za korištenje kada je potrebno koristiti specifičnu komponentu, odabir sljedeće komponente za upotrijebiti nije sasvim jasan te nedostaje konkretna uputa što napraviti kao idući korak.



Slika 4.2. Edukativna igra DevToolsDash (treba citat)

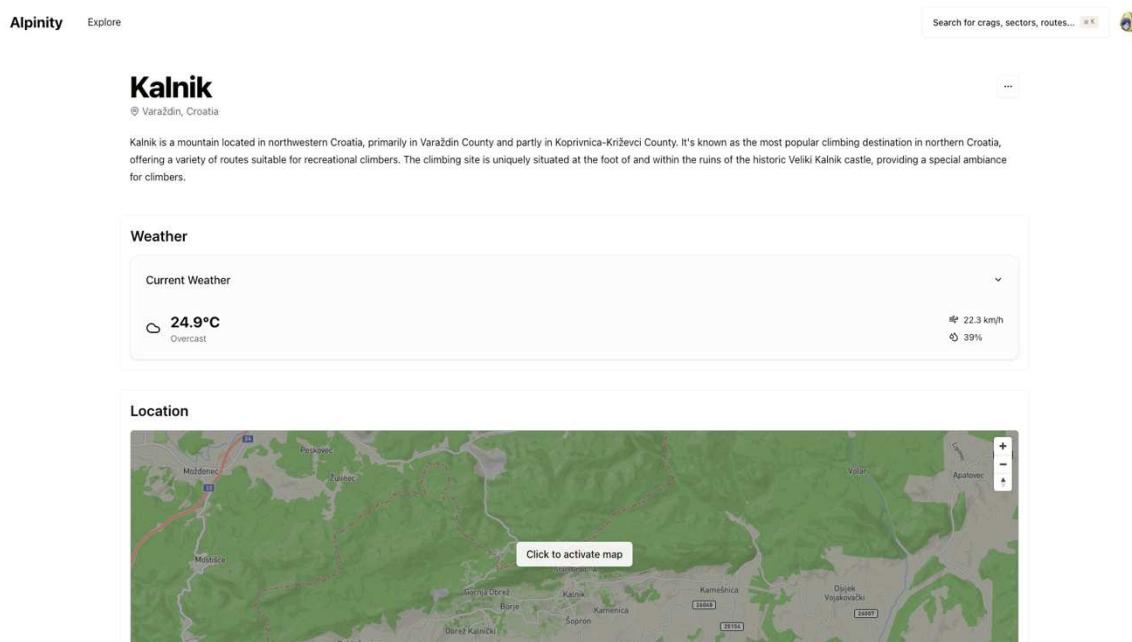
Za potrebe projekta stvorena je grupa resursa devtools-dash (Slika 4.3) koja kao resurse sadrži web-aplikaciju, Postgres poslužitelj i bazu podataka, te virtualnu mrežu koja povezuje Postgres resurs i web-aplikaciju.

| Name | Resource type | Resource ID |
|-------------------------|-------------------|--------------------------------------|
| devtoolsdashdb2 | Postgres Database | 150f4e9c-22b2-4ad2-db5c-08dda852a1d2 |
| devtools-dash-network | Virtual Network | 6c672f76-ca52-428c-db5b-08dda852a1d2 |
| devtools-dash-db-server | Postgres Server | 12ade9ad-0314-49c1-96f7-08dda84e8b93 |
| devtools-dash-app | Web Application | 29d2811c-8b03-476d-96f6-08dda84e8b93 |

Slika 4.3. Pregled grupe resursa *devtools-dash*

4.2. Projekt Alpinity

Slično kao i web-aplikacija DevToolsDash, ovdje se radi o SPA aplikaciji sa API poslužiteljem. Za razliku od prethodnog primjera, u projektu Alpinity zbog poslužiteljske generacije stranica postoje dvije web-aplikacije: Next.js klijentska aplikacija te ASP.NET Core API poslužitelj. Zbog navedenog stvorena su dva resursa web-aplikacije, svaki sa otvaranjem prema internetu s obzirom da korisnički preglednik pristupa objema. Postgres baza podataka stvorena za sustav proširena je sa PostGis proširenjem dostupnim kroz alat za upravljanje Postgres proširenjima unutar sustava Cloudy. Slika iz uspješno postavljenog sustava prikazana je u nastavku (Slika 4.4).



Slika 4.4. Projekt Alpinity (treba citat)

Lista postavljenih resursa demonstrirana je kroz pregled grupe resursa u sučelju (Slika 4.5).

| Name | Resource type | Resource ID |
|-------------------|-------------------|--------------------------------------|
| alpinity_sql_new | Postgres Database | 5ed2c5a7-9ce8-4bb2-3282-08dda90e6ba2 |
| alpinity_back_net | Virtual Network | 0fd5b0fe-e6fc-4b23-f3f7-08dda858a07f |
| alpinity_db_new | Postgres Server | 858b6307-fdd0-4eaf-3281-08dda90e6ba2 |
| alpinity_back | Web Application | 9ef84962-fe3d-4a64-f3f6-08dda858a07f |
| alpinity_web | Web Application | e34a615f-3509-4703-c577-08dda91b8577 |

Slika 4.5. Pregled grupe resursa *alpinity*

Kao i u prethodnom primjeru korisnik sustava ističe jednostavnost korištenja, posebno uz pregled dnevničkih zapisa za vrijeme izgradnje kao i za vrijeme testiranja mobilne aplikacije uživo kako bi se otklonili problemi. Također ističu se problemi nejasnoće što je pravilan redoslijed koraka i koji su preostali kako bi se došlo do proizvodnog stanja sustava. Na temelju dobivenih povratnih informacija, u sustav su dodani odjeljci za sljedeće korake (2.5 Ostali zahtjevi).

4.3. Laboratorijski primjer – posrednici poruka

Primjer predstavlja testiranje sustava koji koriste posrednike poruka u mikroservisnim i drugim okruženjima. U stvorenom okruženju postoji neobrađena (engl. *raw*) baza podataka koja sadrži unesene podatke, koji nemaju definirane relacije niti garantiraju ispravnost. To su podaci o komentarima na objave neke zamišljene stranice. Baza predstavlja izvor sakupljenih podataka prije ikakve obrade, te kao takva sadrži samo jednu tablicu komentara na objave koja može sadržavati prazna ili neispravna polja. Zatim je postavljen Python servis (u sustavu je korišten resurs web-aplikacije, koja nije otvorena prema internetu) koji periodično provjerava podatke, te nove podatke, nakon provjere ispravnosti i grupiranja po objavama šalje na Kafka posrednik poruka. Drugi Python resurs potrošača sakuplja poruke oblikovanih podataka te ih sprema u svoju bazu obrađenih podataka. Na bazu obrađenih podataka također se spaja treća web-aplikacija koja nudi obrađene podatke kroz API krajnjem točku. Okruženje postavljenih resursa vidljivo je u obliku stvorenih grupa resursa u Cloudy sučelju (Slika 4.6).

| Name | Resource type | Resource ID |
|--------------------------|-------------------|--------------------------------------|
| msg_example_raw_db | Postgres Database | f6d29cbb-cc5d-4868-6b7a-08ddb337e085 |
| msg_example_processed_db | Postgres Database | 0afec27b-d71e-4ef3-6b7b-08ddb337e085 |
| messages-net | Virtual Network | 3d87cd0a-63d6-4250-6b7f-08ddb337e085 |
| msgexamplekafka | Kafka Cluster | 67dff594-cf5e-4681-6b78-08ddb337e085 |
| msg_example_pg_server | Postgres Server | 40514d91-4559-4584-6b79-08ddb337e085 |
| messages_producer | Web Application | f5dbf16a-7da2-4f02-6b7c-08ddb337e085 |
| messages_consumer | Web Application | c1ea901b-742d-4fd7-6b7d-08ddb337e085 |
| messages_api | Web Application | 01e473e3-70c8-466e-6b7e-08ddb337e085 |

Slika 4.6. Pregled grupe resursa *messages*

Pregled protoka poruka kroz sustav moguće je napraviti uz pomoć alata za slušanje poruka (Slika 4.7.).

| Partition | Leader | Replicas | ISR | ELR | Last Known ELR |
|-----------|--------|----------|-----|-----|----------------|
| 0 | 1 | 1 | 1 | 1 | 1 |

Slika 4.7. Pregled protoka poruka u primjeru posrednika poruka

Glavni nedostatak u ovom primjeru je nepostojanje autentifikacije u slučaju daljnog skaliranja na veće sustave koji rade s posrednicima poruka. U takvim sustavima svaki proizvođač i potrošač bi trebao imati jasno definirane granice pristupa nad Kafka klasterom.

Kroz testiranje sustava uočeno je da, iako sasvim funkcionalno, okruženje se može još više prilagoditi pozadinskim aplikacijskim servisima koji ne sadrže HTTP poslužitelj. Takvi

servisi često imaju specifične okidače, bilo to vremenski ili na temelju primitka poruke te pružanje mogućnosti jednostavnijeg praćenja okidanja aktivnosti bi pridonijelo uvidu u sustav.

Izvorni kod ostvarenih servisa dostupan je kao privitak zajedno sa izvornim kodom sustava Cloudy na adresi <https://github.com/kjkardum/personal-cloud/tree/master> .

5. Budući rad

U budućem razvoju sustava Cloudy otvara se niz mogućnosti za proširenje funkcionalnosti i povećanje razine automatizacije, sigurnosti i skalabilnosti. Jedan od najvažnijih smjerova je implementacija horizontalnog skaliranja putem Docker Swarm alata. Uvođenjem Swarm klastera omogućila bi se raspodjela resursa na više čvorova, automatsko balansiranje opterećenja i povećana otpornost na greške, što je posebno važno za produkcijska okruženja i veće timove. Swarm bi omogućio dinamičko dodavanje i uklanjanje čvorova, automatsko prebacivanje na druge Docker procese u slučaju kvara te distribuirano izvršavanje zadataka bez ručne intervencije.

Uz horizontalno skaliranje, važno je i unaprijediti vertikalno skaliranje resursa kroz mogućnost postavljanja granica procesora i memorije za svaki pojedini Docker kontejner. Time bi se omogućilo preciznije upravljanje resursima i njihovo dinamičko prilagođavanje temeljem telemetrijskih podataka, a integracija s Docker API-jem omogućila bi automatsko podešavanje ovih parametara prema stvarnim potrebama sustava i korisnika.

Dodatno, postoji prostor za optimizaciju dugotrajnih operacija poput izgradnje Docker slika ili povlačenja velikih repozitorija. Detaljniji uvid u tijek i trajanje ovih procesa omogućio bi bolje planiranje i nadzor, a korisnicima bi pružio jasne informacije o statusu inicijalizacije resursa i eventualnim greškama u radu sustava.

Proširenje mrežnih mogućnosti sustava uključuje podršku za TCP promet i TLS enkripciju na *reverse proxy* sloju, što bi omogućilo siguran pristup bazama podataka i posrednicima poruka, kao i automatsko upravljanje certifikatima za TCP mrežne servise. Ovakva funkcionalnost proširila bi interoperabilnost sustava sa vanjskim servisima koji se oslanjaju na navedene alate.

Implementacija autorizacije temeljene na ulogama, odnosno RBAC (engl. *role-based access control*) modela proširila bi sustav mnogim većim scenarijima upotrebe. Time bi se omogućilo definiranje prilagođenih uloga i precizna kontrola pristupa po resursima i operacijama, što je važno za timove s više korisnika i različitim razinama odgovornosti. Integracija s dnevnikom radnji omogućila bi detaljno praćenje svih promjena i aktivnosti u sustavu.

Na području sigurnosti podataka, prilika je za implementaciju automatiziranih sigurnosnih kopija za PostgreSQL baze podataka, uključujući potencijalnu podršku za *point-in-time*

recovery, kao i šifriranje pohranjenih podataka na razini volumena i redundantno pohranjivanje na udaljenu pohranu. Ove funkcionalnosti dodatno bi povećale pouzdanost i otpornost sustava na gubitak podataka ili sigurnosne incidente.

Daljnje proširenje sustava uključuje podršku za dodatne baze podataka poput MariaDB, Microsoft SQL Servera i MongoDBa, čime bi se korisnicima omogućio širi izbor tehnologija prema potrebama projekata. Također, uz izvedbu integracije servisa za pohranu velikih binarnih objekata (BLOB, engl. ...), automatizirani procesi sigurnosnih kopija i drugih stalnih datoteka mogli bi se direktno kontrolirati pomoću resursa tih servisa.

Proširenje resursa web-aplikacija na druge metode postavljanja, poput privatnih Git repozitorija, Docker repozitorija ili statičnih datoteka bi proširilo korisnost resursa web-aplikacija za više slučajeva korištenja.

Posljednje, jedan od ključnih pravaca razvoja je automatizacija telemetrije, uključujući automatsko generiranje Grafana nadzornih ploča kroz sučelje sustava, te dodatni grafički pregledi telemetrije unutar samog upravljačkog sustava.

6. Zaključak

Ovaj rad uspješno demonstrira mogućnost stvaranja praktičnog sustava za upravljanje vlastitom infrastrukturom koji kombinira jednostavnost infrastrukture u oblaku s kontrolom i ekonomičnosti vlastitih rješenja. Sustav Cloudy predstavlja poseban doprinos u području telemetrije i uključenih alata za upravljanje komponentama infrastrukture.

Implementacija pokazuje da je moguće izgraditi sofisticiran sustav za upravljanje infrastrukturom koristeći tehnologije otvorenog koda, te da takav sustav može pružiti značajnu vrijednost korisnicima kroz automatizaciju, centraliziranu telemetriju i jednostavno upravljanje. Rezultati testiranja potvrđuju praktičnu korisnost sustava, dok identificirana područja za poboljšanje pružaju jasan put za budući razvoj.

Kroz ovaj rad potvrđena je hipoteza da je moguće stvoriti sustav koji premošćuje jaz između skupih upravljenih cloud servisa i kompleksnih neupravljenih rješenja. Iako rad ostavlja mnoga područja za proširenje, postavljena je jasna baza za svako od navedenih proširenja te bi njihova implementacija zahtijevala manje zahtjevne nadogradnje sustava.

7. Literatura

- [1] Docker. Inc, »Docker Docs,« [Mrežno]. Available: <https://docs.docker.com/get-started/docker-overview/>. [Pokušaj pristupa 22 6 2025].
- [2] c. L. L. C., »cPanel stranica proizvoda,« [Mrežno]. Available: <https://cpanel.net/>. [Pokušaj pristupa 14 5 2025].
- [3] »Prometheus Dokumentacija,« [Mrežno]. Available: <https://prometheus.io/docs/introduction/overview/>. [Pokušaj pristupa 4 6 2025].
- [4] »Grafana Dokumentacija,« [Mrežno]. Available: <https://grafana.com/docs/grafana/latest/introduction/>. [Pokušaj pristupa 4 6 2025].
- [5] »Loki Dokumentacija,« [Mrežno]. Available: <https://grafana.com/docs/loki/latest/get-started/overview/>. [Pokušaj pristupa 4 6 2025].
- [6] »Portainer dokumentacija,« [Mrežno]. Available: <https://docs.portainer.io/>. [Pokušaj pristupa 7 6 2025].
- [7] »Dokku Dokumentacija,« [Mrežno]. Available: <https://dokku.com/docs/getting-started/installation/>. [Pokušaj pristupa 7 6 2025].
- [8] »CapRover Dokumentacija,« [Mrežno]. Available: <https://caprover.com/docs/get-started.html>. [Pokušaj pristupa 8 6 2025].
- [9] »Digital Ocean Droplet Docs,« [Mrežno]. Available: <https://docs.digitalocean.com/products/droplets/>. [Pokušaj pristupa 8 6 2025].
- [10] »ASP.NET Core Dokumentacija,« [Mrežno]. Available: <https://learn.microsoft.com/en-us/aspnet/core/?view=aspnetcore-9.0>. [Pokušaj pristupa 15 6 2025].
- [11] »MediatR Wiki,« [Mrežno]. Available: <https://github.com/jbogard/MediatR/wiki>. [Pokušaj pristupa 15 6 2025].

- [12] »FluentValidation Dokumentacija,« [Mrežno]. Available:
<https://docs.fluentvalidation.net/en/latest/>. [Pokušaj pristupa 15 6 2025].
- [13] »Entity Framework Core Dokumentacija,« [Mrežno]. Available:
<https://learn.microsoft.com/en-us/ef/core/>. [Pokušaj pristupa 16 6 2025].
- [14] »Entity Framework Core - Inheritance,« [Mrežno]. Available:
<https://learn.microsoft.com/en-us/ef/core/modeling/inheritance>. [Pokušaj pristupa 16 6 2025].
- [15] »Vite - Getting Started,« [Mrežno]. Available: <https://vite.dev/guide/>. [Pokušaj pristupa 16 6 2025].
- [16] »Mantine Docs,« [Mrežno]. Available: <https://mantine.dev/>. [Pokušaj pristupa 17 6 2025].
- [17] »Mantine Datatables Dokumentacija,« [Mrežno]. Available:
<https://icflorescu.github.io/mantine-datatable/>. [Pokušaj pristupa 17 6 2025].
- [18] »RTK Query Docs,« [Mrežno]. Available: <https://redux-toolkit.js.org/rtk-query/overview>. [Pokušaj pristupa 17 6 2025].
- [19] »React Chart.js 2 Dokumentacija,« [Mrežno]. Available: <https://react-chartjs-2.js.org/>. [Pokušaj pristupa 17 6 2025].
- [20] »OpenTelemetry Collector Contrib Repozitorij,« [Mrežno]. Available:
<https://github.com/open-telemetry/opentelemetry-collector-contrib>. [Pokušaj pristupa 17 6 2025].
- [21] »Nginx HTTPS Dokumentacija,« [Mrežno]. Available:
https://nginx.org/en/docs/http/configuring_https_servers.html. [Pokušaj pristupa 18 6 2025].
- [22] »Nginx Proxy Manager Dokumentacija,« [Mrežno]. Available:
<https://nginxproxymanager.com/guide/>. [Pokušaj pristupa 18 6 2025].

- [23] »Trafeik Dokumentacija,« [Mrežno]. Available: <https://doc.traefik.io/traefik/>. [Pokušaj pristupa 17 6 2025].
- [24] “Automatic HTTPS - Caddy documentation,” [Online]. Available: <https://caddyserver.com/docs/automatic-https>. [Accessed 22 6 2025].
- [25] »Postgis Dokumentacija,« [Mrežno]. Available: https://postgis.net/documentation/getting_started/. [Pokušaj pristupa 19 6 2025].
- [26] »Pgvector Repozitorij,« [Mrežno]. Available: <https://github.com/pgvector/pgvector>. [Pokušaj pristupa 18 6 2025].
- [27] »Apache Kafka 3 Documentation,« [Mrežno]. Available: <https://kafka.apache.org/30/documentation.html>. [Pokušaj pristupa 19 6 2025].
- [28] T. Bilić, »Obrazovna igra o alatu Google Chrome DevTools namijenjenom uvidu u aplikacije weba (u postupku objave),« Zagreb, 2025.
- [29] A. Cvijanović, »Aplikacija za interaktivnu pomoć penjačima po stijenama uporabom proširene stvarnosti (u postupku objave),« Zagreb, 2025.

Prototip sustava za lokalno upravljanje infrastrukturom aplikacija

Sažetak

U ovom radu fokus je na proučavanju, implementaciji te analizi sustava za upravljanje infrastrukturom aplikacija. Izrađen je sustav koristeći model „upravljenih resursa“ u kojima sustav preuzima kontrolu nad održavanjem i postavljanjem resursa, dok korisniku pruža minimalne ovlasti potrebne za upotrebu. Testiranjem nad stvarnim sustavima i korisnicima pokazano je da je implementirani sustav uspješno ispunio svoj zadatak, uz jasno definirana područja za poboljšanje.

Ključne riječi: infrastruktura u oblaku, Docker, upravljeni resursi, telemetrija, prototipiranje

Prototype of a System for Local Application Infrastructure Management

Abstract

This thesis focuses on the study, implementation, and analysis of a system for managing application infrastructure. A system was developed using a "managed resources" model, in which the system assumes control over the maintenance and deployment of resources, while providing the user with the minimum privileges necessary for operation. Testing on real-world systems and with users has demonstrated that the implemented system successfully fulfills its objective, while also identifying clearly defined areas for improvement.

Keywords: cloud infrastructure, Docker, managed resources, telemetry, prototyping