Razvoj nadzorne ploče za klastere Kubernetes

Zečević, Stipe

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 582

RAZVOJ NADZORNE PLOČE ZA KLASTERE KUBERNETES

Stipe Zečević

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 582

RAZVOJ NADZORNE PLOČE ZA KLASTERE KUBERNETES

Stipe Zečević

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 4. ožujka 2024.

DIPLOMSKI ZADATAK br. 582

Pristupnik:	**Stipe Zečević (0036514213)**
Studij:	Računarstvo
Profil:	Računalno inženjerstvo
Mentor:	izv. prof. dr. sc. Vladimir Čeperić

Zadatak: **Razvoj nadzorne ploče za klastere Kubernetes**

Opis zadatka:

U ovom diplomskom radu razvit će se nadzorna ploča za upravljanje i nadzor tehnologije Kubernetes, prilagođena za korištenje na webu i mobilnim uređajima. Koristit će se K3s distribucija Kubernetesa, koja je optimizirana za rad u uvjetima s ograničenim resursima i na rubnim lokacijama, što uključuje i IoT uređaje. Nadzorna ploča imat će mogućnost visoke dostupnosti te pojednostavljenu upotrebu. Posebna pažnja bit će posvećena kompatibilnosti s raznim uređajima unutar klastera poput ARM uređaja. Nadzorna ploča omogućit će korisnicima jednostavnu i efikasnu interakciju s klasterom, bez obzira na lokaciju ili uređaj. Kroz rad će biti detaljno opisan proces razvoja, od postavljanja razvojnog okruženja, preko integracije s klasterom Kubernetes, do testiranja funkcionalnosti na različitim platformama. Uz teorijski dio koji će pokriti osnove tehnologije Kubernetes, rad će pružiti praktične smjernice i kôd potreban za postavljanje i korištenje nadzorne ploče.

Rok za predaju rada: 28. lipnja 2024.

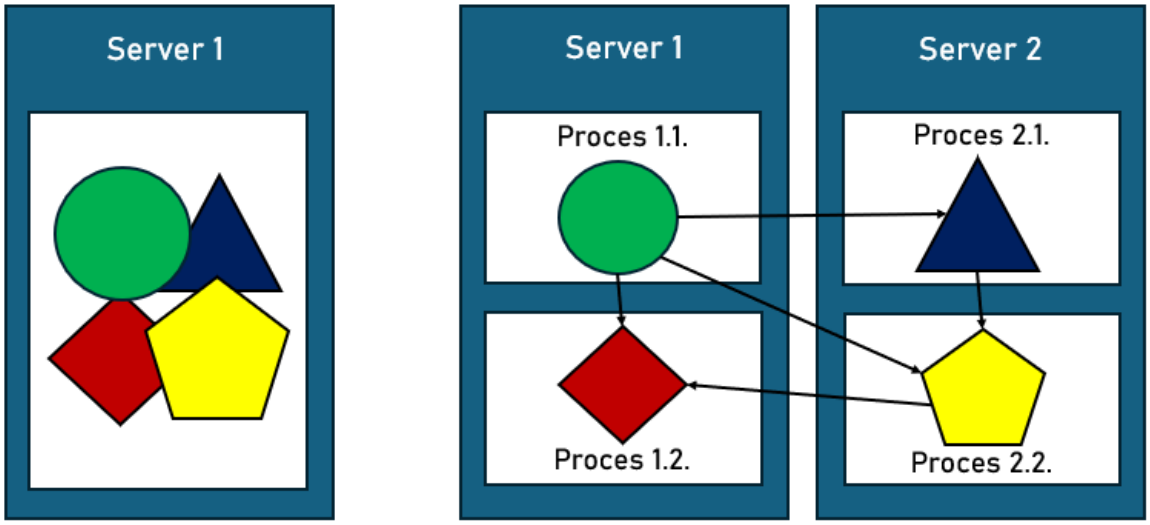
Sadržaj

1. Uvod

Tijekom proteklih godina Kubernetes je doživio strelovitи rast integracije i upotrebe unutar organizacija diljem svijeta. Njegova popularnost nedvojbeno je ubrzana porastom upotrebe i zahtjeva za kontejniziranim mikroservisima. Kada operativni, infrastrukturni i razvojni timovi dolaze do točke kada trebaju graditi, pokretati i održavati takve sustave, nerijetko se okreću Kubernetesu kao ključnom dijelu rješenja. S obzirom na konstantni rast složenosti sustava i broja aplikacija, korištenje alata za orkestraciju kontejnera kao što je Kubernetes kako bi se osiguralo automatsko upravljanje resursima zajedno s mogućnostima skaliranja i održavanja stabilnosti sustava postaje neophodna potreba i zahtjev visokog prioriteta. Ipak, zajedno s takvim sustavom koji pruža vrlu infrastrukturu za upravljanje i praćenje klastera dolaze popratno potreba i zahtjev pojednostavljenog i jasnog praćenja stanja za krajnje korisnike; gdje pod stanjem razmatramo sve resurse i ključne performance. Rješenje koje prozilazi iz nastalog problema je nadzorna ploča za Kubernetes klastere koja programerima iznimno pojednostavljuje upravljanje resursima pružajući bridak uvid u sve komponente sustava zajedno s njihovim trenutnim i proteklim statusima. Dakle, ideja iza takve ploče a ujedno i ovog rada je potreba za sučeljem uz pomoć kojeg se mogu rasporediti kontejnizirane aplikacije u klaster, otkloniti greške i upravljati resursima klastera zbog imperativa za osiguranje zahtjeva visoke dostupnosti i pouzdanosti aplikacija čije se izvršavanje odvija u Kubernetes okruženju.

2. Kubernetes

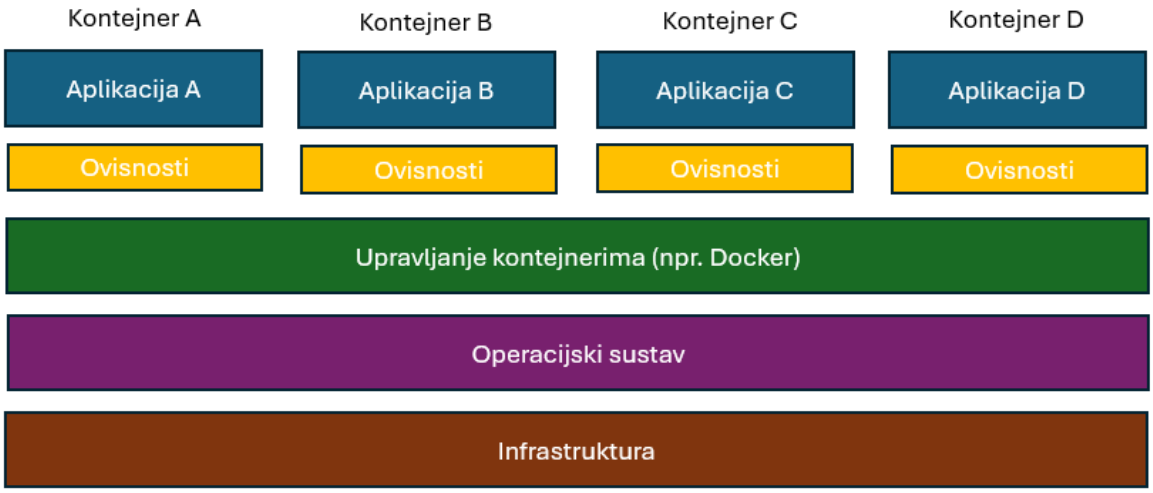
Tijekom godina većina je softverskih aplikacija bila monolitna te im se izvedba tvorila od jednog ili manjeg broja procesa raspoređenih na serverima. Danas se takve velike aplikacije vučene ostavštinom polako razbijaju u manje neovisne komponente zvane mikroserisi. Budući da su miksoservisi odvojeni jedni od drugih, oni se mogu individualno razvijati, implementirati i skalirati omogućujući pritom jednostavnu izmjenu komponenti koja se može prilagođavati po potrebi što je nužno za održavanje koraka s današnjim brzomijenjajućim poslovnim zahtjevima (Slika 1).



S rastućim brojem komponenti i sve većim podatkovnim centrima postalo je sveopće važno i zahtjevno konfigurirati i upravljati sistemom koji mora raditi glatko [13]. Ručno održavanje takvog sustava je glomazan zahtjev te je potrebna automatizacija koja uključuje vremensko zakazivanje komponenti na serverima, konfiguraciju, nadzor i upravljanje kvarovima. Ovdje nastupa Kubernetes.

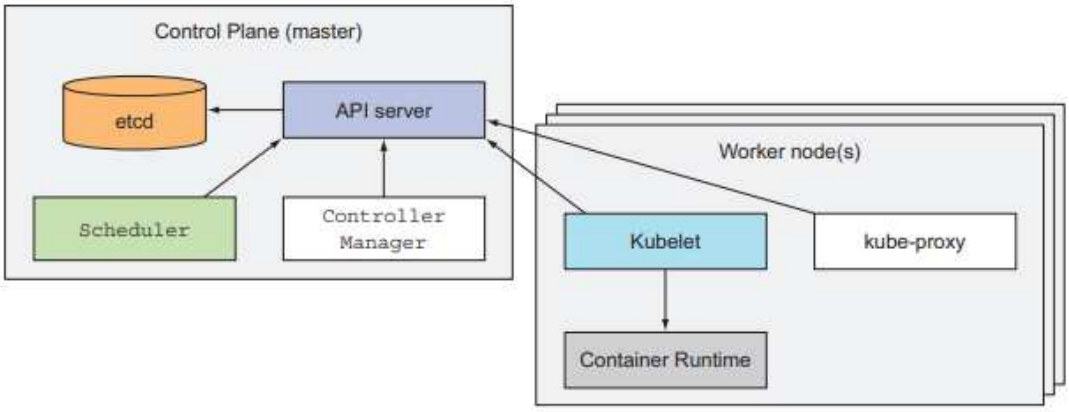
2.1. Kontejneri

Kada je aplikacija sačinjena od malog broja velikih komponenti, potpuno je prihvatljivo svakoj komponenti pridijeliti dedicirani virtualni stroj. Ipak, sa smanjenjem veličine komponenti i obrnuto proporcionalnim rastom njihovog broja poželjno je izbjeći dodjeljivanje virtualnog stroja svakoj komonenti u namjeri smanjivanja uzaludne potrošnje hardverskih resursa. Dodatno, osim samog problema povećane potrošnje resursa riječ je i o potrebi za konfiguriranjem i upravljanjem svakog pojedinačnog virtualnog stroja. Umjesto upotrebe virtualnih strojeva kako bi se izolirala okruženja svakog mikroservisa (ili općenito softverskog procesa) programeri se okreću Linuxovoj tehnologiji kontejnera koja omogućuje pokretanje više servisa na istom stroju, izlažući pritom drugačija okruženja svakom pojedinačno, a pritom ih izolira jednog od drugog (Slika 2) [4]. Iznimno je teško izgraditi aplikaciju satkanu od velikog broja malih dijelova. Tehnologija kontejnerizacije pruža priliku dizajniranja aplikacija uz pomoć mikroservisne arhitekture. Dakle, najvažniji osnovni koncept iz poglavlja je: „Kontejner je malo okruženje za izvođenje koje spakira aplikaciju zajedno sa svim ovisnostima koje su potrebne za pokretanje“ [2].



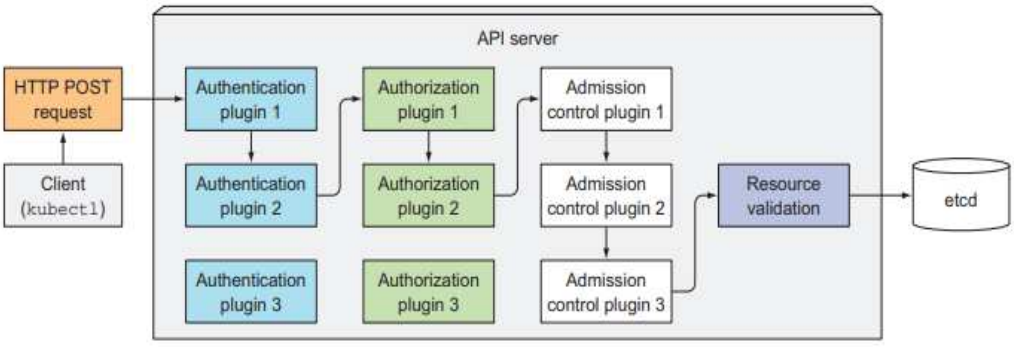
2.2. Osnovni koncepti Kubernetesa

Kubernetes je sustav koji dozvoljava jednostavno postavljanje i upravljanje kontejniziranim aplikacijama. Oslanja se na značajke Linuxovih kontejnera u smislu izvođenja heterogeniziranih aplikacija bez potrebe za posjedovanjem uvida i znanja o unutarnjim detaljima tih aplikacija i bez potrebe za ručnim postavljanjem tih aplikacija na svakom domaćinu. Kako se aplikacije vrte u kontejnerima, one ne utječu na druge aplikacije na istom serveru što je krucijalno za izvođenje aplikacija za potpuno drugačije organizacije na istom hardveru [6]. Kubernetes omogućava izvođenje aplikacija na tisućama čvorova koji skupa čine cjelinu poput jednog velikog računala. Apstrahira temeljnu infrastrukturu te time pojednostavnjuje razvoj, postavljanje i upravljanje kako za razvojne tako i operativne timove. Postavljanje aplikacija kroz Kubernetes je uvijek isto; neovisno je li riječ o nekoliko čvorova ili tisućama, dodatni čvorovi predstavljaju samo dodatne resurse dostupne aplikacijama [7]. Na hardverskoj razini, Kubernetes klaster se dakle sastoji od puno čvorova koji se mogu podijeliti na dvije vrste: *master* čvor koji je domaćin Kubernetesovoj upravljačkoj ravnini koja kontrolira i upravlja cijelim Kubernetes sistemom te na upravljačke čvorove koji zapravo vrte aplikaciju (Slika 3).

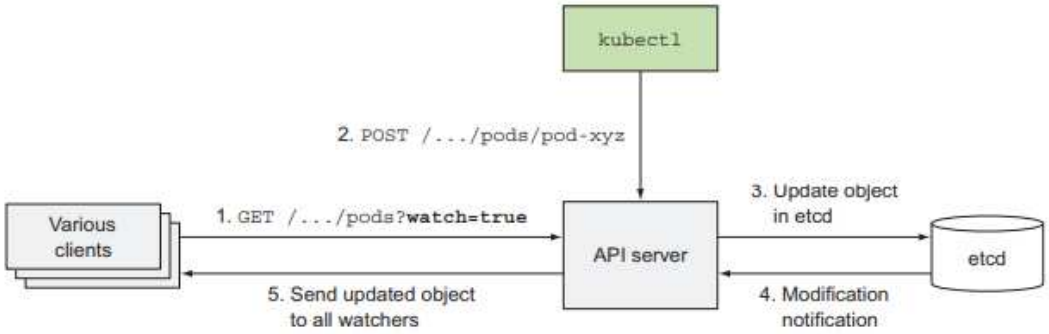


Kontrolna ravnina je dio koji kontrolira klaster i ono što ga čini funkcionalnim. Sastoji se od više komponenti koje se mogu izvršavati na jednom master čvoru ili mogu biti raspodijeljene

preko više čvorova i replicirane kako bi se osigurala visoka dostupnost. Komponente kontrolne ravnine su:



Slika 4. Operacije API poslužitelja [4]



Slika 5. Praćenje izmjena uz pomoć API poslužitelja [4]

- *Raspoređivač* – raspoređuje aplikacije; dodjeljuje radni čvor svakoj komponenti. Obično se ne navodi čvor klastera na kojem će se pod izvršavati, to je prepušteno raspoređivaču. Izdaleka se rad raspoređivača naizgled čini jednostavnim; sve što radi je čeka novokreirane podove putem mehanizma API poslužitelja za praćenje i dodjeljuje čvorove na koje će otići novi podovi koji nemaju postavljeni čvor. Raspoređivač ne daje upute određenom čvoru za pokretanje poda; sve što radi je ažuriranje definicije poda kroz API poslužitelj koji potom obavijesti Kubelet da je pod raspoređen. Čim Kubelet na krajnjem čvoru primijeti da je pod zakazan za taj čvor, pokrene proces stvaranja i izvršavanja kontejnera poda [4].
- *Upravitelj kontrolera* – obavlja funkcije na razini klastera, kao što su repliciranje komponenti, praćenje radnih čvorova i rukovanje kvarovima čvora. Kontroleri dakle mogu izvoditi različite radnje, ali svi oni promatraju API poslužitelj u potrazi za promjenima na resursima i izvode operacije za svaku izmjenu. Većinu vremena te

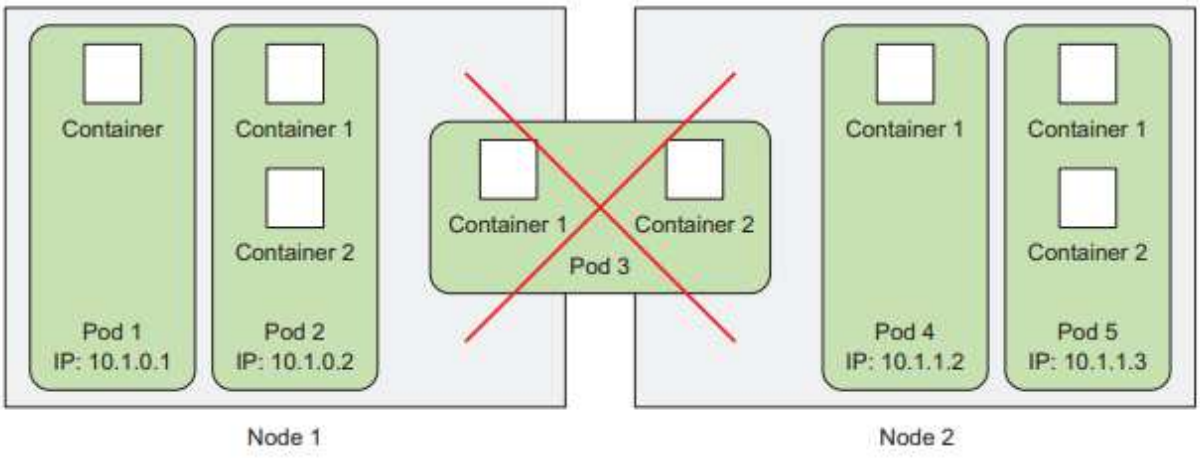
operacije uključuju kreiranje drugih resursa ili ažuriranje samih promatranih reursa (da bi se primjerice promijenio status objekta). Općenito, kontroleri pokreću petlju usklađivanja koja usklađuje stvarno stanje sa željenim stanjem određenim u sekciji speficikacija resursa i zapisuje novo stvarno stanje. Kontroleri koriste mehanizam promatranja da bi bili obaviješteni o promjenama, ali kako promatranje ne garantira da kontroler neće propustiti neki događaj, oni također izvode operaciju ponovnog popisivanja kako bi bili sigurni da nisu ništa propustili

- *Etcd* – pouzdana distribuirana pohrana podataka koja sadrži konfiguraciju klastera. Svi kreirani objekti trebaju biti negdje pohranjeni na održiv i postojan način kako bi manifesti preživjeli ponovna pokretanja i kvarove API poslužitelja. Etcd je brza, distribuirana i konzistentna pohrana tipa ključ-vrijednost. S obzirom da je distribuirana, može biti pokrenuto više od jedne etcd instance kako bi se osigurala visoka dostupnost i bolje performance. Jedina komponenta s kojom ectd direktno komunicira je API poslužitelj; sve druge komponente čitaju i pišu podatke posredno preko API poslužitelja.

Komponente kontrolne ravnine upravljaju stanjem klastera, ali ne pokreću aplikacije; to je djelo radnih čvorova.

2.3. Podovi

Podovi su centralni i najvažniji koncept Kubernetesa. Sve ostalo upravlja, izlaže ili je korišteno od strane podova. Podovi su grupe zajedno smještenih kontejnera i predstavljaju osnovni gradivni blok u Kubernetesu. Umjesto ručnog pojedinačnog postavljanja kontejnera, uvijek se postavlja i radi na podu kontejnera; gdje kontejnera može biti više ili samo jedan na jednom podu. Kada pod sadrži više kontejnera svi se oni uvijek izvode na jednom radnom čvoru (Slika 6).



Slika 6. Prikaz izvođenja podova na čvorovima [4]

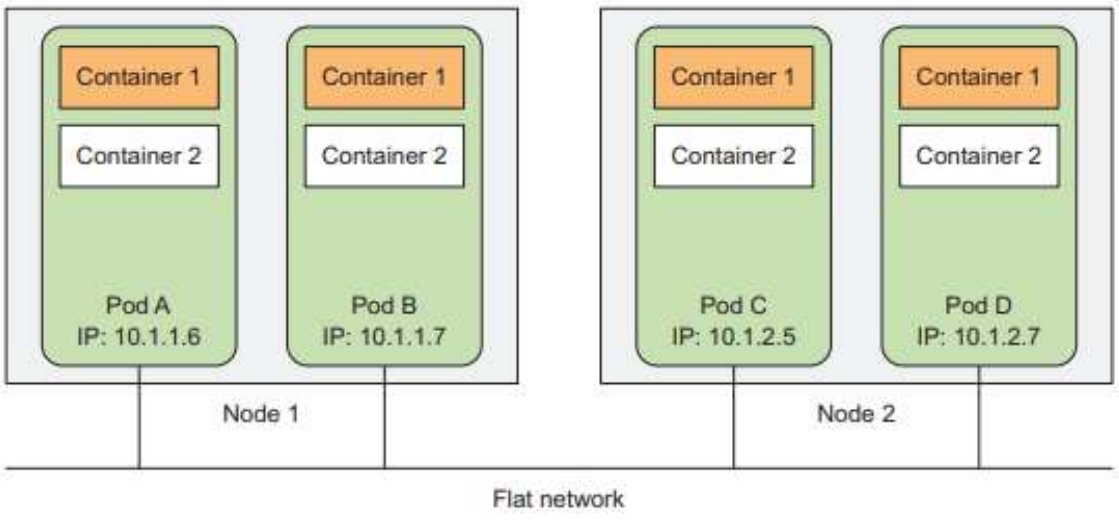
Postavlja se pitanje zašto su uopće potrebni podovi i zašto se kontejneri ne koriste izravno; zašto bi se uopće izvodio veći broj kontejnera skupa? Kontejneri su zamišljeni da izvršavaju jedan proces po kontejneru. Ako se više nepovezanih procesa izvodi na jednom kontejeru, sva odgovornost održavanja svih procesa, upravljanja zapisnikom i slično otpada na programera; gdje bi morao uključiti mehanizam za automatsko ponovno pokretanje individualnih procesa u slučaju grešaka i uz velike poteškoće morao pratiti zapise više procesa na istom standardnom izlazu. Kako izvršavanje više procesa nije zamišljeno na jednom kontejneru, vidno je potrebna druga konstrukcija više razine koja će omogućiti povezivanje kontejnera i upravljanje njima kao jednom jedinicom; što čini obrazloženje potrebe za podovima.

Pod kontejnera dozvoljava zajedničko izvođenje blisko povezanih odnosa i pružanje gotovo istog okruženja kao da se svi izvode u istom kontejner, a ipak ih istovremeno držeći izoliranima. Na taj način moguće je iskoristiti sve prednosti koje značajke kontejnera pružaju, a da se istovremeno pruža iluzija jedinstvenog izvođenja. Želja je da kontejneri unutar svake grupe dijele određene resurse, iako ne sve, tako da oni nisu u potpunosti izolirani. Kubernetes navedeno postiže konfiguriranjem Dockera na način da svi kontejneri poda dijele isti set Linuxovih prostora imena umjesto da svaki kontejner ima set vlastitih. Budući da svi kontejneri poda se izvode pod istom mrežom i UTS prostorima imena (ovdje je riječ o prostorima imena

Linuxa), svi oni dijele ime domaćina i mrežna sučelja. Slično, svi kontejneri poda pod istim IPC prostorima imena mogu komunicirati kroz IPC.

Ipak, na razini datotečnog sustava, stvari su malo drugačije. Budući da većina datotečnih sustava kontejnera dolazi od slike spremnika, oni su svi međusobno potpuno izolirani. Međutim, moguće je imati zajedničke direktorije datoteka koristeći Kubernetes *Volume* koji predstavlja objekt koji omogućuje trajnije ili zajedničko spremanje podataka za aplikacije unutar kontejnera.

Jedna stvar koju je potrebno naglasiti je da zato što kontejneri rade u istom podu u istom mrežnom prostoru imena, oni također dijele istu IP adresu i prostor vrata. Dakle, procesi koji se vrte u kontejnerima istog poda trebaju brinuti da se ne vežu na ista vrata ili će doći do konflikta. Svi kontejneri unutar istog poda također dijele isto povratno mrežno sučelje, tako da kontejner može komunicirati sa svim drugim kontejnerima unutar poda preko lokalnog računala. Svi podovi u Kubernetes klasteru su u jednom ravnom zajedničkom mrežnom prostoru (Slika 7), što znači da svaki pod može pristupiti svakom drugom podu na njegovoj IP Adresi; ne postoji NAT pristupnik između njih. Kada dva poda pošalju mrežne pakete jedno drugom, oni zapravo vide prave IP adrese jedne drugih kao izvorni IP u paketu.



Slika 7. Međusobna komunikacija podova [4]

2.4. K3s

K3s je certificirana Kubernetes distribucija koju je kreirao Rancher Labs. K3s ne sadrži nikakve dodatne značajke koje nisu vitalne za upotrebu s Kubernetesom, ali one mogu biti dodane kasnije. K3s pruža istu moć kao i Kubernetes ali u malom optimiziranom paketu obogaćenom sa značajkama dizajniranim posebno za računalne sustave na rubu. Dodatna je prednost K3s-a jednostavnost korištenja u usporedbi s drugim distribucijama Kubernetesa. Navedena distribucija je osim za sustave na rubu vrsna i za internet stvari, kontinuiranu integraciju i računala s jednom pločom (ARM). Dodatna poboljšanja koja sadrži K3s distribucija uključuju [3]:

- Distribucija kroz jednu binarnu ili minimalnu sliku kontejnera
- Lagana pohrana podataka temeljena na sqlite3 kao zadanoj pohrani; druge dodatne mogućnosti uključuju MySQL i Postgres
- Jednostavni pokretač koji rukovodi kompleksnošću TLS-a i drugih opcija
- Osigurana sigurnost prema zadanim postavkama s razumnim postavkama za jednsotavna okruženja
- Rad svih komponenti kontrolne ravnine Kubernetesa je inkapsuliran u jednu binarnu datoteku/proces, što omogućuje K3s distribuciji automatizaciju i upravljanje složenim operacijama klastera poput distribucije certifikata
- Vanjske ovisnosti su minimizirane; jedini zahtjevi su moderna jezgra i postavljanje cgroupa.

Kubeconfig datoteka koja se koristi za konfiguriranje pristupa Kubernetes klasteru je pohranjena na lokaciji *„/etc/rancher/k3s/k3s.yaml“.* Ako su prethodno instalirani Kubernetosovi alati naredbenog retka kao što su kubectl ili helm potrebno je postaviti ispravnu stazu na kubeconfig datoteku postavljanjem *KUBECONFIG* varijable okruženja na ispravnu putanju pokretanjem naredbe *export KUBECONFIG=/etc/rancher/k3s/k3s.yaml.*

2.5. Kubectl

Kubectl je alat naredbenog retka koji se može pokretati na osobnom računalu, CI/CD cjevovodima, kao dio operacijskog sustava ili pak kao Docker slika [5]. Kubectl traži konfiguracijsku datoteku imena *.kube* u *$HOME* mapi; u toj datoteci kubectl ima spremljenu konfiguraciju klastera potrebnu za pristup Kubernetes klasteru. Također se može eksplicitno postaviti *KUBECONFIG* varijabla okruženja na željenu datoteku. Neke od važnijih kubectl naredbi koje su u nadolazećem odjeljku poredane slijedno za potrebe praćenja klastera i kreacije nadzorne ploče (s iznimkom *helm* naredbi [11]) uključuju:

1. *kubectl cluster-info* – prikaz informacija o Kubernetes klasteru; uključujući glavnu i servisnu adresu
2. *kubectl create namespace kubernetes-dashboard* – stvaranje novog prostora imena
3. *kubectl get namespaces* – prikaz svih prostora imena (Slika 8)

```
user@DESKTOP-TF19M53:~$ kubectl get namespaces
NAME                   STATUS   AGE
default                Active   5d7h
kube-node-lease        Active   5d7h
kube-public            Active   5d7h
kube-system            Active   5d7h
kubernetes-dashboard   Active   5d4h
```

Slika 8. Prikaz imena prostora

4. *kubectl config set-context --current --namespace=kubernetes-dashboard* – odabir i postavljanje imena prostora
5. *kubectl get nodes* – dohvat liste čvorova
6. *helm repo add kubernetes-dashboard https://kubernetes.github.io/dashboard/* - dodavanje repozitorija za Kubernetes nadzornu ploču
7. *helm upgrade --install kubernetes-dashboard kubernetes-dashboard/kubernetes-dashboard --create-namespace --namespace kubernetes-dashboard* – naredba koja

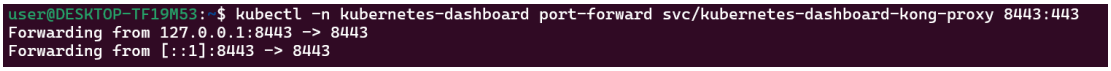
instalira ili ažurira Kubernetes nadzornu ploču u prostoru imena kubernetes-dashboard, a ako taj ne postoji automatski ga kreira

8. *kubectl get pods --namespace=kubernetes-dashboard* – izlistavanje liste podova i njihovih statusa (Slika 9)

```
user@DESKTOP-TF19M53:~$ kubectl get pods --namespace=kubernetes-dashboard
NAME                                         READY   STATUS    RESTARTS        AGE
dashboard-metrics-scraper-795895d745-cgz2s   1/1     Running   0               33m
kubernetes-dashboard-56cf4b97c5-wfl65        1/1     Running   0               33m
kubernetes-dashboard-api-5f55b4dfdb-q6vb9    1/1     Running   5 (3h58m ago)   5d4h
kubernetes-dashboard-auth-56d46b7696-4sdz9   1/1     Running   5 (3h58m ago)   5d4h
kubernetes-dashboard-kong-7696bb8c88-k55vl   1/1     Running   5 (3h58m ago)   5d4h
kubernetes-dashboard-metrics-scraper-5485b64c47-pfkrv   1/1     Running   5 (3h58m ago)   5d4h
kubernetes-dashboard-web-84f8d6fff4-wvs9g    1/1     Running   5 (3h58m ago)   5d4h
```

Slika 9. Lista podova

9. *kubectl describe pod kubernetes-dashboard-web-84f8d6fff4-wvs9g --namespace=kubernetes-dashboard* – detaljan opis poda; najčešće za potrebe otklanjanja grešaka
10. *kubectl get nodes* – prikaz svih čvorova
11. *kubectl describe node desktop-tf19m53* – detaljan opis čvora; najčešće za potrebe otklanjanja grešaka
12. *kubectl create serviceaccount dashboard-admin-sa -n kubernetes-dashboard* – stvara novi račun usluge u određenom prostoru imena
13. *kubectl create clusterrolebinding dashboard-admin-sa --clusterrole=cluster-admin --serviceaccount=kubernetes-dashboard:dashboard-admin-sa* – dodjeljuje cluster-admin ulogu računu usluge u određenom prostoru imena; uloga koja ima potpuni pristup svim resursima u klasteru; potrebna kako bi račun koji pristupa nadzornoj ploči imao pristup svim resursima
14. *kubectl -n kubernetes-dashboard create token dashboard-admin-sa* – generiranje API tokena za račun usluge koji je potreban za autentifikaciju u Kubernetes klasteru; potrebno je spremiti generirani token za buduće autentifikacije
15. *kubectl -n kubernetes-dashboard port-forward svc/kubernetes-dashboard-kong-proxy 8443:443* – preusmjeravanje vrata 8443 s lokalnog računala na vrata 443 servisa kubernetes-dashboard-kong-proxy (Slika 10)



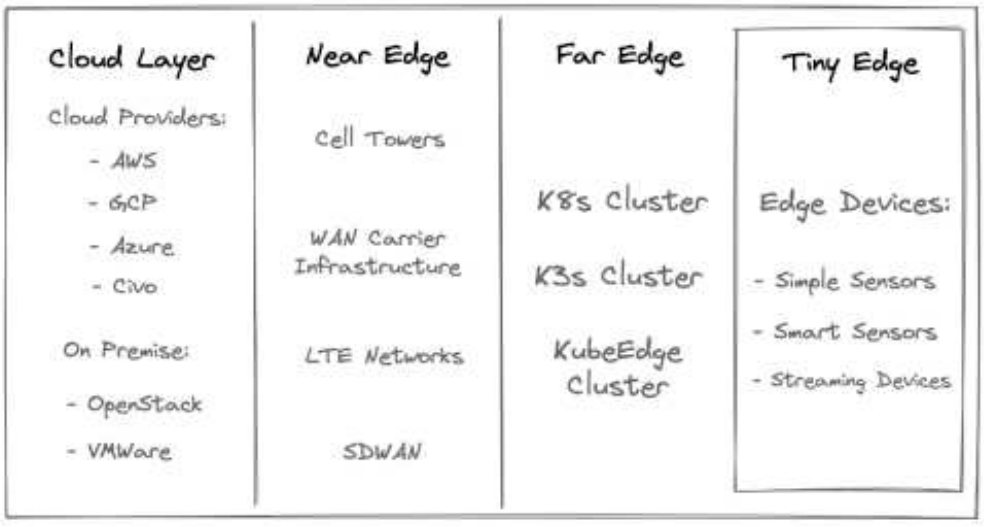
Slika 10. Prosljeđivanje vrata

3. Računarstvo na rubu

S evolucijom oblaka kompanije i organizacije su počele migrirati svoje zadatke procesiranja na računalne uređaje na rubu; sve s ciljem smanjenja troškova i dobivanja više koristi iz plaćene infrastrukture. „Rub“ se može definirari kao bilo što gdje se podatci obrađuju prije nego što prijeđu WAN, dok se računarstvo na rubu može shvatiti kao obrada i analiza podataka duž ruba mreže, nabliže točki njihova prikupljanja tako da podatci postanu djelotvorni [2]. Dakle, računarstvo na rubu se odnosi na obradu podataka u blizini izbora i distribuciju računanja na različitim mjestima koristeći uređaje koje su blizu podataka.

Slika 11 pokazuje kako se podatci obrađuju u različitim kontekstima:

- Sloj oblaka: pružatelji usluga u oblaku kao što su AWS, Azure i GCP
- Blizu ruba: telekomunijskacija infrastruktura i uređaji
- Daleki rub: klasteri na rubu kao što su K3s klasteri ili uređaji koji razmjenjuju podatke između oblaka i rubnog sloja; ovaj sloj se dalje može podijeliti na sitni rubni sloj
- Sitni rub: senzori, uređaji krajnjih korisnika koji razmjenjuju podatke s uređajem koji obrađuje podatke, rubni klasteri na dalekom rubu



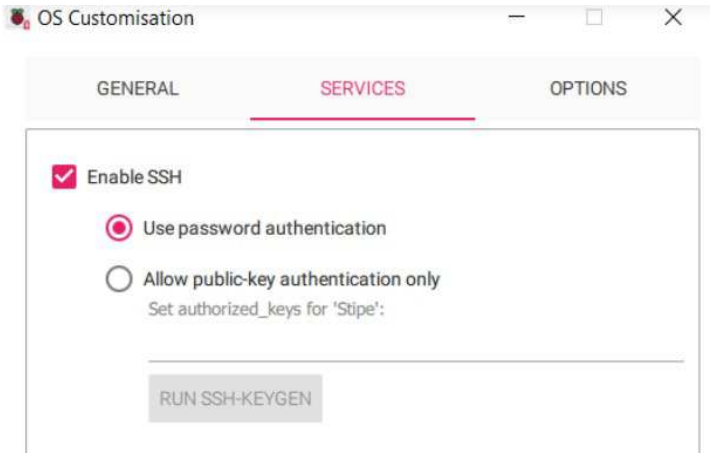
Slika 11. Prikaz slojeva [2]

Neki od glavnih benefita računarstva na rubu uključuju:

- Smanjenje latencije: računarstvo na rubu može obraditi teške računalne procese na rubnim uređajima smanjujući pritom latenciju za donošenje informacija
- Smanjenje propusnosti: rubno računarstvo može smanjiti propusnost dok koristi dio podataka na rubnim uređajima smanjujući posljedično promet na mreži
- Smanjenje troškova: smanjenje latencije i propusnosti za posljedicu sadrži smanjenje operativnih troškova
- Poboljšanje sigurnosti: rubno računarstvo koristi agregaciju podataka i algoritme za enkripciju podataka kako bi se poboljšala sigurnost pristupa podatcima

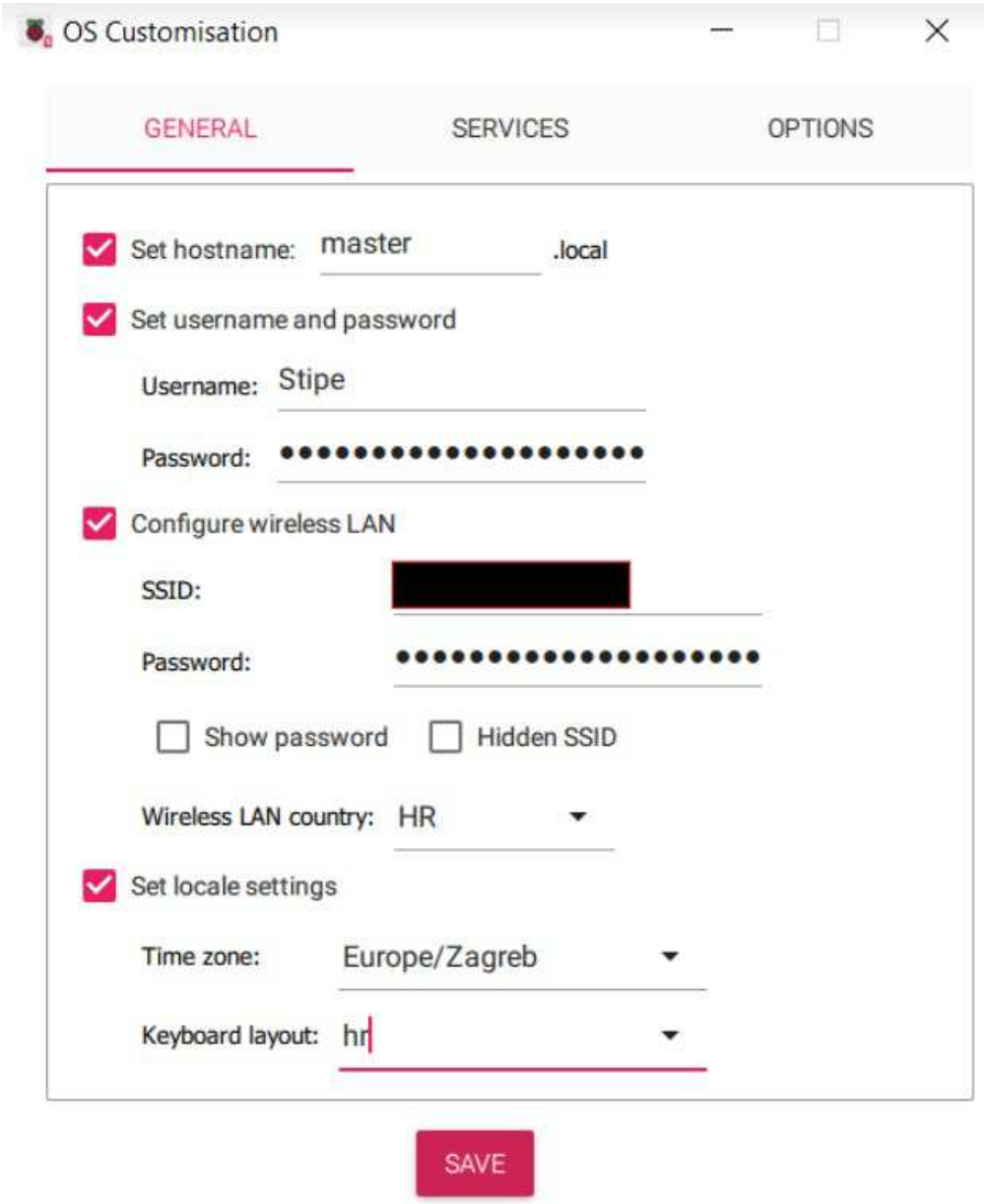
Za izradu projekta potrebna je sljedeća konfiguracija:

- Raspberry Pi 4 Model B 4GB
- Napajanje (5V i 3A)
- Ethernet kabel (Opcionalno, ovisno o OS-u može se upotrijebiti Wi-Fi)
- Mikro HDMI na HDMI kabel (Opcionalno, pristup omogućen SSH-om)
- MicroSD kartica
- Čitač MicroSD kartica
- Tipkovnica i miš (Opcionalno, pristup moguć s lokalnog kompjutera putem SSH koji je potrebno prethodno omogućiti (Slika 12))



Slika 12. Omogućenje pristupa SSH-om

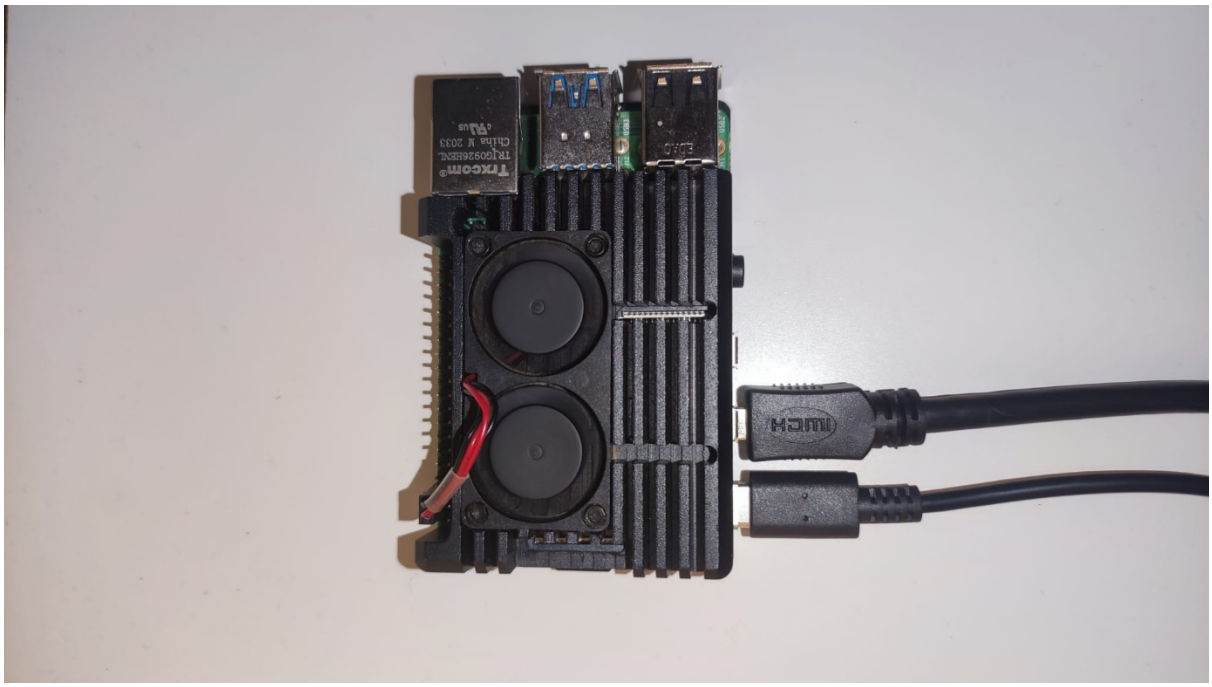
Za potrebe eksperimenta upotrijebljen je Raspberry Pi OS (64-bit) operacijski sustav s navedenom konfiguracijom konfiguracijom (Slika 13).



Slika 13. Dodatna konfiguracija operacijskog sustava

Sljedeći korak je pokretanje uređaja te instalacija kubectla i podizanje K3s klastera. Zbog jednostavnosti računalu se pristupalo uz pomoć SSH naredbom *ssh korisnik@domaćin.local*,

odnosno u konkretnom slučaju *Stipe@master.local*, gdje umjesto domaćin.local također može stajati IP adresa računala. Po ulasku u računalo nakon instalacije svih preduvjeta zanima nas kako pristupiti stvorenom klasteru sa svog lokalnog računala; da bismo navedno ostvarili potrebno je otkriti sadržaj *KUBECONFIG* datoteke kojem možemo pristupu pozivajući *kubectl config view -–raw* , gdje je parametar –raw potreban kako bi se mogli iščitati podatci koji bi inače bili skriveni (npr. certifikati). Kako bi lokalno računalo moglo pristupiti K3s klasteru na Raspberry Pi uređaju (Slika 14) potrebno je *KUBECONFIG* datoteku s uređaja prenijeti te spojiti u *KUBECONFIG* lokalnog računala.



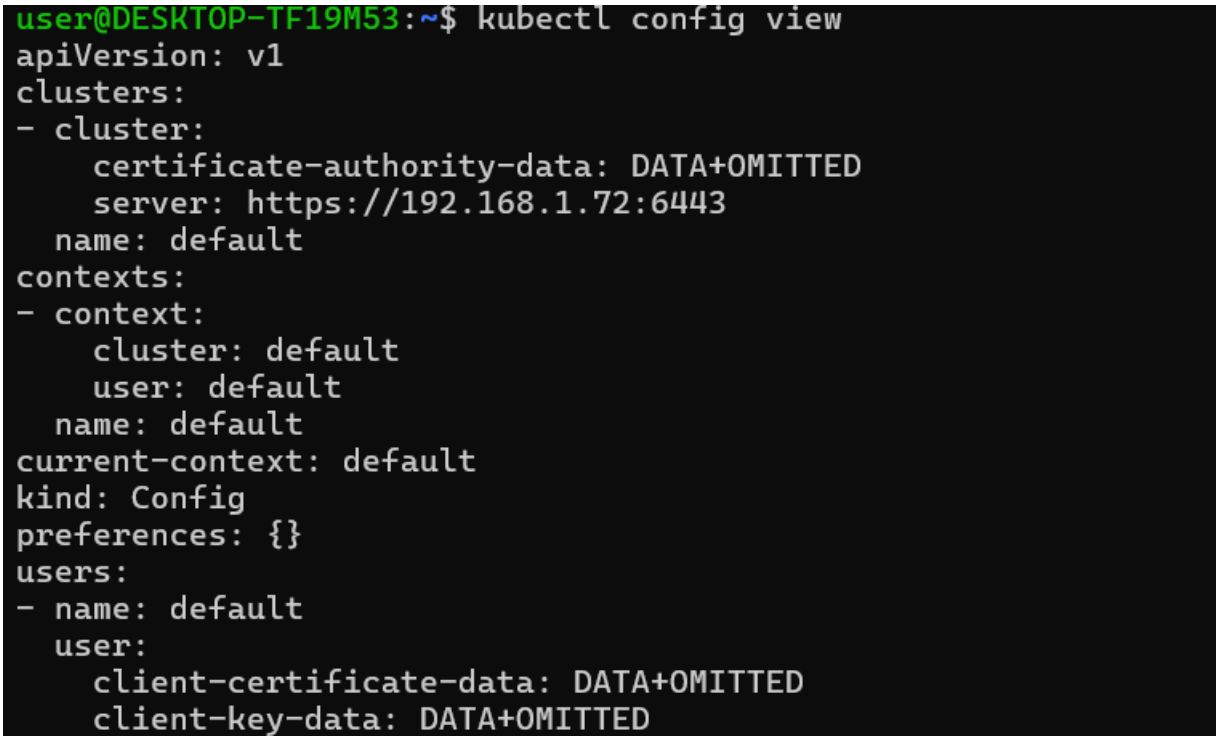
Slika 14. Raspberry Pi 4 uređaj

Kako bi se to ostvarilo jedna od mogućnosti je na varijablu okruženja *KUBECONFIG* nadovezati *YAML* datoteku K3s klastera s Raspberry Pi uređaja; ali s izmjenom parametra server gdje je porebno upisati IP adresu tog računala. Navedena datoteka se sastoji od četiri dijela (Slika 14):

- Lista klastera – unos koji predstavlja Kubernetes klaster te sadrži URL API poslužitelja, datoteke ovlaštenja za izdavanje certifikata (CA) te moguće drugih konfiguracija povezanih s komunikacijom s API poslužiteljem. CA certifikat može biti pohranjen u

zasebnoj datoteci i biti referenciran u *KUBECONFIG* datoteci ili se može izravno unijeti u polje

- Lista korisnika – svaki korisnik definira certifikate koje će koristiti kada komunicira s API poslužiteljem. Ovo može bii par korisničkog imena i lozinke, autentifikacijski token ili klijentov ključ i certifikat (kao što je slučaj u navedenom primjeru); kao i u prethodnom slučaju vrijednosti se mogu dobavljati iz drugih datoteka ili one mogu biti eksplicitno napisane
- Kontekst – kontekst povezuje klaster, korisnika i zadani prostor imena koji kubectl treba koristiti prilikom izvođenja naredbi; višestruki konteksti mogu ukazivati na istog korisnika ili klaster
- Trenutni kontekst – Iako u datoteci može biti definirano više konteksta, u bilo kojem trenutku samo jedan od njih može biti trenutni.



Slika 15. Konfiguracija klastera

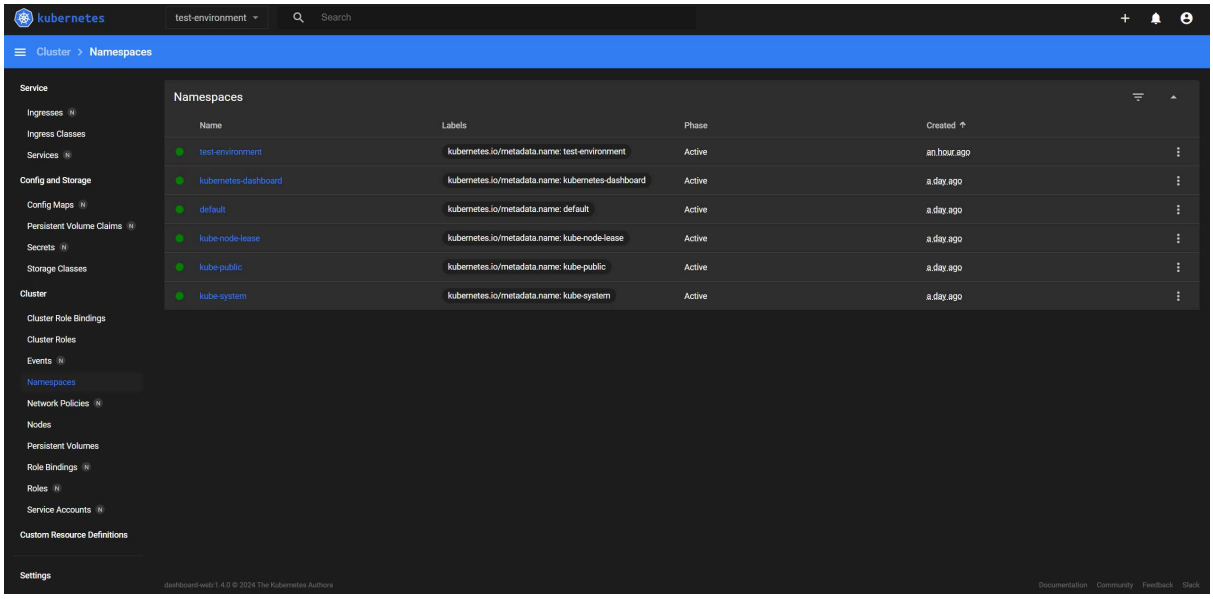
Nakon što je kontekst postavljen na željeni klaster Kubernetes nadzorna ploča je spremna za podizanje.

3.1. Kubernetes nadzorna ploča

Kubernetes nadzorna ploča je web-bazirano korisničko sučelje za Kubernetes. Može se koristiti za pregled aplikacija koje se izvode na klasteru, postavljanje kontejniziranih aplikacija na Kubernetes klaster i upravljanje resursima klastera. Također omogućuje rješavanje problema za kontejnizirane aplikacije pružajući informacije o zdravlju Kubernetes resursa i svim pogreškama koje su se dogodile. Kuberntes nadzorna ploča omogućuje stvaranje ili izmjenu pojedinačnih Kubernetes resursa kao što su *deploymenti*, poslovi, *DaemonSetovi* i *StatefulSetovi*. Također se može koristiti za izravno upravljanje Kubernetes podovima. Vrativši se na Kubectl poglavlje potrebne su nam prvo šesta i sedma naredba kako bi preuzeli i instalirali nadzornu ploču, a potom i naredbe pod rednim brojem dvanaest i trineast zbog kreiranja računa usluge i postavljanja njegovih prava, te naposljetku naredba četrnaest koja generira token potreban za autentifikaciju nakon naredbe petnaest kojom se omogućuje pristup servisu.

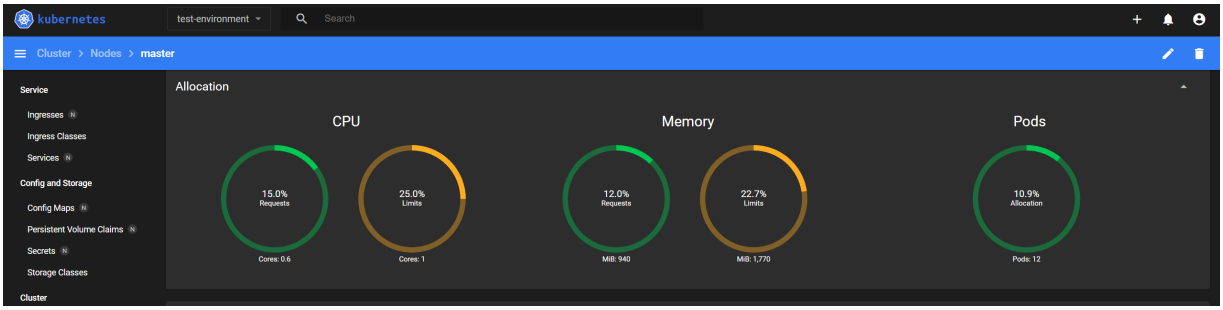
U ovom poglavlju za potrebe boljeg prikaza implementirat će se jednostavni Kubernetes resursi (uz već postojeće) kako bi se olakšalo istraživanje nadzorne ploče. Koristit će se jednostavan primjer postavljanja Nginx web poslužitelja unutar Kubernetes klastera. Prvo se kreirao deployment u testnom prostoru imena s dvije replike Nginx kontejnera. Takva postavka osigurava dostupnost aplikacije i priprema je za vizualizaciju na nadzornoj ploči. Uz *deployment* stvoren je i popratni servis u istom prostoru imena. Servis je konfiguriran kao *LoadBalancer* koji izlaže vrata 80 i prosljeđuje promet Nginx kontejnerima. Ova konfiguracija omogućuje vanjski pristup aplikaciji i integrira je u Kubernetesov mrežni sloj. Glavni cilj ove konfiguracije je pružiti jednostavan primjer koji može biti vizualiziran i upravljan putem Kubernetes nadzorne ploče. Postavljanjem i izlaganjem Nginx aplikacije mogu se učinkovito demonstrirati i istražiti značajke nadzorne ploče za Kubernetes, uključujući mogućnost praćenja resursa i upravljanja.

U odjeljku naziva *Cluster* možemo naići na osnovne informacije poput o prostorima imena, čvorovima, volumenima ulogama i klasama pohrane. Klikom na pododjeljke preusmjerava se pregled na odabrano korisničko sučelje. Podpregled prostora imena daje uvid u svaki prostor imena u klasteru (Slika 16). Odabirom prostora imena odvije se usmjeravanje na namjenski prikaz tog prostora imena. Trenutačno ovaj prikaz prikazuje samo nedavne događaje, ali se može pristupiti drugim informacijama specifičnim za prostor imena u općem pregledu.



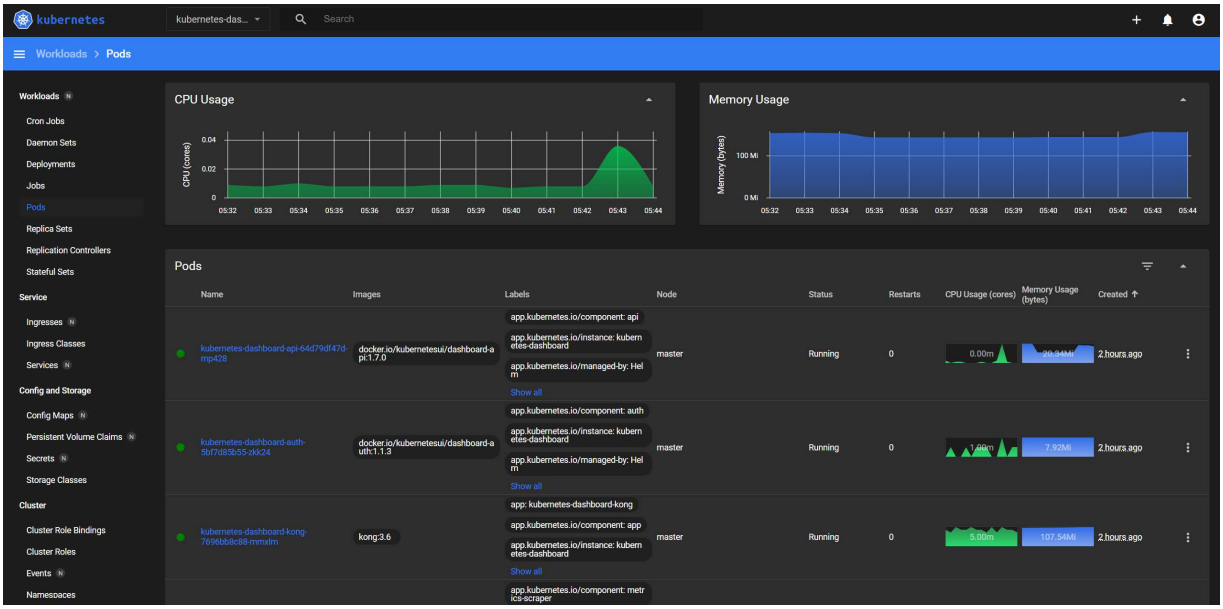
Slika 16. Prikaz untuar odjeljka Klaster

Odabirom određenog čvora u odjeljku događa se preusmjeravanje na detaljnu stranicu za taj čvor. Ta stranica sadrži nekoliko odjeljaka koji pružaju opsežne informacije o čvoru kao što su ID stroja, adrese, dodijeljeni resurse, podovi, uvjeti i događaji. Odjeljak o alokaciji sadrži tri prikaza: dodjelu CPU-a, memorije i podova (Slika 17). Prikazi alokacije memorije i CPU-a određuju memoriju čvora i kapacitet CPU-a uz dodatak ukupnom broju CPU ograničenja i zahtjeva za sve podove koje se izvršavaju na čvoru. Ti prikazi predstavljaju brojeve izražene u postotcima i u apsolutnom iznosu. Alokacija podova pak prikazuje ukupan broj podova koje čvor može podupirati i broj živih podova.



Slika 17. Prikazi alokacija u čvoru

Prikaz radnih opterećenja nudi pregled svih aplikacija koje se izvode u klasteru, uključujući *deploymente*, podove, skupove replika i druge Kubernetes kontrolere. Nakon odabira opcije *Pods* izlistava se pregled svih podova koji se izvode u klasteru (Slika 18). Ova stranica prilaže sve važne informacije o individualnim podovima, uključujući čvor, prostor imena, status i broj ponovnih pokretanja. Može se odabrati specifični pod kako bi se dobio detaljni pregled; u takvom pregledu mogu se vidjeti njegove priložene oznake, QoS klasa i status. Također pokazuje kojem kontejneru pod pripada, njegovo stanje i kontroler koji ga je kreirao.



Slika 18. Pregled podova

Odjeljak *Services* sadrži podatke o otkrivanju i uravnoteženju opterećenja. Sastoji se od dva Kubernetes objekta: servisa i ulaza. Odjeljak servisa pruža najvažnije podatke poput

prostora imena, priloženih oznaka i IP adresa. Klikom na servis se dolazi u dedicirani pregled gdje se može vidjeti tip servisa, birače oznaka i liste servisnih krajnjih točaka, podova i događaja.

Odjeljak *Config and Storage* pruža informacije o tajnama, mapama konfiguracije i zahtjevima za trajni volumen (volumen, status, kapacitet, klasa pohrane, način pristupa).

4. Flutter

Flutter je Googleov SDK za razvoj višeplatformskih aplikacija. Flutter aplikacije se sastoje od niza paketa, dodataka i sučeljnih programa. Pruža jednostavan način podizanja i izvođenja aplikacija na više platformi. Flutter nije programski jezik već skup alata koji omogućuje jednsotavno stvaranje aplikacija koje rade za iOS. Flutter koristi Dart koji je također Googleov proizvod. Neke od važnijih prednosti korištenja Fluttera uključuju [10]:

- Flutter je SDK otvorenog koda; odnosno, lako je pratiti njegovu evoluciju i znati što je sljedeće, pa tako i isprobavati nove značajke u razvoju, kreirati vlastite zakrpe, pakete ili doprinositi kodu
- Trenutno učitavanje – značajka koja omogućuje pravljenje izmjena u kodu uz trenutno praćenje izmjena na emulatoru/uređaju bez potrebe za ponovnim prevođenjem aplikacije
- Trenutno ponovno pokretanje – pri većim izmjenama potrebna je adekvatna nadogradnja značajke trenutnog učitavanja; ona preuzima oblik trenutnog ponovnog pokretanja koja je nešto sporija od trenutno učitavanja, ali ipak brža od potpunog ponovnog pokretanja
- Flutter posjeduje atraktivan dizajn – dolazi sa sjajnim animacijama i prijelazima, a svatko može izraditi vlastite widgete te je korisničko sučelje je u potpunosti prilagodljivo.

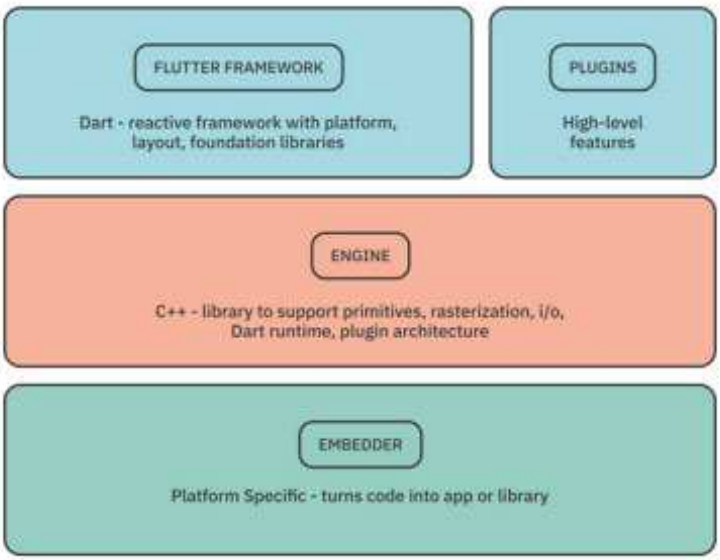
4.1. Arhitektura Fluttera

Flutter ima modularnu slojevitu arhitekturu. To omogućuje pisanje logike aplikacije jednom, a da ponašanje bude dosljedno preko svih platformi, iako je temeljni kod drugačiji ovisno o platformi [9]. Slojevita arhitektura također izlaže različite točke za prilagođavanje i nadjačavanje prema potrebi. Fluterr arhitektura se sastoji od tri glavna sloja (Slika 19) [12]:

- Razvojni okvir – razvojni okvir je napisan u Dartu i sadrži biblioteke koje se mogu izravno koristiti za izradu aplikacija. To uključuje temu grafičkog sučelja, widgete, izgled, animacije i temeljnje gradivne blokove. Uz glavni Flutter razvojni okvir tu su i

dodatci: značajke visoke razine poput JSON serijalizacije, geolokacija, pristup kameri i plaćanja unutar aplikacije. Ova arhitektura koja se temelji na dodatcima dopušta uključivanje dodataka po potrebi.

- Sloj motora – sadrži osnovne C++ biblioteke koje čine primitive koji podupiru Flutter aplikacije. Motor implementira primitivne niske razine Flutter API-ja, poput I/O grafika, rasporeda teksta, pristupačnosti i Dart izvođenja. Motor je također odgovoran za rasterizaciju Flutter scena za brzo renderiranje na zaslonu.
- *Embedder* – različit za svaku ciljanu platformu i bavi se pakiranjem koda kao samostalne aplikacije ili građenog modula.



Slika 19. Arhitektura Fluttera [12]

Svaki od slojeva arhitekture sastoji se od drugih podslojeva i modula, čineći ih gotovo fraktalnima.

4.2. Dart

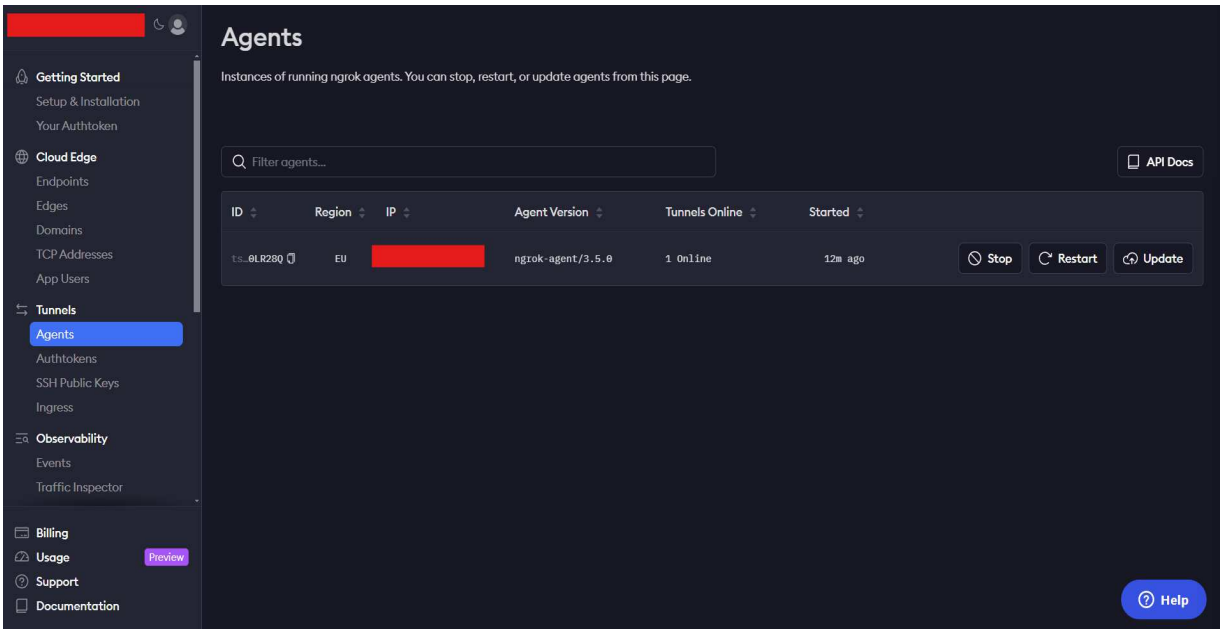
Dart je objektno orijentirani jezik iz Googleove radionice s potporom za reaktivno i asinkrono programiranje. Rukovanje Flutterom uključuje bitno poznavanje Darta kao jezika; kod aplikacije, kod dodataka i upravljanje ovisnostima koriste Dart i njegove značajke. Temelji se na zajedničkim značajkama i strukturama mnogih postojećih jezika što znači da je jednostavan za prelazak s drugih jezika. Važnije značajke i prednosti Darta s Flutterom uključuju [1]:

- Prevođenje – Dart je iznimno fleksibilan i pruža razne načine izvođenja aplikacije ovisno o okolnostima. Tijekom faze testiranja koristi gotovo trenutno JIT prevođenje zbog bržeg odraza na izmjene, dok se tijekom faze produkcije koristi AOT prevođenje čija je glavna bit izvrnost performanci.
- Visoke performance – Budući da Dart podupire AOT prevođenje, Flutter ne zahtijeva sporiji most između dviju ovlasti (primjerice nenativnog i nativnog Flutter koda) što čini aplikacije responzivne s brzim pokretanjem.
- Prikupljanje smeća – Flutter koristi funkcionalni stil s objektima kratkog životnog vijeka što podrazumijeva hrpu kratkoživeće alokacije. Dart pritom upravlja kolekcijom smeća bez zaključavanja rezultirajući time brzom alokacijom.
- Deklarativno korisničko sučelje – U kontrastu s prevladavajućim imperativnim stilom gdje se promjene specifične komponente moraju eksplicitno navesti, u Flutteru se koristi deklarativni stil što znači da su widgeti nepromjenjivi i zapravo samo lagani nacrti. Da bi se promijenilo korisničko sučelje, widget sam aktivira ponovnu izgradnju kreirajući pritom novi nacrt koji se uglavnom naslanja na prethodni s manjim izmjenama.

4.3. Ngrok

Ngrok je globalno distribuirani obrnuti proxy koji štiti, osigurava i ubrzava aplikacije i mrežne usluge bez obzira odakle su pokrenute. Ngrok je neovisan o okruženju i može isporučiti promet servisima pokrenutima bilo gdje bez mrežnih lokalnih promjena okruženja. Dakle, bilo da je aplikacija pokrenuta na AWS-u, lokalnom Kubernetes klasteru ili Raspberry Pi uređaju, sve će

raditi isto. Tvori ujedinjenu ulaznu platformu kombinirajući sve komponente za isporuku prometa sa željenog servera; skupa sačinjava obrnuti proxy, balanser opterećenja, API pristupnik i DDoS zaštitu [8]. Da bi se pristupilo uslugama ngroka potrebno je obaviti registraciju te odabrati plan. Za potrebe rada se koristio besplatni plan koji pruža sve osnovne zahtjeve za usluge pružanja jednog porxyja u razvojnom okruženju; u jedan takav plan uključeni su automatski SSL/TLS certifikati i ograničenje pristupa putem Oautha s mogućnošću do dvadeset tisuća zahtjeva mjesečno uz ograničenu propusnost. Osim pružanja usluge tuneliranja, ngrok također nudi nadzornu ploču za praćenje potrošnje, statusa i pregleda usluga (Slika 20).



Slika 20. Ngrok nadzorna ploča

Nakon instalacije i integracije na osobno računalo omogućeno je stvaranje HTTP krajnje točke. S obzirom da je nadzorna ploča podignuta na *https://localhost:8443* potrebno je pokrenuti naredbu *ngrok http https://localhost:8443* kojoj rastavljajući je na dijelove dajemo sljedeća značenja:

- *ngrok* – alat za stvaranje sigurnog tunela prema lokalnom serveru
- *http* – protokol koji će ngrok koristiti za tunel; iako lokalni server koristi HTTPS i dalje se koristi naredba http jer ngrok sam upravlja SSL/TLS enkripcijom

- *https://localhost:8443* – adresa lokalnog servera koji se izlaže; označava da server radi na lokalnom računalu i sluša na vratima 8443 koristeći HTTPS.

Nakon pokretanja naredbe kreira se proxy te se lokalnoj poveznici može pristuputi s bilo koje mreže i uređaja pristupajući joj s poveznice navedene u odjeljku *Forwarding*. Izuzev toga, na poveznici navedenoj pod *Web Interface* možemo pristupiti nadzornoj ploči koja sadrži promet i aktivnosti povezane s tunelom kao i mogućnost praćenja zahtjeva, odgovora i pristup statistikama i zapisnicima (Slika 21).

```
Session Status                online
Account                       [redacted] (Plan: Free)
Update                        update available (version 3.16.0, Ctrl-U to update)
Version                       3.5.0
Region                        Europe (eu)
Latency                       43ms
Web Interface                 http://127.0.0.1:4040
Forwarding                    https://5c0c-95-178-162-223.ngrok-free.app -> https://localhost:8443

Connections                   ttl     opn     rt1     rt5     p50     p90
                              19      7       0.24    0.06    0.20    4.22

HTTP Requests
-------------

GET /api/v1/job/default                    200 OK
GET /api/v1/statefulset/default            200 OK
GET /api/v1/pod/default                    200 OK
GET /api/v1/daemonset/default              200 OK
GET /api/v1/cronjob/default                200 OK
GET /api/v1/namespace                      200 OK
GET /api/v1/deployment/default             200 OK
GET /api/v1/replicationcontroller/default 200 OK
GET /api/v1/replicaset/default             200 OK
GET /api/v1/replicationcontroller/default 200 OK
```

Slika 21. Rezultat pokrenute ngrok naredbe

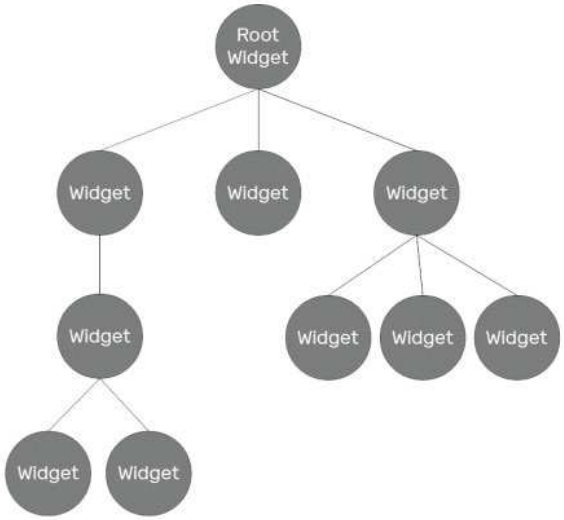
4.4. Widgeti

Flutter widgeti su posvuda u aplikaciji. Možda nije baš sve widget, ali gotovo da je. Čak je i aplikacija widget u Flutteru i zato je ovaj koncept tako važan. Widget predstavlja dio grafičkog sučelja, što ne znači da je isključivo vidljiv. Widget tako može biti:

- Vizualni element koji je osnovni strukturni element, kao što su widgeti za gumb ili tekst
- Element speficičan za raspored koji može definirati poziciju, margine ili ispune
- Element stila koji može pomoći pri bojanju i vizualnoj temi

- Element interakcije koji pomaže odgovoriti na interakcije na različit način

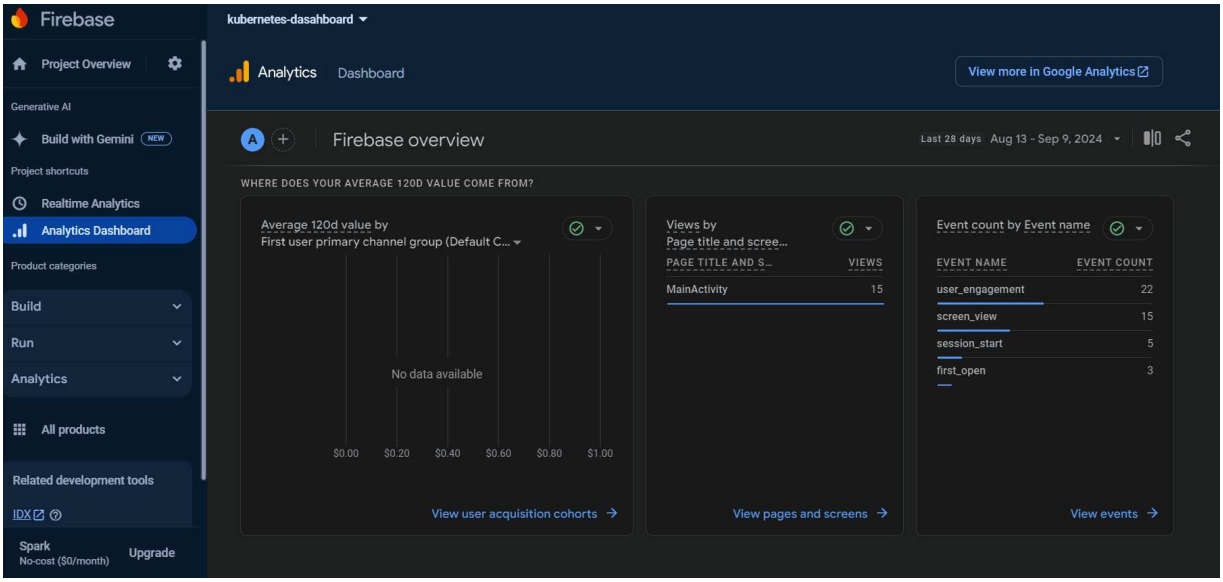
Widgeti oživljavaju u stablima widgeta koji predstavljaju logičku prezentaciju svih widgeta grafičkog sučelja. Izračunavaju se tijekom rasporeda (mjere i strukturne informacije) i koriste se tijekom renderiranja (od okvira do zaslona) i testiranja pogodaka (dodirne interakcije). Koristeći puno optimizacijskih algoritama pokušava manipulirati stablom što je manje moguće, smanjujući pritom ukupnu količnu posla utrošenu na renderiranje s ciljem veće učinkovitosti [1]. Widgeti su u stablu predstavljeni kao čvorovi (Slika 22) gdje svaki widget ima svoje stanje; svaka promjena stanja rezulira i ponovnom izgradnjom widgeta i njegove djece. Dakle, struktura djece stabla nije statična i definirana je opisom widgeta.



Slika 22. Stablo widgeta [1]

4.5. Firebase

Firebase pruža robustan skup Flutter dodataka koji povezuju aplikaciju s Firebase uslugama. Firebase integracija s Flutter aplikacijama omogućuje poboljšanje kvalitete aplikacije s manje uloženog vremena i truda kako bi se optimizirala s posljedicom uvećanog korisničkog zadovoljstva. Omogućava autentifikaciju, analitike i proširenja u oblaku (Slika 23).



Slika 23. Isječak iz nadzorne ploče projekta za analitiku

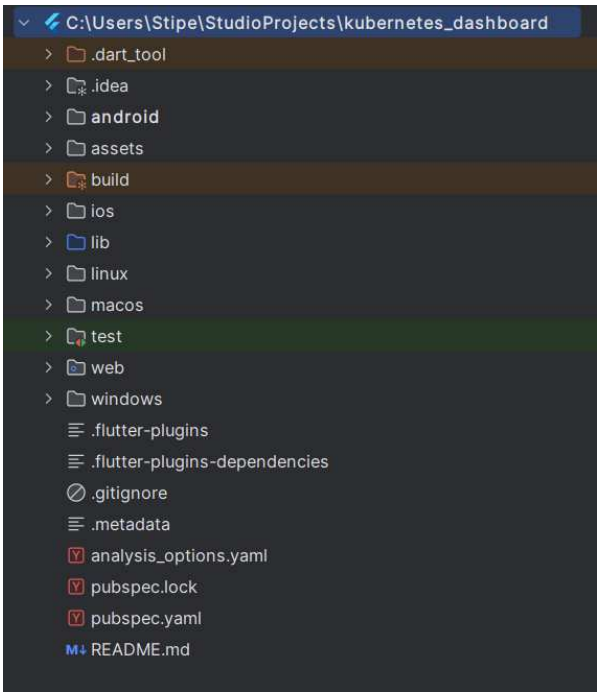
Kako bi se koristila ta proširenja potrebno je registrirati aplikaciju na Firebase konzoli i skinuti konfiguracijsku datoteku za platformu po želji. Za potrebe projekta kreiran je Firebase projekt imena *kubernetes-dashboard* unutar kojeg je registrirana aplikacija *com.example.kubernetes_dashboard* čiji je naziv zapravo *applicationId* koji je vidljiv u *build.gradle* datoteci na razini aplikacije. Potom se na temelju pruženih informacija kreira *google-services.json* datoteka koju je potrebno pospremiti skupa s novim varijablama u *build.gradle* datotekama na razini aplikacije i na razini projekta čime Firebase biva integriran u Flutter projekt.

4.6. Projekt

Za potrebe rada kreiran je projekt naziva *kubernetes-dashboard*. Za upravljanje Flutterom, Dartom i simulaciju aplikacije na uređajima korišten je Android Studio. Po izlistavanju osnovne strukture elemenata, dobiju se sljedeće važnije datoteke i mape (Slika 24):

- *Android* – sadrži kod specifičan za platformu, mjesto gdje se nalazi prevedena aplikacija

- *Lib* – glavna mapa Flutter aplikacija gdje se odvija većina razvojnog vremena; generirani projekt sadrži barem *main.dart* datoteku za početak
- *Pubspec.yaml* – definira Dart pakete. Jedna od važnijih datoteka projekta koja definira broj izgradnje aplikacije. Također navodi ovisnosti o vanjskim dodatcima, slikama, fontovima i slično.
- *Pubspec.lock* – generirana datoteka zaključavanja koja se ne bi smjela uređivati. Ažurira se po promjeni datoteke *pubspec.yaml* i osigurava da se ne uvedu nekompatibilne verzije paketa.
- *Test* – mapa koja može sadržavati sve datoteke povezane s testiranjem projekta kako bi se osiguralno neunošenje grešaka u aplikaciju tijekom razvijanja.



Slika 24. Struktura projekta

4.7. WebView

WebView je važan dio mobilnih aplikacija jer omogujuće pristup web stranici u samoj aplikaciji umjesto odvođenja u preglednik. Koristan je u situacijama kada je potreban eksterni web resurs u aplikaciji, ali ga ne želimo izraditi posebno za mobilnu aplikaciju zbog

cijene i vremenske dugotrajnosti. WebView widget je widget koji prikazuje web stranicu u Flutter aplikaciji kao i kontrolere koji se koriste za upravljanje widgetom. Nakon rješavanja svih ovisnosti u *pubspec.yaml* datoteci potrebno je pokrenuti naredbe za rješavanje ovisnosti.

Ova Flutter aplikacija u srži integrira Firebase za pozadinske usluge i fokusira se na pružanje korisničkog sučelja koje pristupa web sadržaju putem WebViewa. Početni zaslon prikazuje sliku i gumb, koji pri dodiru vode do novog zaslona na kojem se Kubernetes nadzorna ploča prikazuje pomoću WebViewa (Slika 235).

Slika 25. Početni zaslon aplikacije

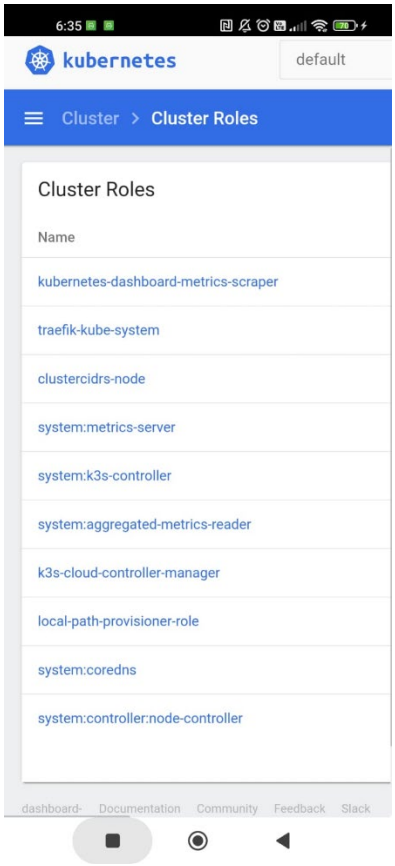
Implementirana u *webView.dart* dadoteci, komponenta WebView koristi WebViewController za učitavanje i prikaz udaljene web stranice, omogućujući korisnicima neometanu interakciju s Kuberentes nadzornom pločom unutar aplikacije. Kako bi se uputila aplikacija na koju web adresu je potrebno skrenuti, potrebno je u *loadRequest* upisati URL

dobiven ngrok tuneliranjem kako bi aplikacija bila dostupna sa svih lokacija, a ne samo na istoj mreži (slika 34).

```
class _WebViewState extends State<WebView> {
  final cotroller = WebViewController()
    ..setJavaScriptMode(JavaScriptMode.unrestricted)
    ..loadRequest(Uri.parse("https://example-505.ngrok-free.app"));
```

Slika 26. Upis URL-a u aplikaciji

Rezultat svih prethodnih operacija i upravljanja je mobilna aplikacija koja sadrži pregled nadzorne ploče istovjetan dostupnome lokalno na URL-u na računalu, samo s prilagođenim izgledom (Slika 27).



Slika 27. Nadzorna ploča u aplikaciji na mobilnom uređaju

5. Zaključak

Rad je kroz praktične i teorijske smjernice prikazao jedan osnovni tijek razvoja sustava praćenja za Kubernetes klastere; od podizanja samog klastera pa do implementacije nadzorne ploče kao web i mobilnog sučelja. Nastavno, ista implementacija je prikladna za mnoštvo klastera i uređaja te predstavlja korak ka zamašnom rješenju sustava kompletnog praćenja; gdje kompletno ukaziva na dodatna moguća poboljšanja koja mogu uključivati primjerice aktivne ažurne sustave za upozoravanje na greške u sklopu slanja obavijesti u stvarnom vremenu, implementaciju aplikacije na daljnje sustave te prilagodbu na značajno veća opterećenja i složenije infrastrukture. Iako rad prikazuje jednostavnu upotrebu i praćenje, takav primjer je usporediv i predstavlja okosnicu za daljnja unaprjeđenja i usklađivanje s nadolazećim budućim zahtjevima.

6. Literatura

[1] Bailey, Thomas, et al. Flutter for Beginners an Introductory Guide to Building Cross-Platform Mobile Applications with Flutter 2. 5 and Dart, 2nd Edition. Birmingham, Packt Publishing, Limited, 2021

[2] Mendez, S. (2022). *Edge Computing Systems with Kubernetes*. Packt Publishing Ltd

[3] docs.k3s.io. (2023). *K3s - Lightweight Kubernetes | K3s*. [na mreži] Dostupno na: https://docs.k3s.io/ [Pristupljeno 1. rujna 2024].

[4] Lukša, M. (2018). *Kubernetes in action*. Shelter Island, Ny: Manning Publications Co

[5] Mocevicius, R. (2020). *Kubectl : Deploy, Manage, and Debug Container Workloads Using the Kubernetes CLI*. Birmingham: Packt Publishing, Limited

[6] Poulton, Nigel (2024). *The Kubernetes Book*. NIGEL POULTON LTD

[7] Robinson, E. (2018). *Kubernetes on AWS*. Packt Publishing Ltd

[8] Ngrok.com. (2024). *What is ngrok? | ngrok documentation*. [na mreži] Dostupno na: https://ngrok.com/docs/what-is-ngrok/ [Pristupljeno 13. srpnja 2024].

[9] Windmill, E. and Rischpater, R. (2020). *Flutter in action*. Shelter Island, Ny Manning Publications Co

[10] Rap Payne (2019). *Beginning app development with Flutter : create cross-platform mobile apps*. New York, Ny: Apress

[11] Kubernetes. (n.d.). *Deploy and Access the Kubernetes Dashboard*. [na mreži] Dostupno na: https://kubernetes.io/docs/tasks/access-application-cluster/web-ui-dashboard/.

[12] Katz, M., Moore, K.D., Ngo, V. and Guzzi, V. (2024). *Flutter Apprentice*. Razeware Llc

[13] Yuen, Billy, et al. *GitOps and Kubernetes : Continuous Deployment with Argo CD, Jenkins X, and Flux*. Shelter Island, Ny, Manning Publications, 2021.

Sažetak

RAZVOJ NADZORNE PLOČE ZA KUBERNETES KLASTERE

U radu je razvijena nadzorna ploča za upravljanje i nadzor Kubernetes klastera prilagođena za korištenje na webu i mobilnim uređajima. Koristi se K3s distribucija Kubernetesa koja je optimizirana za rad u uvjetima s ograničenim resursima i na rubnim lokacijama. Nadzorna je ploča kompatibilna s raznim uređajima unutar klastera poput ARM uređaja te omogućuje korisnicima jednostavnu interakciju s klasterom bez obzira na lokaciju ili uređaj. Detaljno je opisan proces razvoja od postavljanja razvojnog okruženja preko integracije s Kubernetes klasterom. Uz teorijski dio koji pokriva osnove Kubernetesa i ostalih korištenih tehnologija, rad sadrži praktične smjernice i kod potreban za postavljanje i korištenje nadzorne ploče.

Ključne riječi: Kubernetes, nadzorna ploča, k3s, Flutter, računarstvo na rubu, Dart, mikroservis, klaster

Summary

DEVELOPMENT OF KUBERNETES DASHBOARD

In this paper, a dashboard for managing and monitoring Kubernetes clusters, optimized for both web and mobile devices, has been developed. It uses K3s distribution of Kubernetes, which is optimized to work in resource-constrained environments and in edge locations. The dashboard is compatible with various devices within the cluster such as ARM devices and allows users to easily interact with the cluster regardless of location or device. The development process from setting up the development environment to integration with the Kubernetes cluster is described in detail. In addition to the theoretical part that covers the basics of Kubernetes and other used technologies, the paper contains practical guidelines and code needed to set up and use the dashboard.

Keywords: Kubernetes, dashboard, k3s, Flutter, edge computing, Dart, microservice, cluster

Skraćenice

SDK *Software development kit – paket za razvoj programa*

DDoS *Distributed denial-of-service – distribuirano uskraćivanje usluge*

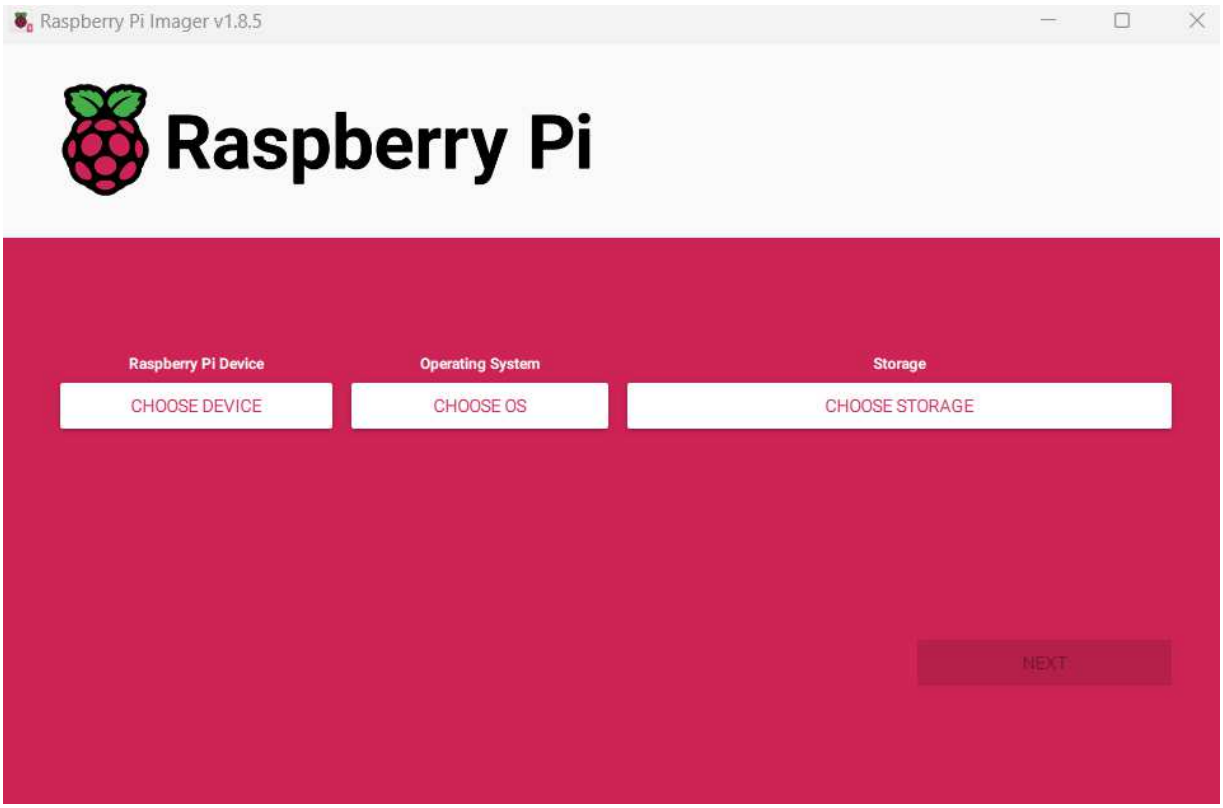
AOT *Ahead-of-time – prije vremena*

JIT *Just-in-time – točno na vrijeme*

Privitak

Instalacija i korištenje programske podrške

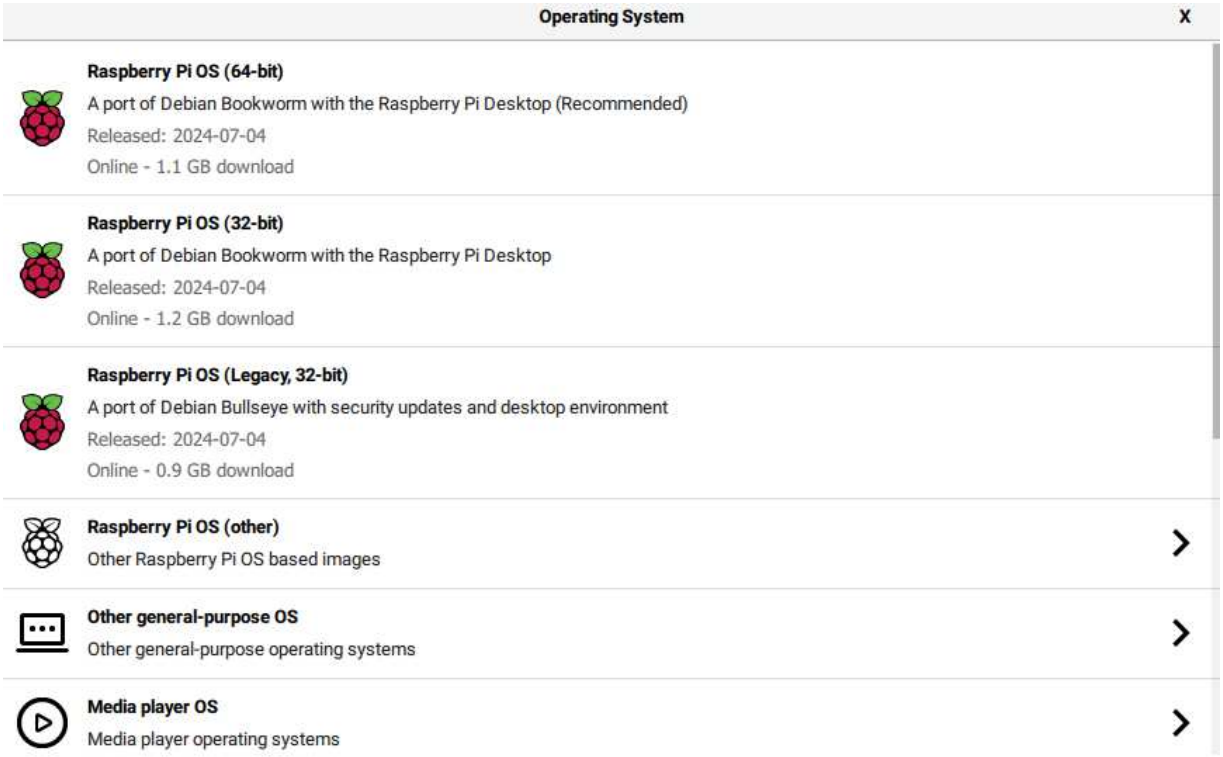
Rad je izveden uz pomoć besplatnog softvera Raspberry Pi OS-a koji je kompatibilan sa svim Raspberry Pi modelima. Spomenuti softver se može instalirati uz pomoć Raspberry Pi Imager aplikacije koju je moguće preuzeti s web stranice *https://www.raspberrypi.org/software/*. Pokretanjem aplikacije dolazi se na početni ekran na kojem su ispisane tri opcije – odabir operacijskog sustava koji će se prvo skinuti pa potom i instalirati, mjesto na koji će se odabrani softver instalirati te izbornik Raspberry Pi uređaja (Slika 28).



Slika 28. Raspberry Pi Imager

Klikom na gumb "CHOOSE OS" iskače novi prozor na kojem se može birati među nekolicine već predviđenih softvera, ali se također može ubaciti i vlastiti u obliku datoteke formata IMG. Za potrebe ovog rada instalirana je potpuna verzija Raspberry Pi OS-a (Slika 29). Instalacija

potpunog softvera je preporučena jer su osim prisustva radne površine već predinstalirane sve preporučene aplikacije.



Slika 29. Dostupni operacijski sustavi