Testiranje strategija uvođenja novih usluga s pomoću dodatka Istio

Pipić, Zvonimir

Undergraduate thesis / Završni rad

2024

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Electrical Engineering and Computing / Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:168:163648

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 1548

TESTIRANJE STRATEGIJA UVOĐENJA NOVIH USLUGA S POMOĆU DODATKA ISTIO

Zvonimir Pipić

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 1548

TESTIRANJE STRATEGIJA UVOĐENJA NOVIH USLUGA S POMOĆU DODATKA ISTIO

Zvonimir Pipić

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 4. ožujka 2024.

ZAVRŠNI ZADATAK br. 1548

Pristupnik:	**Zvonimir Pipić (0036544015)**
Studij:	Elektrotehnika i informacijska tehnologija i Računarstvo
Modul:	Računarstvo
Mentor:	prof. dr. sc. Mario Kušek

Zadatak: **Testiranje strategija uvođenja novih usluga s pomoću dodatka Istio**

Opis zadatka:

U današnjem razvoju usluga na Internetu nije dovoljno uslugu instalirati na jednom računalu već je zbog velikog broja korisnika potrebno koristiti raspodijeljeni sustav usluga. Obično se takvi sustavi sastoje od mikrousluga. Kako je takvih usluga puno onda se koriste alati za orkestraciju kao što su Kubernetes. Vaš zadatak je instalirati i podesiti Kubernetes s dodatkom Istio koji omogućuje upravljanje komunikacijom između usluga tj. omogućuje usmjeravanje podataka koji se razmjenjuju između usluga. Na studijskom primjeru sustava s uslugama potrebno je ispitati načine uvođenja nove verzije usluge bez ispada. Ispitajte različite strategije uvođenja nove usluge kao što su: rekreiranje, rampa, plavo/zelena strategija, kanarinac, A/B testiranje i mračno pokretanje. Svu potrebnu literaturu i uvjete za rad osigurat će Vam Zavod za telekomunikacije.

Rok za predaju rada: 14. lipnja 2024.

Sadržaj

Uvod

U današnjem razvoju usluga na Internetu nije dovoljno uslugu instalirati na jednom računalu već je zbog velikog broja korisnika potrebno koristiti raspodijeljeni sustav usluga. Obično se takvi sustavi sastoje od mikrousluga. Kako je takvih usluga puno onda se koriste alati za orkestraciju kao što su Kubernetes.
Zadatak je instalirati i podesiti Kubernetes s dodatkom Istio te na primjeru sustava koji koristi te usluge ispitati načine uvođenja nove verzije usluga. Strategije koje se ispituju su: plavo/zelena strategija, rekreiranje, kanarinac, kotrljajuća zamjena, A/B testiranje i mračno pokretanje.

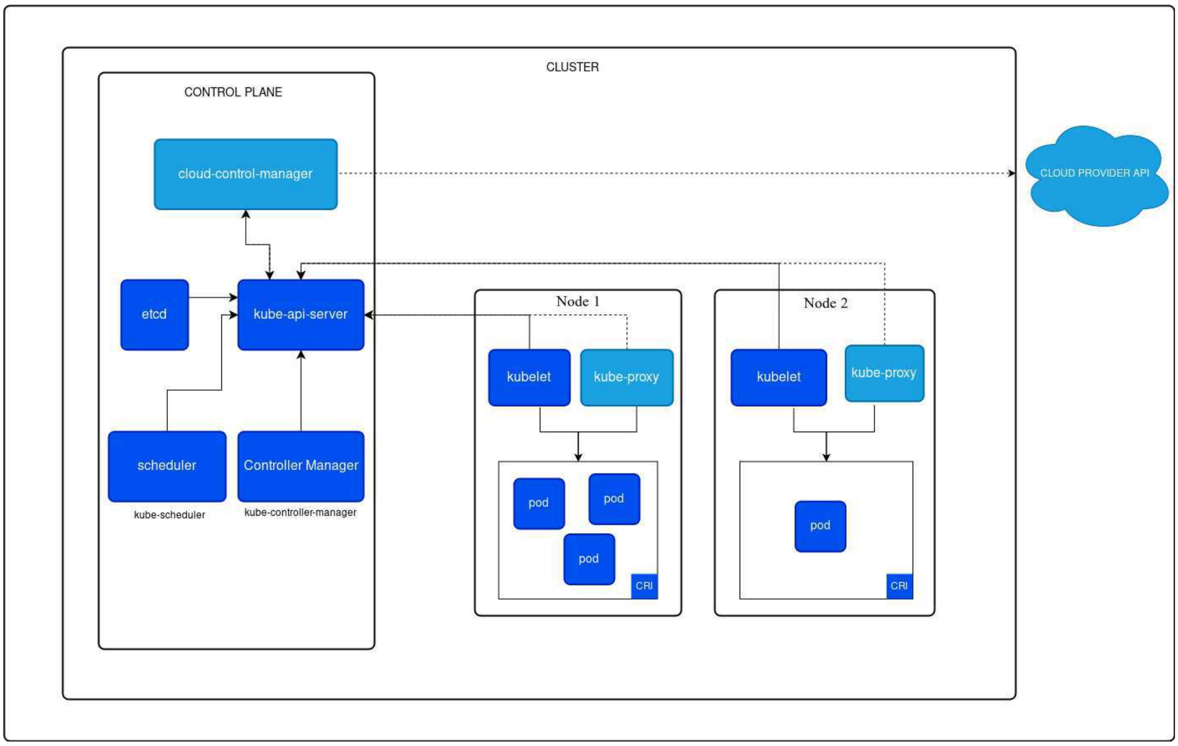
1 Alati za orkestraciju

Alati za orkestraciju ključni su za upravljanje kompleksnijim aplikacijama koje se sastoje od više servisa i virtualnih strojeva ili kontejnera.
Virtualni stroj je softverska emulacija fizičkog reačunala koja se ponaša identično kao fizičko računalo, ali je pokrenuta unutar drugog računala, računala domaćina. Unutar virtualnog stroja su simulirane sve komponente fizičkog računala poput operacijskog sustava, diska i memorije. [1]
Kontejner je softverska cjelina koja sadrži sve potrebne značajke za pokretanje aplikacije poput biblioteka, konfiguracijskih i drugih datoteka. [1]
Za razliku od virtualnih strojeva, kontejneri nemaju svoj operacijski sustav nego koriste operacijski sustav domaćina. Zbog toga zahtijevaju manje mjesta na disku, manje memorije te manje procesorske snage što je dovelo do toga da su u današnje vrijeme korišteniji od virtualnih strojeva. [2]
Prednost virtualnih strojeva u odnosu na kontejnere je što su u potpunosti izolirani jedni od drugih jer svaki ima svoj operacijski sustav te ako se na jednom pronađe sigurnosni propust, taj propust ne mora nužno postojati na drugom, dok kod kontejnera ako se pronađe propust u operacijskom sustavu domaćina, svi kontejneri su podložni iskorištavanju tog propusta. [2]
Zbog toga što su i virtualni strojevi i kontejneri izolirane cjeline, uvodimo alate za orkestraciju. Ti alati automatiziraju uvođenje, skaliranje i operativne aspekte upravljanja aplikacijama unutar grozda domaćina. Unutar ove domene pojavila su se različita rješenja, od kojih svako nudi jedinstvene mogućnosti i fokusira se na različite aspekte procesa orkestracije.
Docker je platforma za upravljanje kontejnerima koja omogućava izradu, prijenos i pokretanje kontejnera. Docker koristi kontejnerizaciju kako bi aplikacije učinio prijenosnima što znači da bi njihovo izvođenje trebalo biti jednako na bilo kojem uređaju na kojem su pokrenute te da se vrlo lako mogu pokrenuti na različitim uređajima. [3] Ako se svi kontejneri nalaze na istom računalu, Docker je dovoljan za njihovo pokretanje. Međutim, za složenije scenarije koji sadrže više računala, sam Docker nije dovoljan pa je potrebno korisititi naprednije alate za orkestraciju poput Kubernetesa, Apache Mesosa, OpenShifta, Docker Swarma i drugih.
U ovom radu usredotočit ćemo se na Kubernetes, specifično K3s i Istio.

1.1 Kubernetes

Kuberntes je prijenosna, proširiva platforma za upravljanje radnim opterećenjem kontejnera i servisima. Pruža mehanizme za implementaciju, raspoređivanje, ažuriranje i održavanje aplikacija, što ga čini okosnicom modernih implementacija aplikacija.

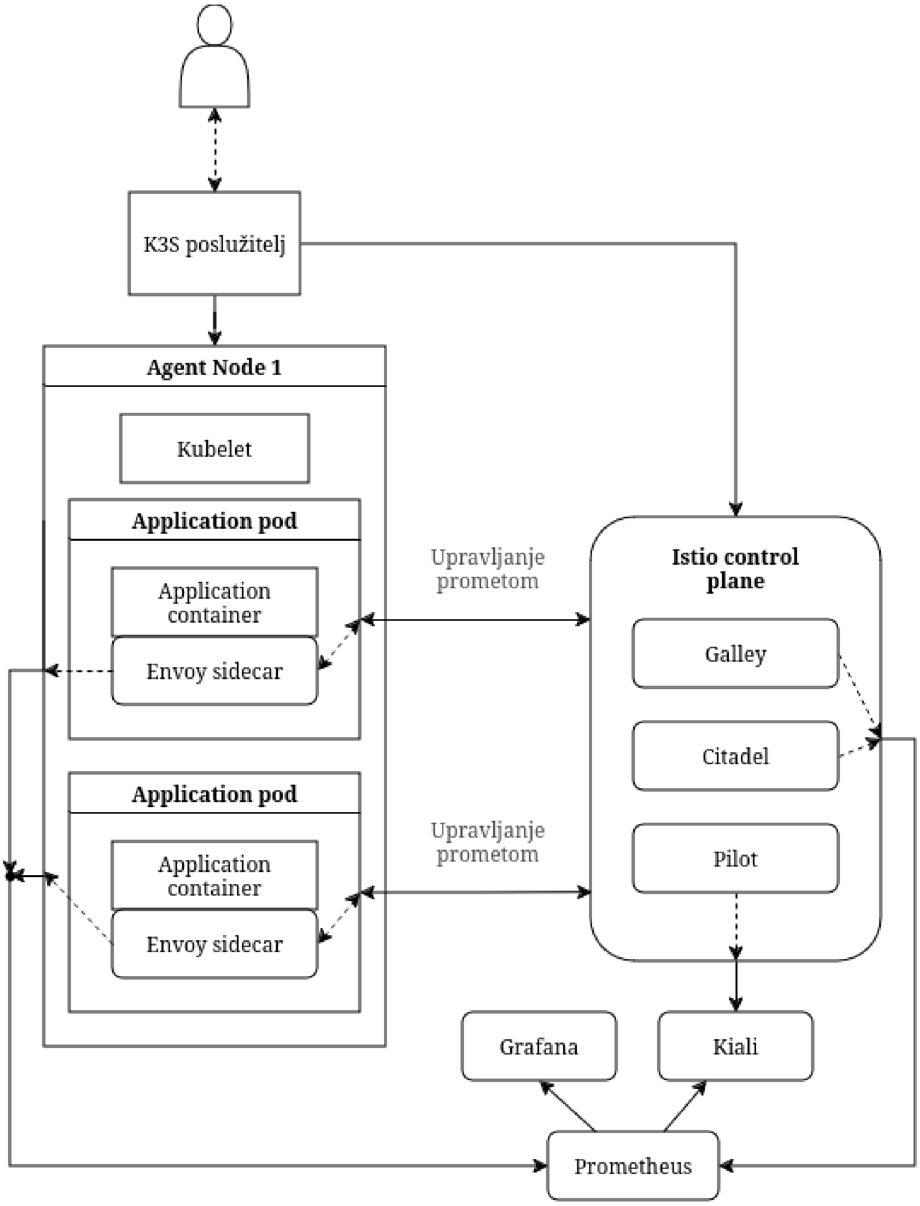
Podržava niz kontejnerskih alata od kojih je najkorišteniji Docker. Kubernetes djeluje u različitim okruženjima, od internih podatkovnih centara do javnih platformi unutar oblaka. [4]



Slika 1.1: Arhitektura Kubernets grozda [5]

Na slici 1.1 vidi se arhitektura Kubernetesovog grozda. Grozd je skup čvorova koji mogu biti ili fizička ili virtualna računala (*Node 1 i Node 2*).
Upravljačka ravnina (*Control plane*) centralan je upravljački sloj Kubernetesa. Služi kao most između vanjskog svijeta (na ovoj slici *Cloud provider API*, u ovom radu to će biti kubectl i ingress upravljači) i čvorova te za upravljanje svim kontejnerima i aplikacijama unutar njih. API poslužitelj (*kube-api-server*) otkriva Kubernetes API kako bi vanjske usluge mogle komunicirati s grozdom, ali također komunicira i s čvorovima. *Etcd* je spremnik za sve konfiguracije i stanja grozda te informacije o svim resursima. Planer (*scheduler*) raspoređuje novo kreirane kapsule u čvorove prema dostupnosti resursa i zadanim pravilima. Upravljač upravljačima (*Controller Manager*) sadrži više upravljača koji upravljaju stanjem grozda kako bi ono odgovaralo željenom stanju. [6] Čvorovi će detaljnije biti objašnjeni u poglavlju 1.2 o K3s-u.

1.2 K3s



Slika 1.2: Arhitektura sustava s K3s i Istio

K3s je pojednostavljena distribucija Kubernetesa namijenjena jednostavnijoj instalaciji i minimalnoj upotrebi resursa. [7] K3s se sastoji od agenta čvora(*Node agent*), aplikacijske kapsule i kontejnera aplikacije (slika 1.2). [8]

Node Agent

Agent node je dio mreže koji sadrži komponente Kubernetesa potrebne za pokretanje i izvršavanje aplikacija. Čvorovi su neovisni jedni o drugima te mogu biti ili virtualne mašine ili zasebna računala. Sadrži kapsule s aplikacijom koje mogu sadržavati različite verzije aplikacije što omogućuje primjenu strategija zamjene. Svaki čvor sadrži Kubelet agent zadužen za komunikaciju s K3s poslužiteljem. Kubelet [9] čvoru agentu pruža informacije o tome koje kapsule treba pokrenuti, a poslužitelju statuse svih kapsula unutar čvora te status samog čvora. [10]

Aplikacijska kapsula (*Application Pod*)

Application pod je najjednostavniji i najmanji dio Kubernetesa koji predstavlja jednu instancu pokrenutog procesa. Sadrži kontejner aplikacije i *Envoy sidecar*.

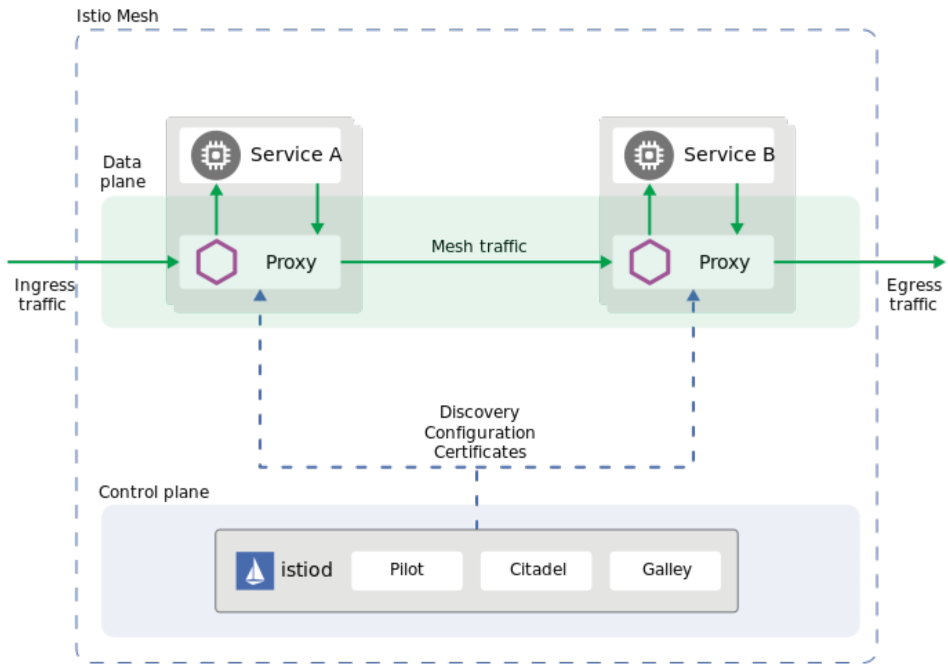
Aplikacijski kontejner *Application Container*

Application container je dio koji pokreće aplikaciju. Svi kontejneri su međusobno neovisni što omogućuje različite verzije aplikacije.

1.3 Istio

Istio je mreža usluga koja omogućuje kontrolu podataka između usluga unutar arhitekture mikrousluga. Osmišljen je za upravljanje otkrivanja usluga, balansiranjem opterećenja i sigurnosnim zahtjevima, te mjerenje i zapis svog prometa unutar grozda. Može se nadodati kao dodatni sloj infrastrukture te ne zahtijeva dodatne promjene koda. [11]

Arhitektura Istia



Slika 1.3: Arhitektura Istia[12]

Istio se sastoji od upravljačke ravnine i ravnine podataka (data plane) [13] [14] kako je prikazano na slikama 1.2 i 1.3.

Upravljačka ravnina

Istio upravljačka ravnina ključna je komponenta upravljanja mrežom servisa. Pruža potrebne konfiguracije i izmjerene podatke o prometu između dijelova sustava te preusmjerava promet na zadane čvorove. Sastoji se od sljedećih komponenti:

Pilot

Pilot je zadužen za upravljanje prometom i njegovo preusmjeravanje te za postavljanje konfiguracije izaslaničkih prikolica (*Envoy sidecars*) kako bi promet bio preusmjeravan na željeni način. Komunicira s Kubernetesom preko *Envoy sidecara* koji se nalaze unutar aplikacijskih kapsula (*pods*).

Galley

Galley služi za upravljanje konfiguracijama mreže servisa i distribucije istih na odgovarajuće komponente te za prikupljanje mjerenja prometa. U novijim verzijama Istia, Galley je uklonjen te su njegove odgovornosti prebačene na Envoy.

Citadel

Citadel je sigurnosna komponenta upravljačke ravnine. Upravlja ključevima i sigurnosnim certifikatima potrebnim za komunikaciju između usluga preko mTLS (mutual Transport Layer Security) protokola. Izdaje i mijenja sigurnosne certifikate kako bi spriječila nedozvoljen ili zlonamjeran pristup uslugama.

Ravnina podataka

Istio ravnina podataka ima ulogu upravljanja prometom između usluga što čini preko *Envoy sidecara*.

Envoy Sidecar

Envoy sidecar dio je ravnine podataka koji je „prikvačen“ na kontejner aplikacije unutar svake kapsule. Uloga mu je presijecanje svog prometa namijenjenog za aplikaciju, njegovo preusmjeravanje na odredišta zadana preko upravljačke ravnine te provođenje sigurnosnih politika poput autentikacije ili ograničenja prometa. U novijim verzijama zadužen je i za telemetrijske podatke.

2 Strategije zamjene

Cilj strategija zamjene je što lakša i "bezbolnija" nadogradnja sustava ili aplikacije sa što manjim utjecajem na korisničko iskustvo. Zamjena se smatra bezbolnom ako je cijelim njenim tokom sustav bio dostupan svim korisnicima i ako je broj grešaka i nepravilnosti nakon što je nova verzija puštena u promet minimalan. [15] [16]

2.1 Plavo/zelena strategija

Sadrži dva identična okruženja, "plavo" (neaktivno) i "zeleno" (aktivno). Samo Jedno okruženje je aktivno u datom trenutku. Zamjena se vrši tako da se promet samo prebaci s jednog okruženja na drugo.

Prednosti

- Eliminacija nedostupnosti sustava jer je u svakom trenutku jedno okruženje aktivno.
- Brzi povratak na prethodnu verziju u slučaju problema s novom verzijom.
- Mogućnost temeljitog testiranja nove verzije u produkcijskom okruženju prije njene aktivacije.

Nedostatci

- Održavanje dva potpuna okruženja istovremeno može biti skupo i memorijski zahtjevno.
- Sinkronizacija konfiguracija i ovisnosti mora se vršiti za oba okruženja prilikom čega može doći do razlika između njih i problema kod prelaska s jednog na drugo.
- Baza podataka mora biti kompatibilna s oba okruženja što može stvarati probleme prilikom dodavanja ili brisanja novih tablica ili atributa.

2.2 Rekreiranje

Jednostavna zamjena 2 verzije; zaustavlja se trenutna, te se postavlja nova verzija aplikacije.

Prednosti

- Nova verzija ne ovisi o prethodnoj, svakom novom ciklus aplikacije počinje ispočetka što može eliminirati prijašnje greške i druge probleme.
- Potrošnja resursa je minimalna pošto se stara verzija skroz zaustavlja prije nego se postavi nova.

Nedostatci

- Prilikom zamjene verzija sustav je nedostupan, a ovisno o veličini pripadajućih datoteka to može potrajati jer je potrebno sve prvo isključiti i pobrisati, zatim sve postaviti na poslužitelja i pokrenuti, a ako je sustav glomazan nije moguće pokrenuti sve iste sekunde kada je prijašnja verzija isključena.
- Postoji rizik da je nova verzija neispravna što može izazvati dugotrajnu nedostupnost sustava dok se greška ne otkloni.
- Ova strategija nije prikladna za skaliranje jer je moguće istovremeno imati pokrenutu samo jednu verziju aplikacije.

2.3 Kanarinac

Mali postotak korisnika preusmjerava se na novu verziju aplikacije.

Prednosti

- Nova verzija je dostupna samo malom broju korisnika što smanjuje utjecaj mogućih problema na korisničko iskustvo većine korisnika.
- Za razliku od plavo/zelene strategije, testiranje se vrši u stvarnom svijetu sa stvarnim korisnicima iz čega proizlazi precizni podaci o performansama sustava.
- Mogućnost prilagodbe brzine izbacivanja nove verzije postavljanjem postotka korisnika kojima je dostupna nova verzija.
- U slučaju povratka na staru verziju nije potrebno mijenjati cijeli sustav nego samo njegov mali dio.

Nedostatci

- Održavanje dvije verzije istovremeno je komplicirano i zahtjeva velike količine resursa.

- Potrebno je više vremena za potpunu zamjenu sustava pošto se nova verzija prikazuje postepeno sve većem dijelu korisnika sve dok nije dostupna svima.
- Korisnici mogu imati različito korisničko iskustvo što može dovesti do zabuna ako su međusobno u kontaktu.

2.4 Kotrljajuća zamjena

Dio po dio sustava se mijenja, na primjer jedan po jedan *pod* u Kuebrentesu.

Prednosti

- Sustav je dostupan prilikom cijelog procesa zamjene.
- Ako dođe do nekog problema taj problem će utjecati samo na manji dio korisnika te se taj problem može lakše otkloniti, ili ako je potrebno može se vratiti na staru verziju.
- Testiranje se može vršiti u proizvodnji što može pružiti precizne podatke o ponašanju sustava sa stvarnim prometom.

Nedostatci

- Potrebni su napredniji alati jer je kompliciranije mijenjati dio po dio sustava.
- Detekcija problema može potrajati jer je nova verzija u početku dostupna manjem broju korisnika pa je statistički manja vjerojatnost da će se otkriti pogreške.
- Nova verzija aplikacije može zahtijevati drugačiju bazu podataka što dovodi do problema prilikom sinkronizacije jer su istovremeno aktivne obje verzije.

2.5 A/B testiranje

Podjela aplikacije na dvije verzije, od kojih su obje istovremeno dostupne različitim korisnicima. Prva verzija (A) je tenutna vezija, a verzija B je nova verzija aplikacije koja je vidljiva određenoj grupi korisnika koja može biti odabrana nasumično ili ciljano, na primjer to mogu biti beta testeri.

Prednosti

- Pruža realne podatke o performansama različitih verzija aplikacije što omogućava pružanje što boljeg korisničkog iskustva.

Nedostatci

- Testiranje na ovakav način može potrajati jer je potreban veći broj korisnika kako bi podaci imali statističko uporište.

2.6 Mračno pokretanje

Mračno pokretanje (*dark lanuch*) strategija je u kojoj su na poslužitelju aktivne i originalna i nova verzija te se zahtjevi šalju na obje verzije, ali korisnik prima odgovore samo s originalne verzije te nema dojam da je u sustavu nešto drugačije nego inače.

Prednosti

- Testiranje svih elementa sustava se vrši u pozadini te nema utjecaja na korisničko iskustvo.
- Problemi i greške se mogu otkriti prije nego što nova verzija postane javna.

Nedostatci

- Ovakav pristup zahtjeva dvostruko više resursa jer su istovremeno aktivne dvije verzije.
- Nema povratnih informacija od strane korisnika jer oni ne vide kako se nova verzija ponaša.

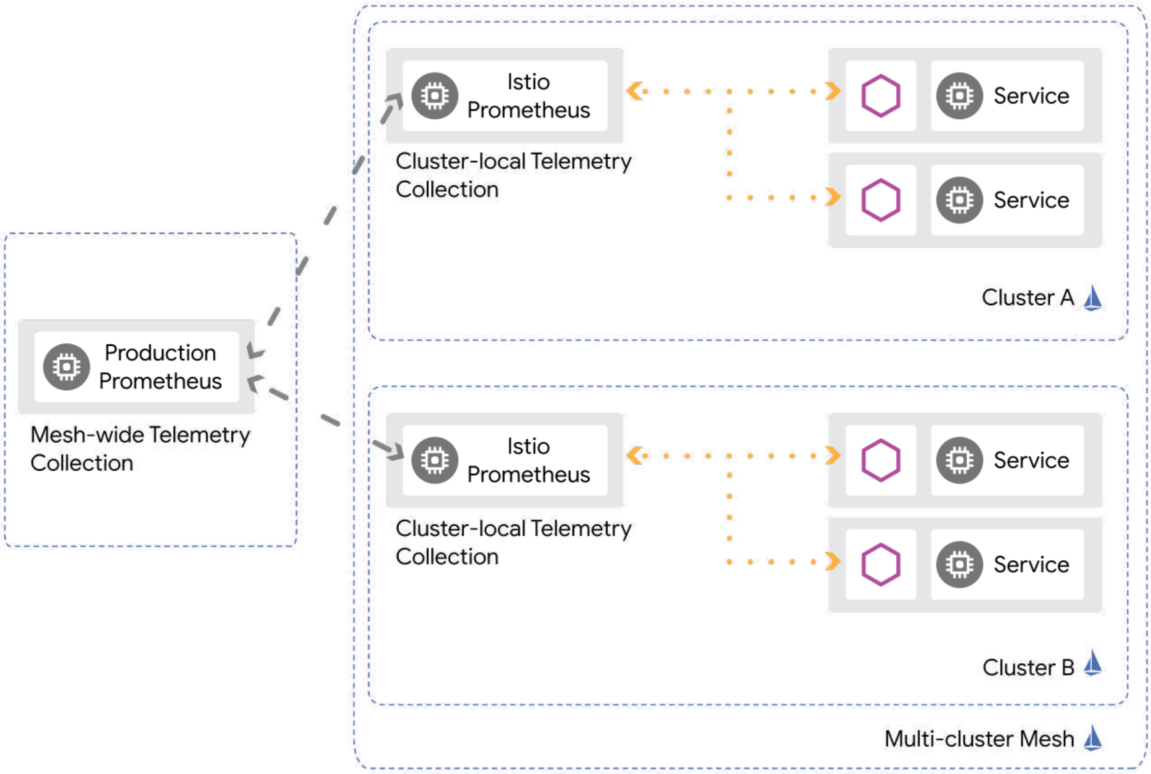
3 Opis rješenja

Za testiranje strategija zamjena korišten je K3s uz pomoć dodatka Istio koji služi za mjerenje i upravljanje prometom prema i od K3s poslužitelja kao što je prikazano na slikama 1.2 i 1.3. K3s poslužitelj središnja je upravljačka točka sustava. Zadužen je za spremanje podataka konfiguracija i stanja sustava, komunikaciju s korisnicima te za upravljanje čvorovima agenata (node agent). To uključuje i postavljanje te upravljanje Istiom i Istio upravljačkom ravninom (plane) i ravninom podataka.

3.1 Mjerenje i nadzor prometa

Mjerenje i nadzor prometa vrši se pomoću dodataka Prometheus, Kiali i Grafana. Ovi alati zajedno omogućuju prikuplanje, vizualizaciju i analizu podataka o prometu.

Prometheus

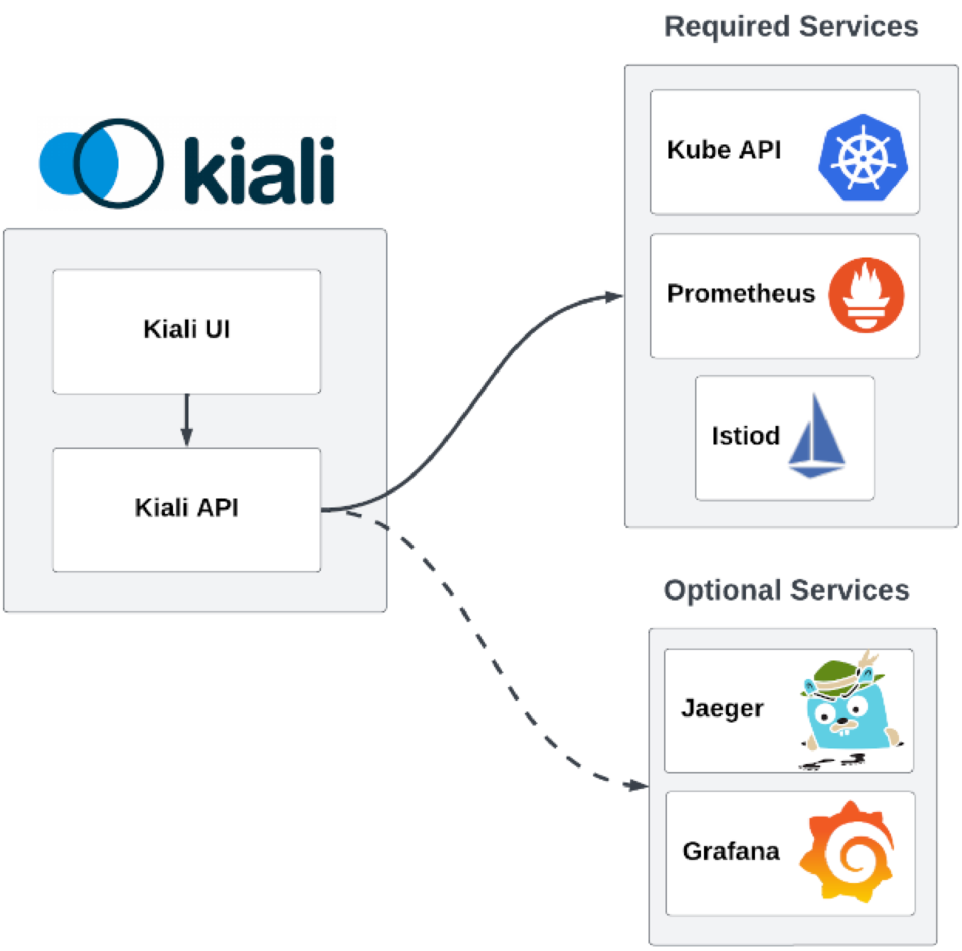


Slika 3.1: Prikaz Prometheusa povezanog na Istio s više grozdova [17]

Prometheus je dodatak za Istio koji periodički dobiva podatke mjerenja od Istio kontrolne ravnine i *Envoy sidecara* sa zadana putanje (u ovom sustavu /stats/prometheus:15090) te ih obrađuje. Mjerenja uključuju broj zahtjeva, postotak uspješnosti i vrijeme odgovora

na zahtjeve. Sve dobivene podatke sprema lokalno te pruža mogućnost pretraživanje tih podataka pomoću jezika PromQL. PromQL moćan je upitni jezik koji služi za složenije filtriranje podataka. Prometheus se može postaviti tako da administratorima šalje upozorenja elektroničkom poštom ako neka mjerenja ispadnu iz zadanih granica što bi sugeriralo na probleme u sustavu.
Dijagram 3.1 prikazuje kako je Prometheus povezan s Istio ili Kubernets grozdovima i kako spaja mjerenja u jednu cjelinu. Svaki grozd ima svoj primjerak Prometheusa koji komunicira s glavnim, zajedničkim Prometheusom zaduženog za prikaz mjerenja korisnicima. Svaki lokalni primjerak Prometheusa komunicira i prikuplja mjerenja s pripadajućih usluga unutar tog grozda.

Kiali



Slika 3.2: Prikaz arhitekture Kiali [18]

Kiali je Istio proširenje zaduženo za vizualizaciju mreže usluga. Pruža uvid u međusobne odnose između usluga unutar mreže. Od kontrolne ravnine dobiva podatke o konfiguraciji, a od Prometheusa dobiva mjerenja te ih kombinira u dijagrame toka i druge korisne grafove. Ti grafovi prikazuju put prometa između usluga, podatke o brzini i uspješnosti zahtjeva te trenutno stanje usluga (dostupnost). Kiali je ključan u detektiranju točne pozicije problema unutar sustava kako bi se problem mogao otkloniti. Za funkcioniranje Kiali nužno je imati instaliran Prometheus, a moguće ga je povezati i s Grafana i Jaeger dodacima. Na slici 3.2 vidljivo je da Kiali API komunicira sa svim povezanim uslugama, a korisničko sučelje prikazuje dobivene podatke preko API-ja.

Grafana

Grafana je alat koji služi za crtanje dijagrama, karata vrućine (*hetmap*), histograma, kružnih grafova i drugih vizualizacijskih oblika. Kao i Prometheus, pruža mogućnost slanja upozorenja u slučaju nepravilnosti mjerenja. Prikuplja podatke s pomoću Promethesua tako što se šalje zahtjev za željena mjerenja u jeziku PromQL te ih obrađuje i pretvara u grafove. Dobivene grafove moguće je organizirati u nadzorne ploče kako bi pregled podatka bio lakši i čitljiviji. Nadzorne ploče omogućuju personalizirani prikaz ključnih metrika prema potrebama korisnika. Postoji i mogućnost izvoza podataka u csv format.
I za Grafanu je Prometheus nužan za funkcioniranje, poput Kiali dodatka (slika 3.2) tako da i komunikacija funkcionira kao što je prikazano na toj slici.

Primjer zahjteva u jeziku PromQL za podatke s Prometheusa:

```
sum by(destination_version) (rate(istio_requests_total
{destination_service="app-service.istio-test.svc.cluster.local",
  destination_version=~"v1|v2"}[1m]))
```

Zahtjev dohvaća podatke o broju zahtjeva prema određenoj verziji po sekundi.

- **istio_request_total** predstavlja ukupan broj zahtjeva obrađenih Istio dodatkom
- **destination_service="app-service.istio-test.svc.cluster.local** odabire samo zahtjeve koji su obrađeni s *app-service* servisom unutar *istio-test* namespace-a.
- **destination_version= "v1|v2"** fokusira se samo na zahtjeve prema verzijama 1 i 2 te eliminira šum ostalih zahtjeva
- **rate()** funkcija računa broj zahtjeva po sekundi za zadane podatke
- **sum by(destination_version) ()** funkcija grupira podatke po verziji aplikacije

3.2 Primjena

Kako bi se strategije zamjene uspješno primijenile potrebno je primijeniti konfiguracije za pokretanje (*Deployment*), usluge (*Service*), virtualne usluge (*VirtualService*), poveznike

(*Gateway*) i odredišna pravila (*DestinationRule*). Sve datoteke sadrže *kind* atribut koji određuje vrstu kreiranog Kubernetes resursa te *metadata* dio koji specificira osnovne podatke o resursu poput imena, verzije i prostora (*namespace*) u kojem će se taj resurs nalaziti.

Deployment

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: app-v1
  namespace: istio-test
  labels:
    app: app
    version: v1
spec:
  replicas: 1
  selector:
    matchLabels:
      app: app
      version: v1
  template:
    metadata:
      labels:
        app: app
        version: v1
      annotations:
        prometheus.io/scrape: 'true'
        prometheus.io/port: '15090'
        prometheus.io/path: '/stats/prometheus'
    spec:
      containers:
      - name: app
        image: zpipic/app:v1
        ports:
        - containerPort: 5000
```

Slika 3.3: Primjer deployment.yaml datoteke

Deployment konfiguracija definira pokretanje aplikacije (slika 3.3). Metadata dio specificira podatke o pokretanju te u kojem prostoru (*namespace*) će se nalaziti. *Replicas* određuje broj kapsula koje će sadržavati aplikaciju, *selector* definira kako identificirati aplikaciju (u ovom slučaju po verziji). *Template* opisuje kapsule kreirane ovim pokretanjem. Najvažniji dijelovi *templatea* u ovom primjeru su *annotations* koji Pormetheusu omogućava prikupljanje podataka definiranjem putanje i vrata (*port*) i *containers* koji definira kontejner s Dockera korišten za aplikaciju. [19]

Service

```yaml
apiVersion: v1
kind: Service
metadata:
  name: app-service
  namespace: istio-test
spec:
  ports:
  - name: http-web
    port: 80
    targetPort: 5000
  selector:
    app: app
```

Slika 3.4: Primjer service.yaml datoteke

Glavna zadaća servisa (slika 3.4) je otkrivanje pokrenute aplikacije unutar Kubernetesovog grozda kako bi joj i druge komponente mogle pristupiti. Definira otkrivena vrata (80 u ovom primjeru) kojem pristupaju druge komponente te ciljani port (5000 u primjeru) na kojem se zapravo nalazi aplikacija. *Slelector* određuje na koje kapsule će promet biti preusmjeren, u našem slučaju na sve koji sadrže oznaku ”app”. [19]

Destination rule

```yaml
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: app-destinationrule
  namespace: istio-test
spec:
  host: app-service.istio-test.svc.cluster.local
  subsets:
  - name: v1
    labels:
      version: v1
  - name: v2
    labels:
      version: v2
  trafficPolicy:
    connectionPool:
      http:
        http1MaxPendingRequests: 5000
        maxRequestsPerConnection: 5000
    outlierDetection:
      consecutiveGatewayErrors: 5
      interval: 5s
      baseEjectionTime: 15s
      maxEjectionPercent: 50
```

Slika 3.5: Primjer datoteke destinationrule.yaml

Konfiguracija odredišnih pravila (slika 3.5) upravlja pravilima koje se izvršavaju za određeni servis (*host*). Definira podskupove aplikacije (*subsets*) koji u ovom primjeru predstavljaju verzije i sigurnosne zaštite u vidu ograničavanja maksimalnog broja zahtjeva s iste veze (*trafficPolicy*). [20]

Mrežni prilaz (*Gateway*)

```yaml
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: app-gateway
  namespace: istio-test
spec:
  selector:
    istio: ingressgateway
  servers:
  - port:
      number: 80
      name: http
      protocol: HTTP
    hosts:
    - "*"
```

Slika 3.6: Primjer gateway.yaml datoteke

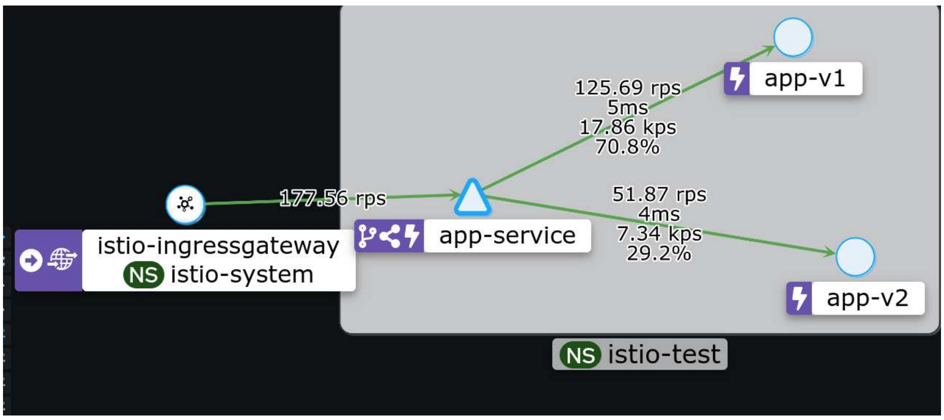
Mrežni prilaz (slika 3.6) upravlja i preusmjerava vanjski promet u Istio mrežu servisa. *Selector* definira koji Istio poveznik se koristi za upravljanje prometom. *s* dio određuje koji port treba biti otkriven i s kojim protokolom, te za koje domaćine (*hosts*) treba obrađivati zahtjeve. [21]

Virtual service

```yaml
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: app-virtualservice
  namespace: istio-test
spec:
  hosts:
    - "*"
  gateways:
    - app-gateway
  http:
  - route:
    - destination:
        host: "app-service.istio-test.svc.cluster.local"
        subset: v1
      weight: 70
    - destination:
        host: "app-service.istio-test.svc.cluster.local"
        subset: v2
      weight: 30
```

Slika 3.7: Primjer virtualservice.yaml datoteke

Virtualni servis (slika 3.7) definira pravila usmjeravanja koja upravljaju raspodjelom prometa. *Hosts* određuje na koje domaćine se primjenjuju ta pravila, a *gateways* koji poveznik se koristi za otkrivanje servisa. *Http* sekcija specificira destinacije preusmjerenog prometa. *Subset* je podskup definiran unutar odredišnih pravila i u ovom primjeru predstavlja verziju aplikacije. *Weight* određuje postotak prometa preusmjeren na pripadajuću verziju. [22]



Slika 3.8: Povezanost poveznika i virtualnog servisa

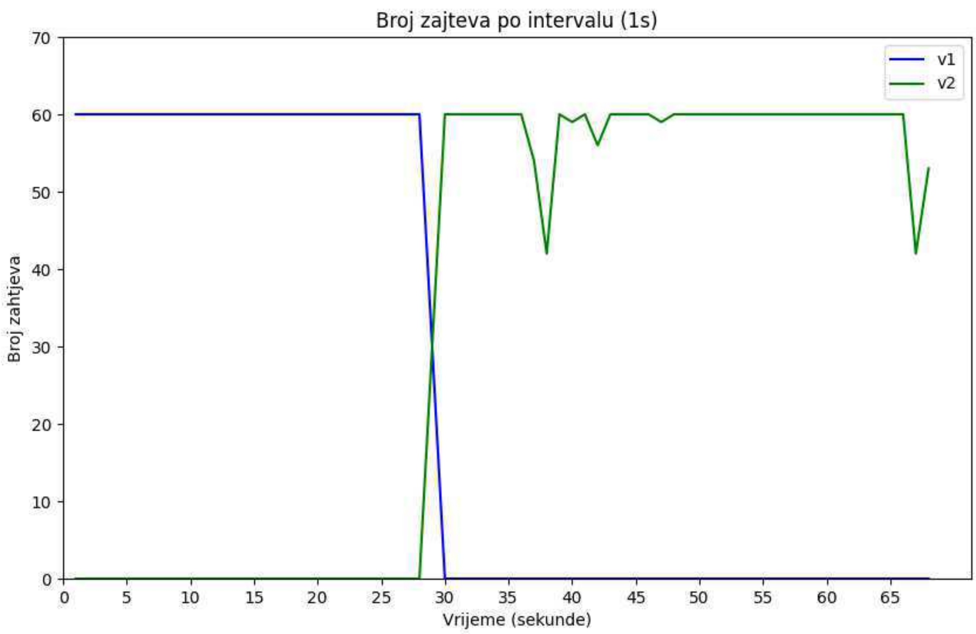
Na slici 3.8 dobivenoj pomoću dodatka Kiali može se vidjeti kako promet putuje od poveznika prema servisu koji zatim raspodjeljuje promet na verzije aplikacije kako je zadano u konfiguraciji.

4 Testiranje

Za potrebe testiranja napravljene su dvije verzije aplikacije za koje se vrši zamjena. Aplikacije su napravljene u jeziku Python te obje rade istim principom: na GET zahtjev odgovoraju teksom ”This is Version 1/2” kako bi mogli razlikovati verzije. Te aplikacije su kontejnerizirane i postavljne na Docker.

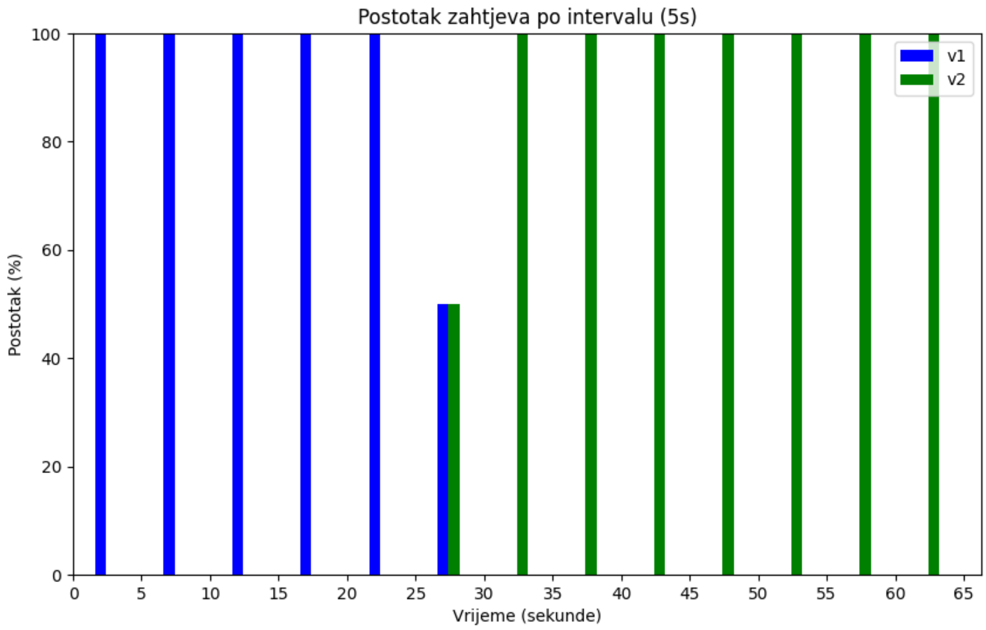
Testiranje je izvedeno tako da sam pokrenuo skriptu test.py (nalazi se u dodatku) koja konstantno šalje zahtjeve na poslužitelj te bilježi koja verzija aplikacije je odgovorila. U određenom trenutku bih promijenio parametar *weight* unutar konfiguracije virtualnog servisa kako bi se promet preusmjeravao na željene verzije i time sam simulirao strategiju zamijene. Rezultati su zapisani u csv datoteku koja se šalje kao parametar u skriptu graph.py (nalazi se u dodatku). Skripta graph.py obrađuje te podatke i crta dva grafa, jedan koji prikazuje broj odgovora u sekundi za svaku verziju i drugi koji prikazuje postotak odgovora obje verzije u zadanom vremenskom intervalu.

4.1 Plavo/zelena strategija



Slika 4.1: Prikaz zahtjeva po sekundi plavo/zelene strategije

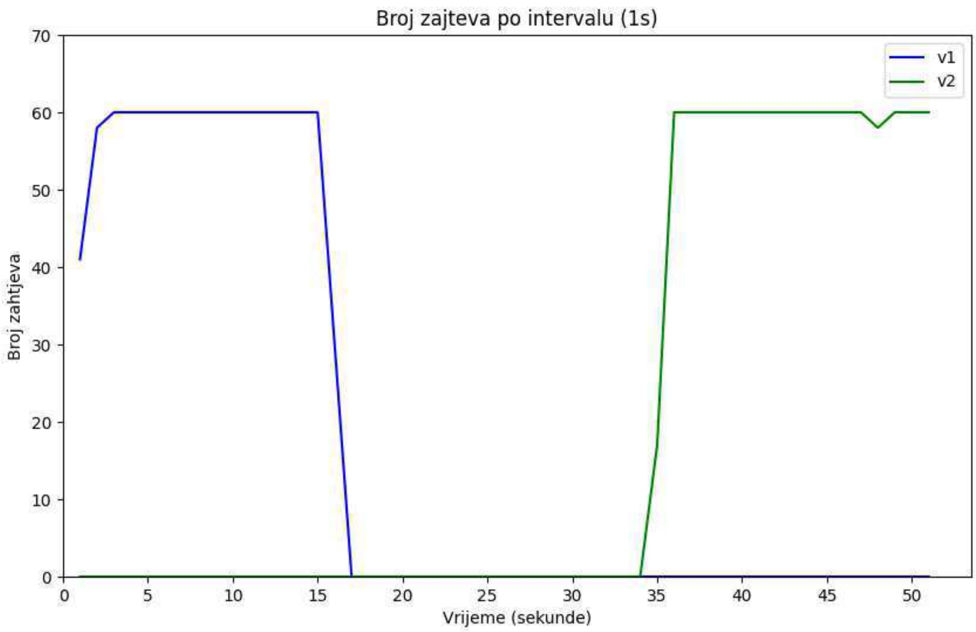
Na slici 4.1 može se vidjeti kako je prvih 28 sekundi aktivna verzija 1 (”plava verzija”). U tom trenutku izvršena je zamjena na verziju 2 (”zelenu verziju”) te se može primijetiti nagli pad zahtjeva na prvu verziju koja je u potpunosti prestala s radom u tridesetoj sekundi. Za to vrijeme je druga verzija u potpunosti preuzela sav promet s prijašnje verzije što je dovelo do za krajnje korisnike neprimjetnog prelaska na noviju verziju, i to bez i jednog trenutka nedostupnosti sustava.



Slika 4.2: Postotak zahtjeva plavo/zelene strategije

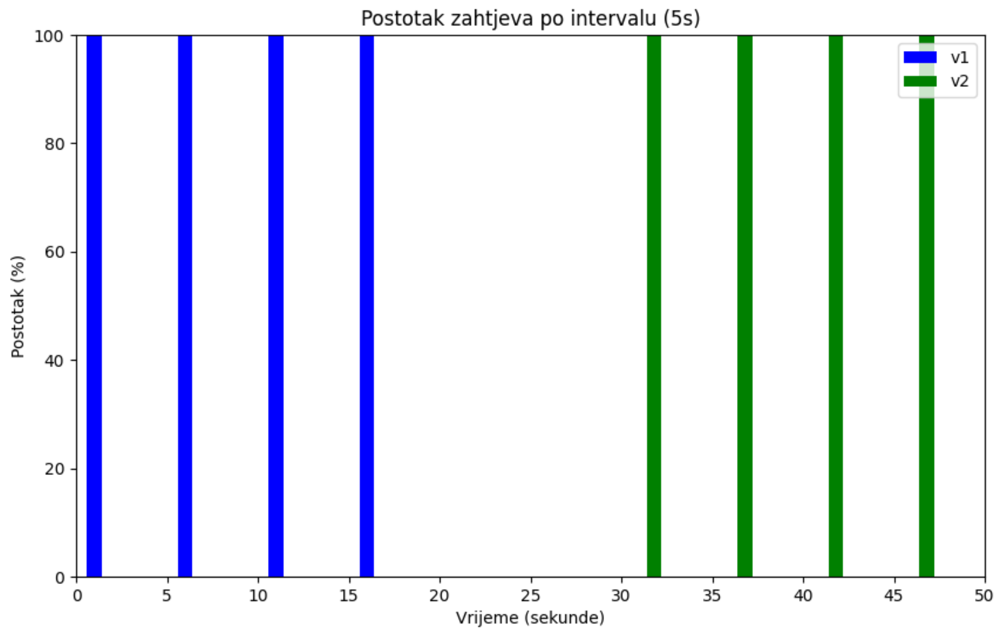
Graf na slici 4.2 prikazuje postotak prometa na svakoj verziji u vremenskom intervalu od 5 sekundi. Na početku je sav promet bio usmjeren na prvu verziju sve do intervala između 25. i 30. sekunde gdje je raspodjela prometa pola-pola između dvije verzije, iz čega proizlazi da se tada dogodila zamjena. Nakon toga sav promet je išao prema novoj, "zelenoj", verziji.

4.2 Rekreiranje



Slika 4.3: Prikaz zahtjeva po sekundi strategije rekreiranja

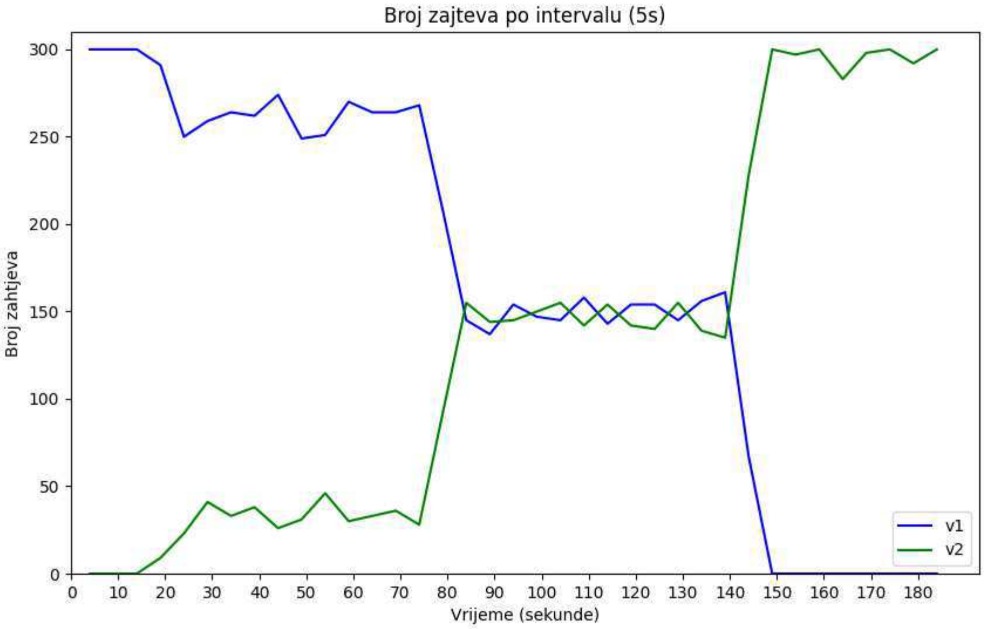
Iz grafa na slici 4.3 vidi se kako je prva verzija bila aktivna do 17. sekunde, nakon čega je isključena s poslužitelja. U tom trenutku aplikacija je postala nedostupna korisnicima, sve do 34. sekunde kada je postavljena nova verzija. Princip ove strategije sličan je plavo/zelenoj strategiji, ali glavni nedostatak joj je vrijeme nedostupnosti poslužitelja korisnicima za vrijeme zamjene.



Slika 4.4: Postotak zahtjeva strategije rekreiranja

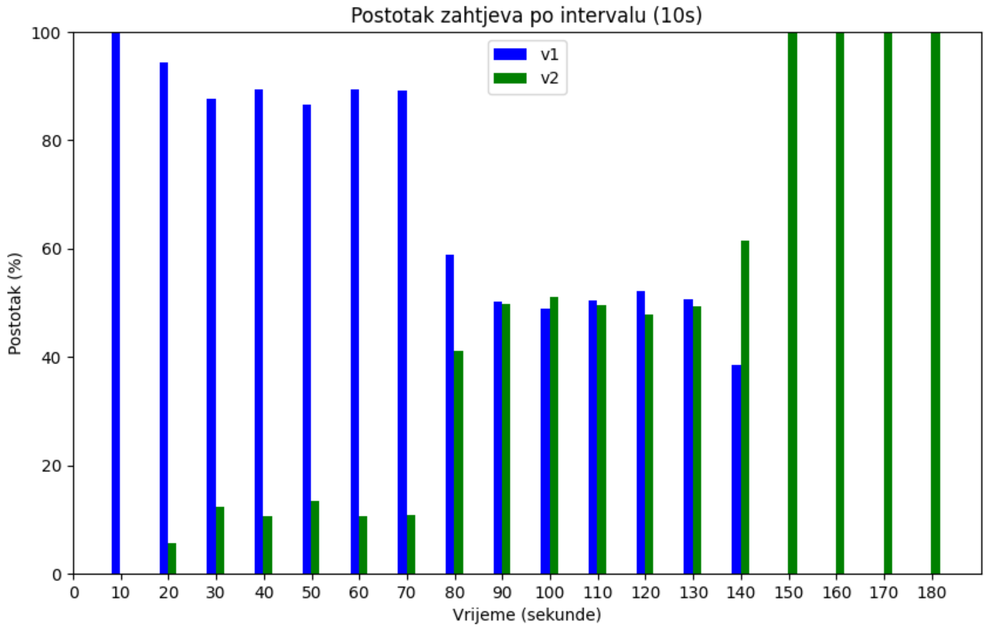
Graf sa slike 4.4 koji prikazuje postotak zahtjeva jasno prikazuje da ni u jednom trenutku obje verzije nisu bile aktivne istovremeno, te da postoji vremenski interval u kojem je poslužitelj nedostupan.

4.3 Kanarinac



Slika 4.5: Prikaz zahtjeva po sekundi strategije kanarinac

Dijagram na slici 4.5 prikazuje da je do 15. sekunde bila aktivna samo verzija 1, od 15. do 85. u malom postotku bila je aktivna i verzija 2, od 85. do 145. obje verzije bile su podjednako aktivne, a nakon toga sav je promet preusmjeren na drugu verziju aplikacije. Poslužitelj je bio dostupan u svim trenutcima zamjene.



Slika 4.6: Postotak zahtjeva strategije kanarinac

Iz dijagrama 4.6 moguće je iščitati kako je u početku sav promet išao prema verziji 1, od 15. do 85. sekunde je oko 10% prometa bilo preusmjereno na novu verziju, od 85. do 135. sekunde promet je bio raspodijeljen podjednako, a nakon toga je sav promet preusmjeren na novu verziju.

4.4 Kotrljajuća zamjena

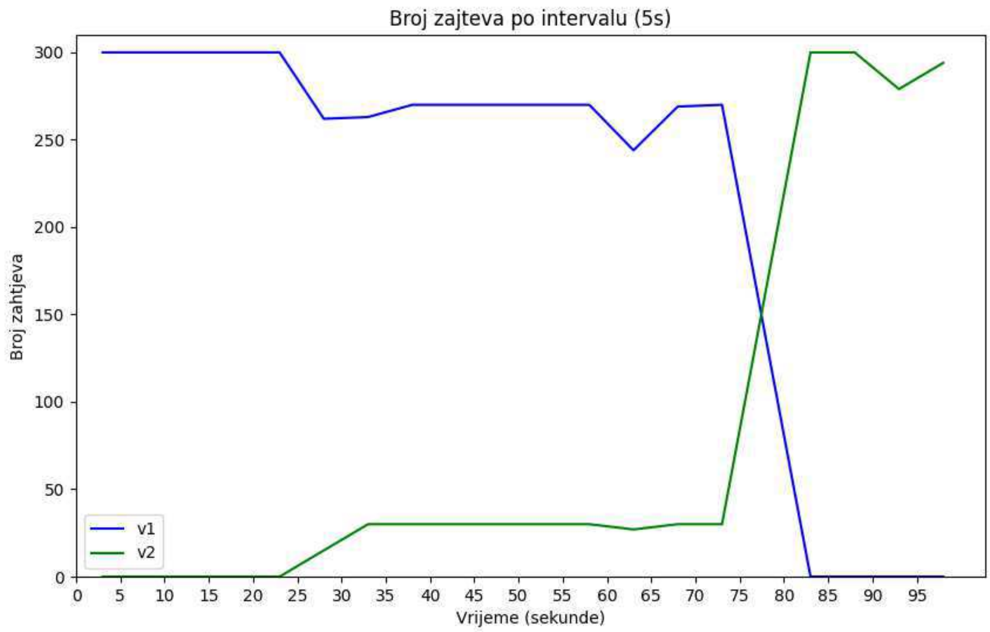
Kotrljajuća zamjena implementira se poput zamjene rekreiranjem, uz razliku da se mijenja kapsula po kapsula umjesto cijelog sustava odjednom. Prednost u odnosu na zamjenu rekreiranjem je u tome što je sustav dostupan cijelo vrijeme. Grafovi izgledaju poput kanarinac zamjene (slike 4.5 i 4.6) tako da ova strategija nije posebno implementirana.

4.5 A/B testiranje

```yaml
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: app-virtualservice
  namespace: istio-test
spec:
  hosts:
    - "*"
  gateways:
    - app-gateway
  http:
  - match:
    - headers:
        X-User-Type:
          exact: beta
    route:
    - destination:
        host: "app-service.istio-test.svc.cluster.local"
        subset: v2
      weight: 100
  - route:
    - destination:
        host: "app-service.istio-test.svc.cluster.local"
        subset: v1
      weight: 100
```

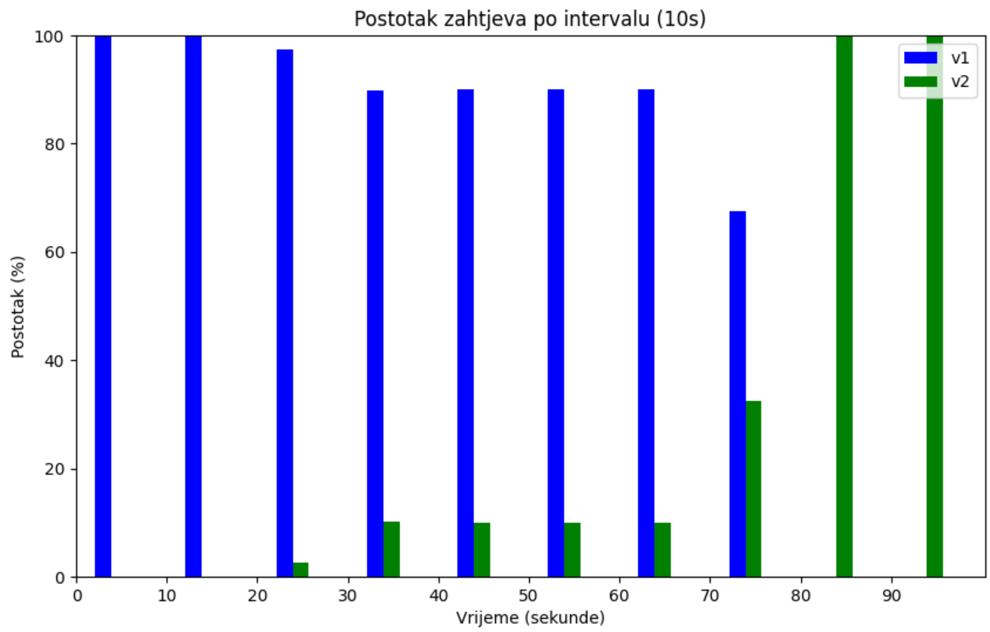
Slika 4.7: Prikaz konfiguracije virtualnog servisa za A/B testiranje

Za ovo testiranje potrebno je koristiti drugačiju konfiguraciju virtualnog servisa (slika 4.7) koja prvo filtrira korisnike po zaglavlju beta te njih preusmjerava na verziju 2 , a sve ostale na verziju 1



Slika 4.8: Prikaz zahtjeva po sekundi A/B testiranja

Iz grafa na slici 4.8 vidljivo je da je nakon 24. sekunde u promet puštena beta verzija (v2) aplikacije za samo dio korisnika. U ovom testu korišteno je 30 dretvi od kojih 3 imaju zaglavlje beta što znači da bi njima trebala biti vidljiva verzija 2. Nakon 75. sekunde u promet je puštena verzija 2 svim korisnicima..



Slika 4.9: Postotak zahtjeva A/B testiranja

Iz grafa postotaka sa slike 4.9 jasno je da je promet tijekom testiranja raspoređen sukladno broju beta korinsika (dretvi) kojih je bilo 10%.

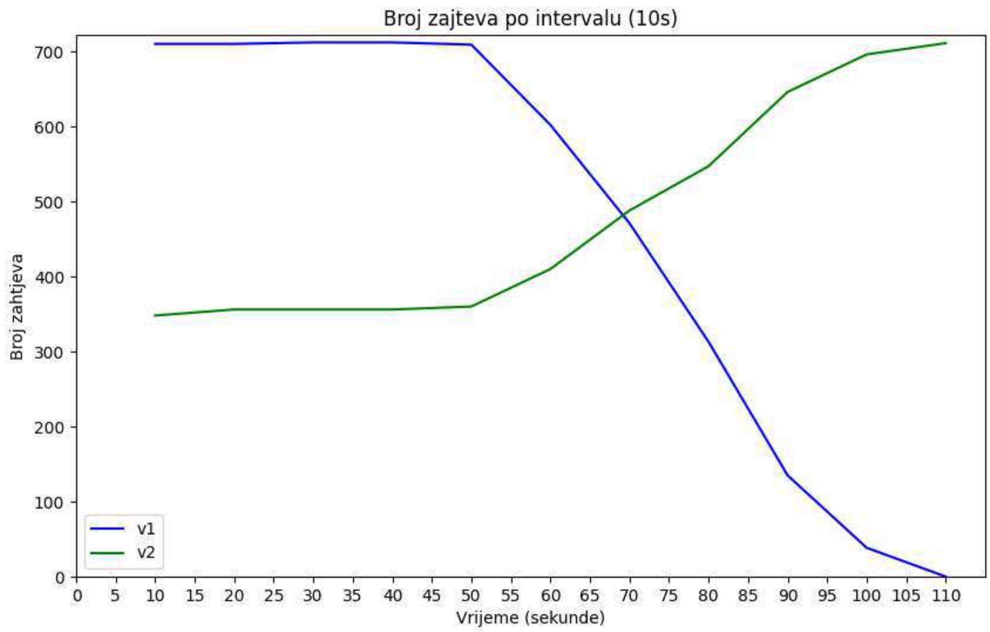
4.6 Mračno pokretanje

Za testiranje ove verzije korištena je i Grafana kako bih dobio mjerenja za usmjeravanje odgovora na verzije.

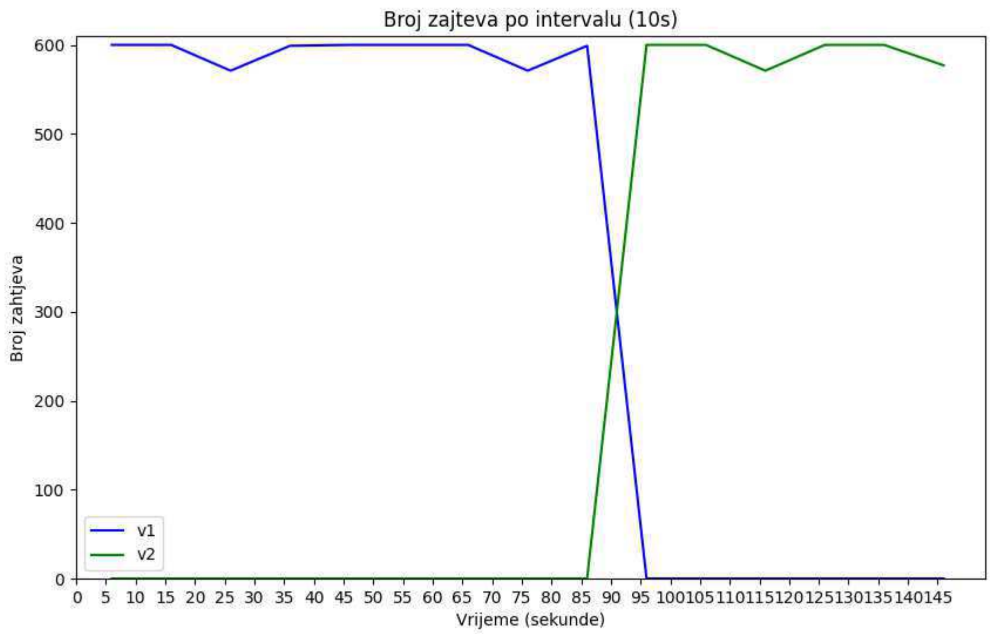
```yaml
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: app-virtualservice
  namespace: istio-test
spec:
  hosts:
    - "*"
  gateways:
    - app-gateway
  http:
  - route:
    - destination:
        host: "app-service.istio-test.svc.cluster.local"
        subset: v1
      weight: 100
    mirror:
      host: "app-service.istio-test.svc.cluster.local"
      subset: v2
    mirrorPercentage:
      value: 100
```

Slika 4.10: Prikaz konfiguracije virtualnog servisa za mračno pokretanje

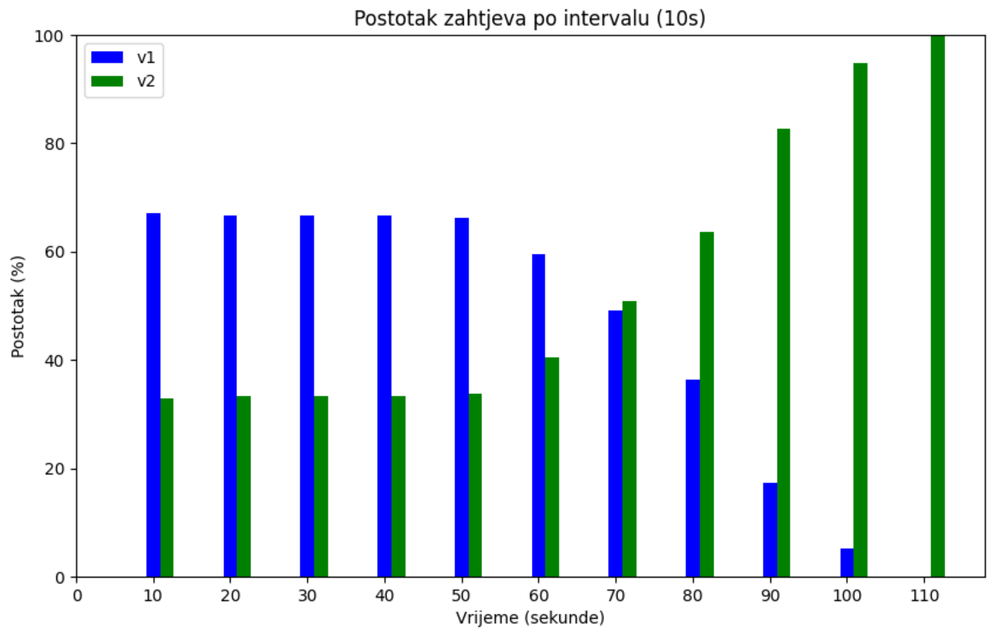
Također je bilo potrebno koristiti drukčiju konfiguraciju virtualnog servisa (slika 4.10) gdje je dodan parametar mirror koji šalje zahtjeve na drugu verziju.



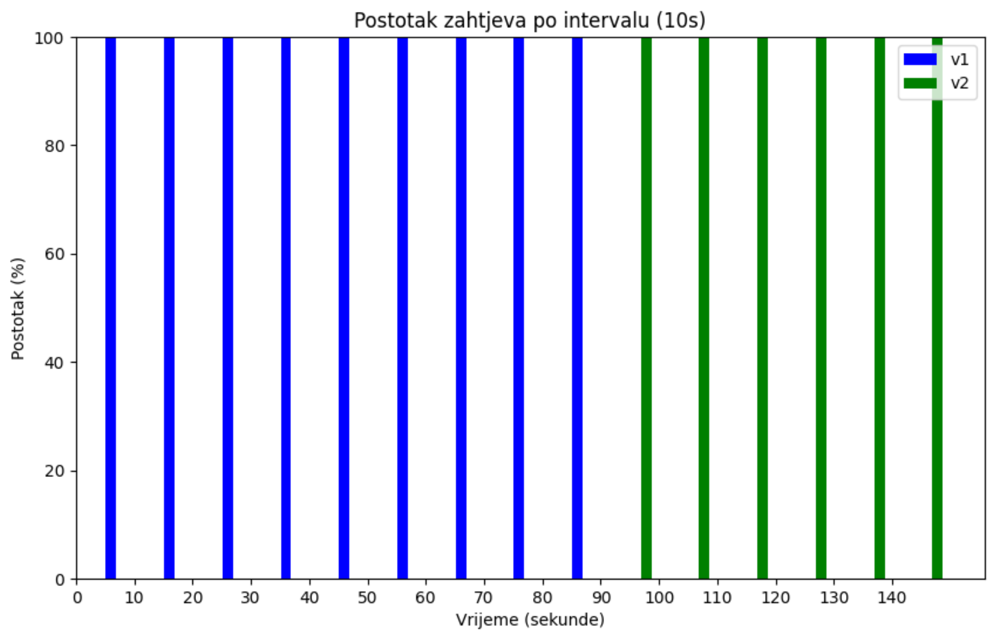
Slika 4.11: Prikaz zahtjeva po sekundi mračne strategije na poslužitelju



Slika 4.12: Prikaz odgovora korisnicima mračne strategije po sekundi



Slika 4.13: Postotak zahtjeva mračne strategije na poslužitelju



Slika 4.14: Postotak odgovora mračne strategije korisnicima

Za opis ove strategije potrebna su četiri grafa, dva koji prikazuju kako su zahtjevi usmjeravani na strani poslužitelja (slike 4.11 i 4.12), i dva koji prikazuju koja verzija je bila aktivna za krajnje korisnike (slike 4.13 i 4.11). Na grafu zahtjeva na poslužitelj (slika 4.11) vidljivo je da se zahtjevi istovremeno šalju i na prvu i na drugu ("mračnu") verziju sve dok sav promet nije prebačen na drugu verziju, a prva je ugašena. S druge strane na grafu korisničkih (slika 4.12) zahtjeva vidljivo je da su svi korisnici dobivali odgovore samo od prve verzije. Sustav je bio dostupan tijekom cijelog trajanja zamjene.

4.7 Predloženi postupak zamjene verzija servisa

U slučaju kada nam je dostupno dovoljno resursa bilo bi dobro povezati više različitih strategija zamjena tako da se zamjena podijeli u stadije.
U prvom stadiju, ako je moguće, bi aktivna bila mračna strategija jer bi korisnici i dalje vidjeli staru verziju koja je ispravna, a zahtjevi bi bili slani i na novu verziju što nam omogućava uvid u ponašanje nove verzije i pronalazak grešaka.
Drugi stadij bi bila strategija kanarinca te bi mali postotak dobrovoljnih beta korisnika bio usmjeren na novu verziju. Ako dođe do neočekivanih pogrešaka, vrlo lako se može vratit na staru verziju, a ako sve prođe po planu, postepeno se može povećavati broj korisnika, ili čak odmah prijeći na A/B testiranje i treći stadij.
Treći stadij je potpuno gašenje stare verzije i preusmjeravanje svih korisnika na novu verziju. Ako to prođe bez poteškoća zamjena se može smatrati uspješnom.
Postoje i situacije kada se mijenja ili baza podataka, ili cijela struktura aplikacije poput krajnjih točaka ili adresa. U tom slučaju ranije navedene strategije nisu podobne jer su zahtjevi koji se šalju različiti. Tada se primjenjuje plavo/zelena strategija. Njena glavna prednost u ovom slučaju je što su obje verzije aplikacije aktivne te je lagano sve korisnike preusmjeriti sa stare verzije na novu i obrnuto u slučaju problema s novom verzijom.

Zaključak

U današnjem razvoju aplikacija tvrtke su sve više i više fokusirane na korisničko iskustvo. Zbog toga su vrlo česte nadogradnje aplikacija na nove, poboljšane verzije. Pritom je bitno da korisnici uopće ne osjete da je u tijeku zamjena verzije. Tu glavnu ulogu odigravaju strategije zamjene.

Strategije zamjene za cilj imaju minimizirati negativno korisničko iskustvo i vrijeme nedostupnosti poslužitelja kako bi zamjena protekla glatko. Postoje razne strategije zamjene, od kojih sve imaju određene prednosti i nedostatke. Zato bi idealno bilo povezati te strategije kako bi se eliminirali nedostatci. Još jedna važna činjenica koju je potrebno uzeti u obzir jest dostupnost resursa za implementaciju tih strategija. U slučaju kada su nam resursi ograničeni, najbolja je strategija zamjene rekreiranjem jer koristi najmanje resursa, ali onda dolazi do nedostupnosti sustava što se negativno odražava na korisnike, a posljedično i na poslovanje tvrtke.

U realnijem slučaju kada nam je dostupno dovoljno resursa najbolje bio bilo kombinirati mračnu strategiju, strategiju kanarinca ili A/B testiranje ako je moguće, a ako nije preporučuje se plavo/zelena strategija.

U sklopu ovog završnog rada napravljena je simulacija odabranih strategije zamjena.

Sažetak

Testiranje strategija uvođenja novih usluga s pomoću dodatka Istio

Strategije uvođenja novih usluga može se simulirati pomoću K3s platforme i dodatka Istio. Strategije zamjene za cilj imaju minimizirati negativno korisničko iskustvo i vrijeme nedostupnosti poslužitelja kako bi zamjena protekla glatko. U ovom završnom radu testirane su strategije zamjena: plavo/zelena, rekreiranje, kanarinac, kotrljajuća, A/B testiranje i mračno pokretanje. U idealnom slučaju kako bi maksimizirali njihovu učinkovitost poželjno je koristiti više strategija zajedno.

Ključne riječi: Kubernetes, K3s, Istio, strategije zamjena

Summary

Testing of service deployment strategies by using Istio

Deployment strategies can be simulated using the K3s plaform and the Istio add-on. These strategies aim to minimise negative user experience and the server downtime in order for the replacement to go smoothly. In this paper, the following deployment strategies were tested: blue/green, recreation, cannary, rolling, A/B tetsing, dark launch. Ideally, to maximise their effectivnes, it is recommended to combine multiple strategies together.

Keywords: Kubernetes, K3s, Istio, deployment strategies

Literatura

[1] Containers vs VMs, 13. prosinac 2023., https://www.redhat.com/en/topics/containers/containers-vs-vms, [Pristup 10. lipanj 2024.].

[2] I. Buchanan, Containers vs. virtual machines , https://www.atlassian.com/microservices/cloud-computing/containers-vs-vms, [Pristup 10. lipanj 2024.].

[3] S. Susnjara, I. Smalley, What is Docker? , 6. lipanj 2024., https://www.ibm.com/topics/docker, [Pristup 10. lipanj 2024.].

[4] Kubernetes Tutorial, https://www.tutorialspoint.com/kubernetes, [Pristup 10. lipanj 2024.].

[5] https://kubernetes.io/images/docs/kubernetes-cluster-architecture.svg, [Pristup 10. lipanj 2024.].

[6] Cluster Architecture, https://kubernetes.io/docs/concepts/architecture/, [Pristup 10. lipanj 2024.].

[7] K3s - Lightweight Kubernetes, 3. lipanj 2024., https://docs.k3s.io/, [Pristup 10. lipanj 2024.].

[8] Architecture, 3. lipanj, 2024., https://docs.k3s.io/architecture, [Pristup 10. lipanj 2024.].

[9] kubelet, 25. travanj 2024., https://kubernetes.io/docs/reference/command-line-tools-reference/kubelet/, [Pristup 10. lipanj 2024.].

[10] Nodes, 24. travanj 2024., https://kubernetes.io/docs/concepts/architecture/nodes/, [Pristup 10. lipanj 2024.].

[11] What is Istio?, https://cloud.google.com/learn/what-is-istio, [Pristup 10. lipanj 2024.].

[12] Architecture, https://istio.io/latest/docs/ops/deployment/architecture/arch.svg, [Pristup 10. lipanj 2024.].

[13] Architecture, https://istio.io/latest/docs/ops/deployment/architecture/, [Pristup 10. lipanj 2024.].

[14] Istio architecture, *Istio architecture: 4 key components, multi-cluster and more* https://www.solo.io/topics/istio/istio-architecture/, [Pristup 10. lipanj 2024.].

[15] A. Uhrin, Deployment Strategies, 6. svibanj 2024. https://www.baeldung.com/ops/deployment-strategies, [Pristup 10. lipanj 2024.].

[16] Top 6 Kubernetes Deployment Strategies and How to Choose, https://codefresh.io/learn/kubernetes-deployment/top-6-kubernetes-deployment-strategies-and-how-to-choose/, [Pristup 10. lipanj 2024.].

[17] https://istio.io/v1.6/docs/ops/configuration/telemetry/monitoring-multicluster-prometheus/external-production-prometheus.svg, [Pristup 10. lipanj 2024.].

[18] Kiali architecture , 5. prosinac 2023., https://kiali.io/docs/architecture/architecture/, [Pristup 10. lipanj 2024.].

[19] K3s Deployment Guide, https://docs.openeuler.org/en/docs/22.03_LTS_SP1/docs/K3s/K3s-deployment-guide.html, [Pristup 10. lipanj 2024.].

[20] Destination Rule, https://istio.io/latest/docs/reference/config/networking/destination-rule/, [Pristup 10. lipanj 2024.].

[21] Gateway, https://istio.io/latest/docs/reference/config/networking/gateway/, [Pristup 10. lipanj 2024.].

[22] Virtual Service, https://istio.io/latest/docs/reference/config/networking/virtual-service/, [Pristup 10. lipanj 2024.].

Upute za instalaciju

Korišten je operativni sustav Artix Linux i upravitelj servisima OpenRC.

Instalacija k3s

```
#instalacija
curl -sfL https://get.k3s.io | sh -

#pokretanje servisa ako nije pokrenut
sudo rc-service k3s start

#provjera statusa čvorova
sudo kubectl get nodes
```

Listing 1: Instalacija k3s

Instalacija Istio

Potrebno je dodati lokaciju instalacije Istia u PATH varijablu kako bi bilo moguće globalno koristiti istioctl komandu. Problem kod toga je što vrijednost PATH varijable nije očuvana prilikom pokretanja novih terminala. Kako bi se taj problem riješio potrebno je u datoteku `~/.bashrc` dodati liniju

```
export PATH={putanja-istio-direktorija}/bin:$PATH
```

kako bi naredba Istioctl bila globlano dostupna.

```
#preuzimanje
curl -L https://istio.io/downloadIstio | sh -

#prebacivanje u instalirani direktorij (ćmogue odstupanje u verziji)
cd istio-1.21.0

#dodavanje istio u PATH varijablu
export PATH=$PWD/bin:$PATH

#instalacija
istioctl install --set profile=demo -y
  --kubeconfig /etc/rancher/k3s/k3s.yaml
```

Listing 2: Instalacija istio

Instalacija Istio Dodataka

Potrebno je biti u direktoriju .../Istio/samples/addons.

```
sudo kubectl apply -f kiali.yaml
sudo kubectl apply -f prometheus.yaml
sudo kubectl apply -f jaeger.yaml
sudo kubectl apply -f grafana.yaml

sudo kubectl port-forward svc/prometheus 9090:9090 -n istio-system
sudo kubectl port-forward svc/kiali 20001:20001 -n istio-system
sudo kubectl port-forward svc/grafana 3000:3000 -n istio-system
```

Listing 3: Instalacija dodataka

Dodavanje servisa

```
sudo kubectl apply -f <ime yaml datoteke>
```

Listing 4: Dodavanje servisa

Namespace

```
sudo kubectl create namespace istio-test
sudo kubectl label namespace istio-test istio-injection=enabled

#provjera
sudo kubectl get namespace -L istio-injection
```

Listing 5: Namespace

Korisne naredbe

```
#IP adresa
sudo kubectl get pods -o wide -n istio-test

#slanje zahtjeva
hey -n 1000 -c 10 http://10.42.0.65/get
```

Listing 6: Korisne naredbe

Testiranje

```
#port
sudo kubectl get svc istio-ingressgateway -n istio-system

#ip
sudo kubectl get nodes -o wide

#slanje
hey -n 1000 -c 10 http://<ip>:port
```

Listing 7: Testiranje