VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
INFORMATIKOS INSTITUTAS
PROGRAMŲ SISTEMŲ BAKALAURO STUDIJŲ PROGRAMA

Paslaugų tinklo platformų taikymas mikroservisų architektūroje

Application of Service Mesh Platforms in a Microservice Architecture

Bakalauro baigiamasis darbas

Atliko:	Jaroslav Avižen	(parašas)
Darbo vadovas:	partn. doc. Liutauras Ričkus	(parašas)
Darbo recenzentas:	lekt. Artūras Jankus	(parašas)

Vilnius – 2022

Santrauka

Mikroservisų architektūros adaptavimas kuriant dideles paskirstytas programų sistemas priverčia programuotojus ne tik įgyvendinti atskirų servisų verslo logiką, tačiau ir pasirūpinti tarpinio sluoksnio problemomis siekiant užtikrinti saugią, patikimą bei greitą komunikaciją tarp servisų. Sprendžiant šias problemas iš kodo pusės yra rizika paveikti bendrą sistemos kompleksiškumą priežiūros etape dėl poreikio valdyti pakeitimus kiekvieno serviso kontekste. Šiai problemai išspręsti atsirado paslaugų tinklo technologija, leidžianti standartizuoti šiuos nefunkcinius reikalavimus gana aukštame abstrakcijos lygmenyje.

Šiame darbe yra apžvelgiama paslaugų tinklo technologija bei palyginamos „Istio“ ir „Linkerd“ paslaugų tinklo platformos sprendžiant išsikeltas mikroservisų problemas. Darbe yra gilinamasi į tris tarpinio sluoksnio problemų sritis – programų sistemos stebimumą, saugumą bei srauto valdymą.

Praktiškai pritaikant „Istio“ ir „Linkerd“ paslaugų tinklo platformas modeliuojamos sistemos atžvilgiu buvo pristatyti rezultatai, atliktų eksperimentų dėka, siekiant nustatyti, kuri iš dviejų platformų geriausiai susitvarko su išsikeltais scenarijais.

Remiantis gautais rezultatais buvo išsiaiškinta, jog turint daugiau infrastruktūros resursų bei labiau patyrusių programuotojų „Istio“ gali būti tinkamesnis sprendimas nei „Linkerd“ dėl savo funkcionalumo išbaigtumo. Tačiau, „Linkerd“ yra greitesnė, taupesnė bei lengviau konfigūruojama platforma, kas sutaupo nemažai laiko ir infrastruktūros resursų, įdiegiant ją į savo mikroservisų architektūros aplikacijas.

Raktiniai žodžiai: paslaugų tinklas, paslaugų tinklo platformos, mikroservisai, Linkerd, Istio

Summary

Adaptation of microservice architecture while developing large distributed systems, not only forces software engineers to focus on developing services' business logic, but also on solving cross-cutting concerns to ensure safety, resilience and performance in inter-service communication. Covering such concerns in code level might increase overall complexity in maintenance stage due to the change management in every system's component context. To tackle this issue, service mesh has been introduced to provide standard way of handling such non functional requirements in relatively high abstraction level.

This thesis focuses on presenting service mesh technology and compare „Istio" and „Linkerd" platforms while solving selected microservices problems. Document covers functionality from the three main areas of service mesh – observability, security and traffic management.

Results were presented by conducting experiments on practically applied „Istio" and „Linkerd" in modelled system to find out which of the two platforms performs best in specific scenario.

Based on the obtained results, it can be concluded that in case of more infrastructure resources and more experienced developers, „Istio" might be a better choice due to offered functionality and platform's maturity. However, „Linkerd" appeared to be faster and easier to configure platform, requiring less computing power, which saves both time and infrastructure resources while deploying it to microservice based applications.

Keywords: service mesh, service mesh platforms, microservices, Linkerd, Istio

TURINYS

Įvadas

Tobulėjant debesų technologijoms, mikroservisų architektūra tapo vienu iš pagrindinių kelių kuriant dideles komercines aplikacijas. Organizacijos renkasi šią architektūrą dėl lengvesnės priežiūros, testuojamumo, plečiamumo (angl. *scalability*) bei trumpesnių išleidimo ciklų (angl. *release cycles*). Nepaisant akivaizdžių pranašumų, kuriuos siūlo mikroservisai, atsiranda nemažai naujų iššūkių, kurių nebuvo monolitinių sistemų atveju.

Viena iš potencialių problemų – saugios komunikacijos užtikrinimas tarp mikroservisų. Apsaugojus klasterį ribojant išorinę prieigą užkardos pagalba, nėra užtikrinimas klasterio vidinių komponentų saugumas. Taip pat, prižiūrint paskirstytą sistemą (angl. *distributed system*) yra svarbu turėti efektyvų gedimų lokalizavimo mechanizmą, kad potencialių klaidų paieška būtų greita ir veiksminga. Kitas ne mažiau svarbus aspektas yra paskirstytos sistemos patikimumas. Aplikacijų programavimo sąsajos (angl. *Application Programming Interface*, toliau API) kvietimų atveju yra reikalaujama, kad servisai būtų pasiekiami, tačiau jeigu vienas iš servisų nustotų veikti, kritiška užtikrinti, kad API vartotojo (angl. *API consumer*) užklausa nepasimestų ir būtų sėkmingai pakartota.

Būtent dėl šių priežasčių atsirado technologija, siūlanti sprendimą minėtoms problemoms. Paslaugų tinklas (angl. *service mesh*) yra papildomas infrastruktūros sluoksnis padengiantis su pagrindiniu funkcionalumu nesusijusius aspektus tokius kaip: komunikacija tarp mikroservisų, apkrovos balansavimas, srauto valdymas, saugumo valdymas bei sistemos stebimumas (angl. *observability*).

Anot El Malki ir Zdun šiuo metu trūksta mokslinės literatūros, kuri nagrinėtų paslaugų tinklą mikroservisų kūrimo kontekste [MZ19]. Apart internetinių straipsnių ir dokumentacijų, kuriuose apžvelgiamas konkrečios paslaugų tinklo platformos funkcionalumas [OKe19; SB19], nėra daug kokybiškų mokslinių šaltinių, kuriuose būtų gilinamasi į tai, kokį paslaugų tinklą reikėtų pasirinkti priklausomai nuo panaudos atvejo. Egzistuoja nemažai straipsnių, kurie apžvelgia paslaugų tinklo funkcionalumą iš tos perspektyvos, palaikomas norimas funkcionalumas ar ne [Man19], nesigilinant koks yra įgyvendinimo sudėtingumas bei potencialus klaidų kiekis būsimai sistemai.

Taip pat, yra vykdomi greitaveikos testai, kurių metu yra lyginamos paslaugų tinklo platformos atsižvelgiant į tokias metrikas kaip procesoriaus bei atminties sunaudojimas, bendras atsako greitis [Mor21; Pat21], tačiau nėra nagrinėjami konkretūs testavimo scenarijai, kurie gali įtakoti šiuos įvertinimus. Pavyzdžiui, Thilo Fromm greitaveikos palyginimo tarp „Istio“ ir „Linkerd“ tyrimo rekomendacijose pabrėžė, jog svarbu atsižvelgti ne tik į bendrus įvertinimus, bet ir atlikti lyginimo tyrimą pridedant daugiau testavimo scenarijų [Fro21].

Nepaisant to, literatūroje pabrėžiama, jog paslaugų tinklas yra technologija, suteikianti prieinamą abstrakcijos lygmenį sprendžiant architektūrinius iššūkius, kurie kyla greitai besiplėsiančiose sistemose [AF19; Por21].

Šio darbo tikslas - palyginti skirtingas paslaugų tinklo platformas sprendžiant išsikeltas mikroservisų problemas bei padengiant tris pagrindines šios technologijos sritis – programų sistemos stebimumą, saugumą, srauto valdymą.

Tam, kad pasiekti užsibrėžtą tikslą iškeliami tokie uždaviniai:

- Išnagrinėti paslaugų tinklo technologiją siekiant išsiaiškinti jos veikimo principą, architektūrą, privalomumus bei apribojimus;
- Išanalizuoti bei pritaikyti skirtingų paslaugų tinklo platformų siūlomą funkcionalumą iš programų sistemos stebimumo, saugumo ir srauto valdymo sričių;
- Išskirti tyrimo lyginimo kriterijus bei testavimo scenarijus pagal kuriuos bus vykdoma paslaugų tinklo platformų lyginamoji analizė;
- Sukurti prototipus įdiegiant kelias pasirinktas paslaugų tinklo platformas;
- Atlikti praktinį paslaugų tinklo platformų palyginimą siekiant išsiaiškinti, kuris įgyvendinimas yra pranašiausias duotame testavimo scenarijuje;

Pagrinde programų sistema buvo parašyta „.NET Core 3.1“ [Mic22b] karkase „C#“ [Mic22a] programavimo kalba apart vieno iš servisų, kuris buvo įgyvendintas su „Python“ [Fou22b]. Programavimui buvo naudota „Visual Studio Code“ [Mic22c] aplinka. Programų sistemos supakavimui buvo naudojamas „Docker“ [Doc22] bei konteinerių orkestravimui - „Kubernetes“ [Fou22a]. Lokaliam „Kubernetes“ paleidimui buvo naudojamas „Minikube“ [Min22] - vieno mazgo klasteris. Apkrovos testavimui buvo naudojamas „JMeter“ [Fou21] įrankis.

1. Paslaugų tinklo apžvalga

Paslaugų tinklo terminas buvo apibrėžtas 2017 metais Williamo Morgano, vieno iš „Linkerd“ paslaugų tinklo platformos kūrėjo. Paslaugų tinklas - dedikuotas infrastruktūros sluoksnis įgalinantis saugią, greitą bei patikimą komunikaciją tarp servisų [Mor17].

1.1. Atsiradimas

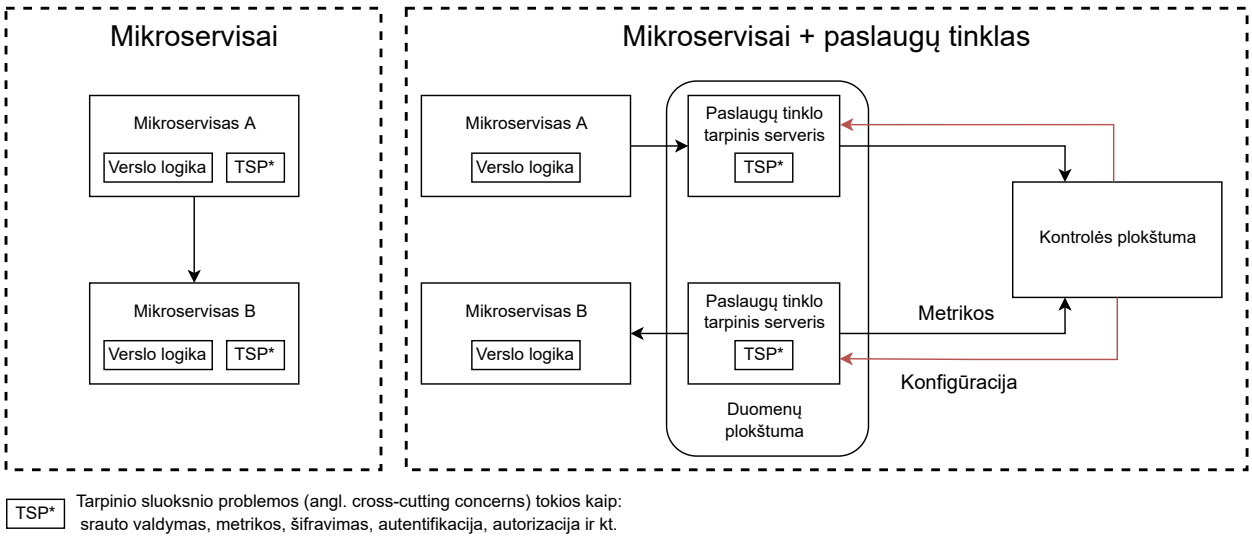
Tipiškai, mikroservisų architektūros programų sistemos susideda iš servisų masyvo, kurie gali komunikuoti tarpusavyje per tinklą nutolusių procedūrinių kvietimų pagalba (angl. *remote procedure calls*, toliau RPC) [LAX+20]. Kadangi RPC vykdomi per tinklą, svarbu valdyti situacijas kuomet užklausa pasimeta dėl tinklo nutraukimo (angl. *network outage*) arba dėl tinklo apkrovos, susijusį su dideliu užklausų kiekiu [Krz21]. Valdyti tokias situacijas iš kodo pusės nėra efektyvu, kadangi šio funkcionalumo įgyvendinimas reikalauja papildomos logikos, kuri dažnai nėra susijusi su pagrindine serviso verslo logika.

Būtent dėl to atsirado paslaugų tinklas palengvinantis programuotojų darbą, kurie iki šiol turėjo valdyti minėtas tinklo rizikas kodo pusėje.

1.2. Architektūra

Paslaugų tinklas susideda iš dviejų pagrindinių komponentų – duomenų plokštumos (angl. *data plane*) bei kontrolės plokštumos (angl. *control plane*) [Cal20]. Tačiau, paslaugų tinklo architektūra gali skirtis priklausomai nuo konkretaus įgyvendinimo, dėl to kai kuriuose šaltiniuose yra pristatomas trečias komponentas – valdymo plokštuma (angl. *management plane*).

1 pav. parodoma, jog iki paslaugų tinklo, tarpinio sluoksnio problemos (angl. *cross cutting concerns*) buvo integruotos šalia atskirų servisų verslo logikos. Paslaugų tinklo atveju jie yra iškeliami į atskirus paslaugų tinklo tarpinius serverius.



1 pav. Klasikinės mikroservisų architektūros bei mikroservisų architektūros su paslaugų tinklo diagrama [KKM+22]

1.2.1. Duomenų plokštuma (angl. *data plane*)

Duomenų plokštuma yra kitaip žinoma kaip tarpinių serverių sluoksnis (angl. proxying layer) [Cal20]. Šioje plokštumoje vyksta tokie procesai kaip servisų atradimas (angl. service discovery), veikimo patikrinimai (angl. *healthcheck*), tinklo srauto valdymas, maršruto parinkimas (angl. *routing*), metrikų surinkimas, saugumo užtikrinimas.

Taip pat šiame sluoksnyje vyksta ateinančių į paslaugų tinklą (angl. *ingressing*) ir išeinančių (angl. *egressing*) užklausų valdymas. Aplikacijos srautas susiduria su interceptoriumi, kuris gaudo visas užklausas, atlieka nustatytas veiklas bei nukreipia srautą į reikiamą servisą. Tokiu būdu, yra užtikrinama viena iš SOLID [1] taisyklių – vienos atsakomybės t.y. verslo logika pasirūpina jai skirtas servisas, o už tinklo logiką yra atsakingas interceptorius arba šiame darbe vadinamas tarpinis serveris.

1.2.2. Kontrolės plokštuma (angl. *control plane*)

Iš pavadinimo galima suprasti, jog tai yra paslaugų tinklo kontrolės arba administravimo sluoksnis. Jis yra atsakingas už konfigūracijos valdymą ir kontrolę tarp visų paslaugų tinklo tarpinių serverių esančių duomenų plokštumoje. Tokiu būdu atskiri tarpiniai serveriai yra agreguojami į sutvarkytą rinkinį – paslaugų tinklą. Be to, šioje plokštumoje yra nustatomos komunikacijos tarp servisų taisyklės, saugumo reikalavimai, renkama telemetrija (angl. *telemetry*).

Dažnai šioje plokštumoje konfigūracijos pakeitimai yra valdomi komandinės eilutės (angl. *Command Line Interface*, toliau CLI), API arba vartotojo sąsajos pagalba. Toks konfigūracijos valdymas įgalina automatizavimą bei pakeitimų diegimą pasinaudojant CI/CD įrankiais [CB19].

1.2.3. Valdymo plokštuma (angl. *management plane*)

Valdymo plokštuma yra aukščiausias paslaugų tinklo sluoksnis, kuris gali būti atsakingas už papildomas funkcijas tokias kaip valdymo šablonai, verslo sistemų integracija bei aplikacijos logikos pagerinimą įdiegus skirtingas paslaugų tinklo platformas [Cal20].

1.3. Diegimo galimybės

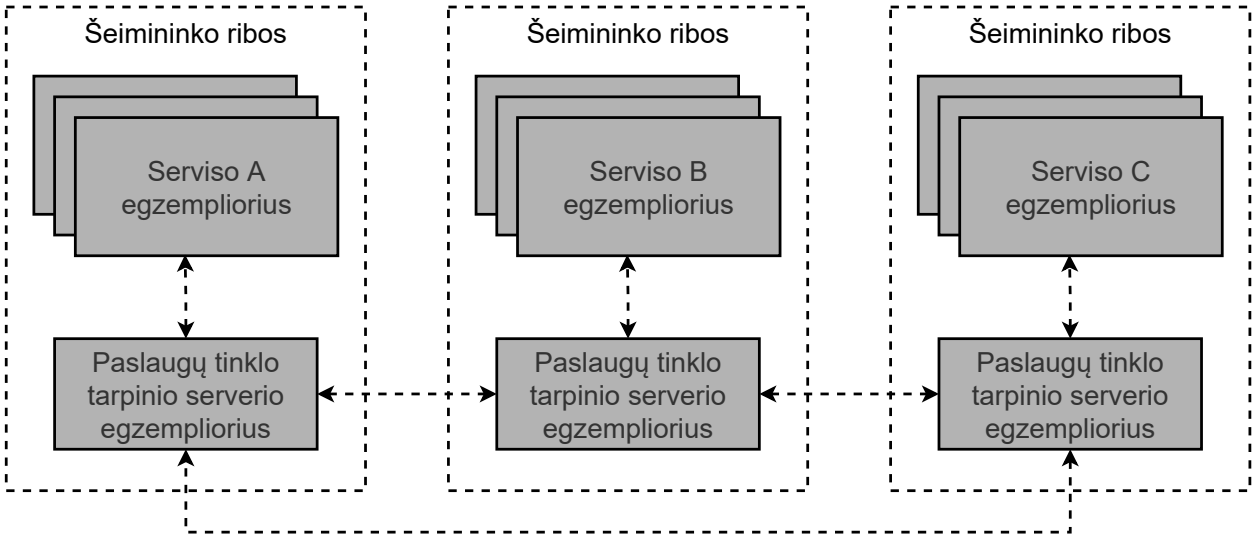
Iš esmės egzistuoja 2 pagrindiniai paslaugų tinklo diegimo šablonai – šoninės mašinos tarpinio serverio (angl. *sidecar proxy*) arba šeimininko tarpinio serverio (angl. *per-host proxy*) strategija.

1.3.1. Šeimininko tarpinio serverio (angl. *per-host proxy*)

Tokio šablono atveju paslaugų tinklas yra diegiamas kiekvienam šeimininkui (angl. *host*). Šeimininkas šiuo atveju gali būti virtuali mašina, fizinis serveris ar „Kubernetes“ darbuotojo mazgas (angl. *worker node*).

[1] Anagrama apibrėžianti penkis objektinio programavimo principus: S: vienos atsakomybės principas O: atvirumo-uždarumo principas L: Liskov pakeitimo principas I: sąsajos atskyrimo principas D: priklausomybių apgręžimo principas [Mar03]

2 pav. yra iliustruojamas atvejis kuomet kiekvienas servisas turi kelis egzempliorius (angl. *service instance*), tačiau tik vieną paslaugų tinklo tarpinį serverį, kuris yra įdiegtas vieno šeimininko kontekste [Tiw17].

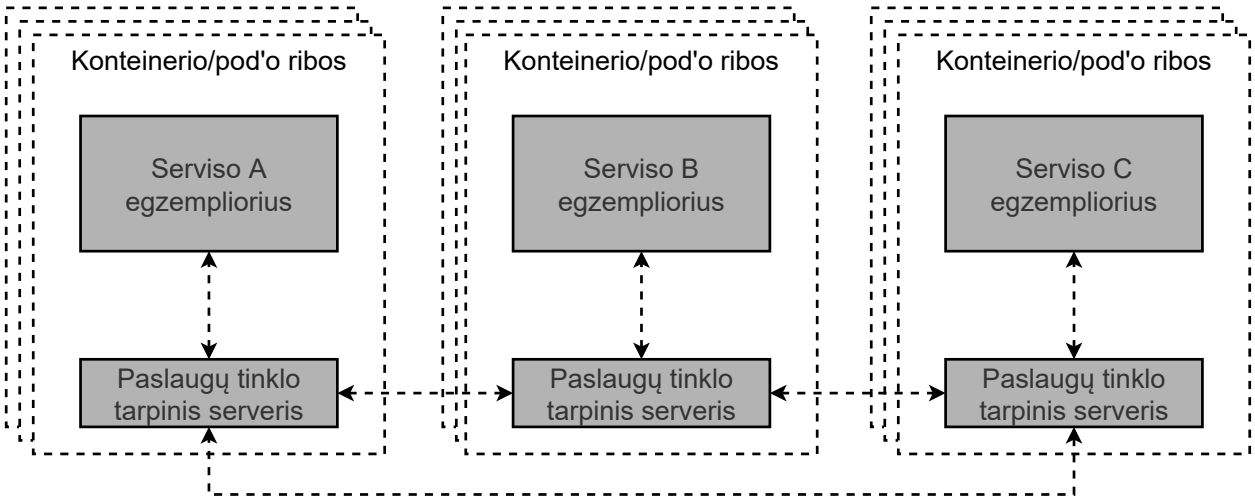


2 pav. Šeimininko tarpinio serverio šablonas. Šaltinis: [Tiw17]

1.3.2. Šoninės mašinos tarpinis serveris (angl. *side-car proxy*)

Šiame šablone kiekvienas šoninės mašinos tarpinis serveris yra diegiamas kartu su kiekvienu serviso egzemplioriumi. Svarbu paminėti, jog šis būdas tinka atvejams, kuomet yra naudojami „Docker“ konteineriai arba „Kubernetes“ pod‘ai.

3 pav. yra parodomas atvejis, kai kiekvienas serviso egzempliorius turi savo paslaugų tinklo šoninės mašinos tarpinį serverį, kuris yra „Kubernetes“ pod‘o arba „Docker“ konteinerio dalis.



3 pav. Šoninės mašinos tarpinio serverio šablonas. Šaltinis: [Tiw17]

1.4. Pagrindinės savybės

Iš esmės paslaugų tinklas buvo sukurtas tam, kad padengti tam tikras sritis, kurios yra svarbios mikroservisų architektūros aplikacijoms arba paskirstytoms sistemos. Šiame poskyryje bus apžvelgiamos pagrindinės savybės, kurias ši technologija padengia.

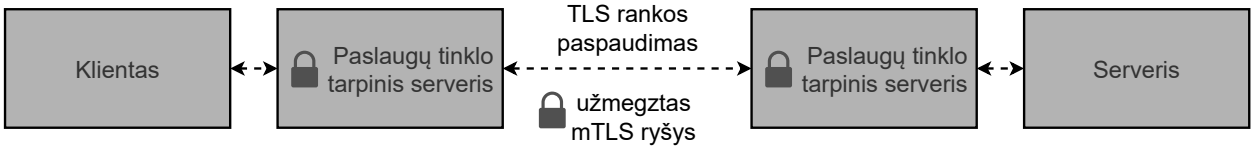
1.4.1. Saugumas

Saugumo aspektas yra vienas iš pagrindinių reikalavimų kuriant modernias programų sistemas. Dažnai yra susiduriama su tuo, kad aplikacijos yra nulaužiamos iš klasterio išorės kenkėjiškų programų, botų tinklo, slaptažodžių žvejybos (angl. *phishing*) arba DDoS (angl. *distributed denial-of-service*) atakų pagalba [DHB+21]. Tačiau svarbu yra suprasti, jog grėsmės gali kilti ne tik klasterio išorės, bet ir klasterio viduje [KBB+21]. Todėl paslaugų tinklas padengia tiek autentifikacijos, tiek autorizacijos reikalavimus.

Šoninės mašinos tarpinis serveris geba inicijuoti abipusį transporto sluoksnio saugumą (angl. *mutual transport layer security*, toliau - mTLS) bei užtikrinti srauto šifravimą tarp servisų nereikalaujant papildomo įgyvendinimo ar pakeitimų iš serviso kodo pusės.

Šis saugumo sprendimas sumažina tarpininko (angl. *man-in-the-middle*) atakų tikimybę, kadangi yra reikalaujama, kad visos užklausos šalys (servisai) turėtų galiojančius sertifikatus, kurie pasitikėtų vienas kitu [KMR+19].

4 pav. iliustruoja atvejį kuomet 2 mikroservisai sąveikauja tarpusavyje šoninės mašinos tarpinių serverių pagalba, kur užmezgus mTLS ryšį yra užtikrinama užšifruota komunikacija. Šoninės mašinos tarpiniai serveriai apsikeičia sertifikatais autentifikuodami vienas kitą sertifikato institucijos pagalba (angl. *certificate authority*) [Gup21].



4 pav. mTLS komunikacija šoninės mašinos įgaliotinių pagalba

Taip pat, užmezgus mTLS ryšį, yra galimybė patikrinti autorizacijos taisykles (angl. *authorization policies*), kurios galėjo būti sukonfigūruotos paslaugų tinklo valdymo plokštumoje, siekiant įsitikinti, jog šie mikroservisai turi teisę komunikuoti vienas su kitu.

1.4.2. Srauto valdymas

Paslaugų tinklas taip pat siūlo sprendimus, kurie padeda užtikrinti patikimą (angl. *resilient*) bei konfigūruojama komunikaciją tarp servisų:

1. Vykdymo laiko (angl. *timeout*) valdymas – suteikia konfigūraciją užklausos sustabdymui, kuomet apdorojimas užtrunka per ilgai ir servisas negrąžina klientui atsakymo per nustatytą laiką.

2. Kritinės ribos (angl. *deadline*) valdymas – leidžia apibrėžti ne tik serviso, bet konkrečios serviso funkcijos vykdymo laiką (angl. *feature timeout*).
3. Pakartojimų (angl. *retries*) valdymas – pakartojimai su eksponentiškai augančiu laiku, galimybę valdyti pakartojimus pagal vykdymo laiką ir kritinę ribą, nustatyti pakartojimų limitus.
4. Grandinės nutraukimas (angl. *circuit breaking*) – galimybė nutraukti užklausų „grandinę“ neleidžiant užklausų vykdymą pasirinktam servisui.
5. Užklausų ribojimas (angl. *rate limiting*) – papildomas srauto kontrolės būdas leidžiantis riboti užklausų kiekį tam, kad užtikrinti servisų stabilumą, kuomet programų sistema patiria didelę apkrovą.

1.4.3. Stebimumas (angl. *observability*)

Kitas ne mažiau svarbus aspektas kalbant apie paskirstytas sistemas yra programų sistemos stebimumas. Klaidų radimas sudėtingose sistemose gali būti komplikuotas, todėl paslaugų tinklas siūlo kelis sprendimus. Beje, tai yra viena iš esminių priežasčių kodėl yra renkamasi paslaugų tinklą.

1. Trasavimas (angl. *tracing*) – paskirstytose sistemose užklausa yra apdorojama keliuose servisuose, kas apsunkina klaidų radimą. Taip pat, servisų kiekis aplikacijose gali būti tiek didelis, kad be papildomų įrankių tampa neaišku kaip servisai sąveikauja tarpusavyje ir koks yra užklausos apdorojimo scenarijus. Dėl to paslaugų tinklas leidžia pridėti trasavimą surenkant reikiamus duomenis ir nusiunčiant į trasavimą palaikančias sistemas tokias kaip „Jaeger“ ar „Zipkin“.
2. Žurnalizavimas (angl. *logging*) – sprendimas leidžiantis žurnalizuoti HTTP užklausas, statuso kodus bei standartinę išvestį iš šoninių mašinų tarpinių serverių.
3. Metrikos – funkcija leidžianti standartizuoti metrikų gavimą be aplikacijos kodo modifikavimo, kuomet servisai privalo skleisti (angl. *emit*) metrikas pagal tam tikrą logiką. Turint standartizuotas ir nuoseklias metrikas klasteryje, yra galimybė įvesti tokias funkcijas kaip automatinį plečiamumą (angl. *autoscaling*).

1.5. Paslaugų tinklo platformos

Paslaugų tinklas yra lyg architektūrinis standartas arba pagal apibrėžimą – dedikuotas infrastruktūros sluoksnis. Todėl kaip ir kitos standarto tipo idėjos, paslaugų tinklas turi savus įgyvendinimus, kurias šiame darbe vadinsime paslaugų tinklo platformos.

Šiuo metu egzistuoja nemažai paslaugų tinklo platformų tokių kaip „Istio“, „Linkerd“, „AWS App Mesh“, „Consul“, „Traefik Mesh“, „Kuma“ ir kitų [KKM+22]. Tačiau renkantis tam tikrą platformą yra būtina įsigilinti į kiekvienos pranašumus bei apribojimus priklausomai nuo panaudos atvejo. Šiame darbe paslaugų tinklo platformos pasirinkimas buvo motyvuotas tokiais kriterijais:

- Platforma yra atviro kodo - tokio tipo programų sistemos yra žinomos kodo skaidrumu bei galimybe prisidėti prie vystymo, kas įgalina greitą kūrimą ir klaidų radimą dėl didelės programuotojų bendruomenės [JVR08];
- Naudojama produkcinėje aplinkoje – tokiu būdu yra užtikrinama, jog bus palyginamos tik brandžios platformos;
- Palaiko „Kubernetes“ platformą – anot tyrimo atlikto „Cloud Native Computing Foundation“ organizacijos, 78% apklausos respondentų renkasi „Kubernetes“ konteinerių orkestravimui [Miy21];

Šiuos kriterijus tenkina 3 paslaugų tinklo platformos – „Istio“, „Consul“ ir „Linkerd“. Tačiau atsižvelgiant į tai, jog „Istio“ bei „Consul“ naudoja tą pačią „Envoy“ technologiją kaip tarpinį serverį [KKM+22], buvo nuspręsta pasirinkti tik vieną iš jų – „Istio“ dėl to, kad tarp brandžiausių platformų sąrašo atsižvelgiant į siūlomą funkcionalumą bei bendruomenės aktyvumą šiuo metu yra „Istio“ ir „Linkerd“ [LLG+19].

2. Tyrimas

2.1. Lyginimo kriterijų nustatymas

Paslaugų tinklo platformos bus lyginamos pagal šiuos kriterijus:

1. Greitaveikos testas – įvertinti paslaugų tinklo platformų greitaveiką naudojantis šiomis metrikomis:
    - Vidutinis P99 (angl. *99 latency percentile*) atsako greitis;
    - Vidutinis CPU sunaudojimas;
    - Vidutinis atminties sunaudojimas;
    - Vidutinis klaidų skaičius;
2. Įgyvendinimo sudėtingumas – įvertinti sprendimo sudėtingumą naudojantis šiomis metrikomis:
    - Resursų skaičius - kiek buvo sukurta Kubernetes(K8S) resursų, CRD resursų bei kitų konfigūracijos failų;
    - Konfigūracijos eilučių skaičius - kiek eilučių užėmė aprašyti minėtus resursus;
    - Dokumentacijos aiškumas - skalėje nuo 1 iki 3, kur 1 - neaiški dokumentacija, 2 - gera dokumentacija, tačiau turi trūkumų, 3 - aiški dokumentacija;
    - Apimtis - sugaištas laikas skalėje nuo 1 iki 3, kur 1 - maža apimtis, 2 - vidutinė apimtis, 3 - didelė apimtis;
3. Galimybės bei apribojimai – nustatyti, kokį funkcionalumą paslaugų tinklo platforma siūlo iškeltam testavimo scenarijui ar problemai atsižvelgiant į šias sritis:
    - Stebimumo funkcionalumas;
    - Saugumo funkcionalumas;
    - Srauto valdymo funkcionalumas;

2.2. Serverio parametrai

Eksperimentai buvo atliekami mašinoje su žemiau nurodytais parametrais:

- Procesorius: Intel® Core™ i7-9750H
- Procesoriaus dažnis: iki 4.50 GHz
- Procesoriaus branduolių skaičius: 6
- Atmintis (RAM): 16.0 GB
- Operacinė sistema: Windows
- Operacinės sistemos tipas: 64-bit

2.3. Taikymo atvejai

Tyrimo metu bus pasirenkamas konkretus taikymo atvejis arba problema, kuri pasireiškia mikroservisų architektūroje bei siūlomas sprendimo būdas. Paslaugų tinklo platformų sprendimų efektyvumas bus įvertinamas nustatytais lyginimo kriterijais.

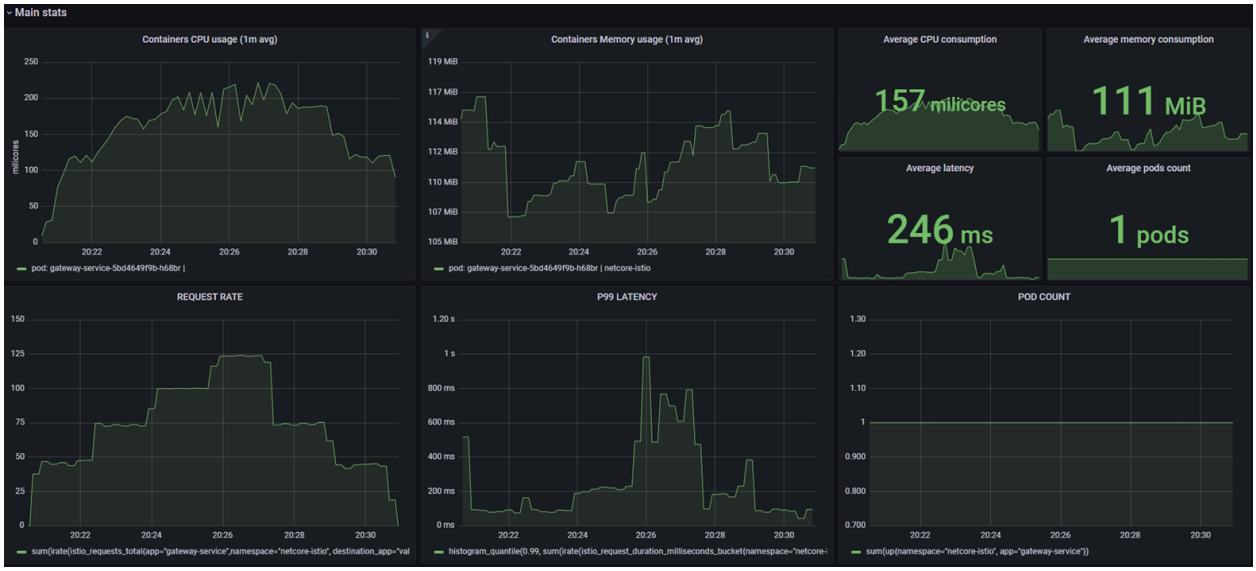
2.3.1. Didelio plečiamumo bei prieinamumo valdymas

2.3.1.1. Problemos apibūdinimas

Kalbant apie dideles paskirstytas sistemas, šis aspektas yra svarbus norint užtikrinti patikimą patirtį sistemos naudotojams. Mikroservisų architektūra suteikia plečiamumo galimybes įdiegus sistemą tokioje aplinkoje kaip „Kubernetes“. Padidėjus apkrovai, vienas iš klasterio servisų gali būti išplėstas tiek vertikaliai padidinant pod‘o resursus, tiek horizontaliai padidinant pod‘ų skaičių. Tačiau, nėra visiškai aišku koks turi būti plečiamumo valdymo algoritmas norint efektyviai plėsti servisus, kaip automatizuoti šį procesą bei kokiu būdu galima gauti metrikas, kurios taptų pasirinkto algoritmo parametrais.

2.3.1.2. Praktinis problemos pavyzdys

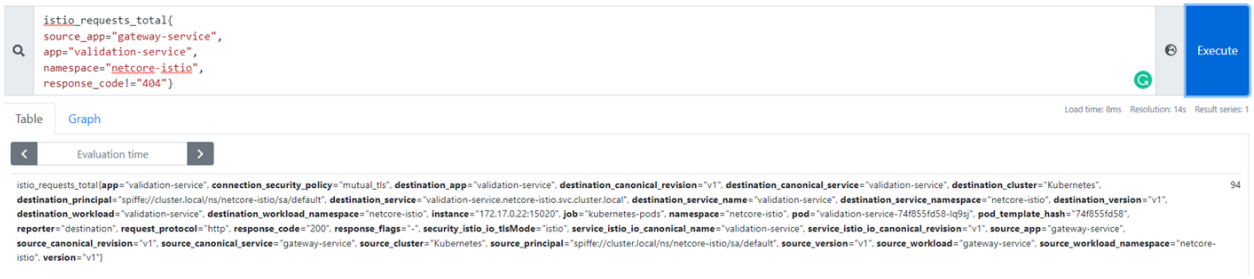
Paleidus apkrovos testą (angl. *load test*) „JMeter“ įrankio pagalba, 5 pav. galima pastebėti, jog apkrovai padidėjus („Request rate“ diagrama), užklausos atsako greitis taip pat didėja („P99 latency“ diagrama). Tai reikštų, kad dalis vartotojų patiria minėtą problemą – sistema nesusitvarko su padidėjusių klientų skaičiumi dėl ko operacijos užtrunka ilgiau.



5 pav. RPS bei užklausos atsako greičio vizualizacija (Istio)

2.3.1.3. Problemos sprendimas – „Istio“ paslaugų tinklas

„Istio“ telemetrijos (angl. *telemetry*) servisas surenka metrikas iš paslaugų tinklo persiųsdamas juos į „Prometheus“. Įdiegus šį servisą, yra galimybė pasiekti bei panaudoti surinktus duomenis automatinio plečiamumo algoritme. Viena iš potencialių metrikų gali būti „istio_requests_total“, kuri reprezentuoja, kiek užklausų ateina į dominantį servisą - 6 pav.



6 pav. „Prometheus“ „istio_request_total“ užklausos rezultatas

Šią metriką galima panaudoti skaičiavimams siekiant nustatyti kiek užklausų per sekundę (angl. *rate per second*, toliau RPS) servisas sulaukia ir pagal ją sukonfigūruoti „Kubernetes“ siūlomą „HorizontalPodAutoscaler“ (toliau HPA) resursą, kuris būtų atsakingas už horizontalų plečiamumą. Pritaikius atitinkamas Prometheus funkcijas, gavome užklausą, kuri grąžina RPS vertę.

```
sum(irate(istio_request_total{app="gateway-service",
namespace="netcore-istio"}[30s])) by (deployment)
```

Iš 6 pav. galima pastebėti, jog apkrovai padidėjus nuo 50 RPS iki 75 RPS, atsako greitis taip pat padidėjo. Todėl vienas iš būdų sukonfigūruoti HPA resursą - nustatant ribinį RPS tašką, kuomet servisą plėstume horizontaliai.

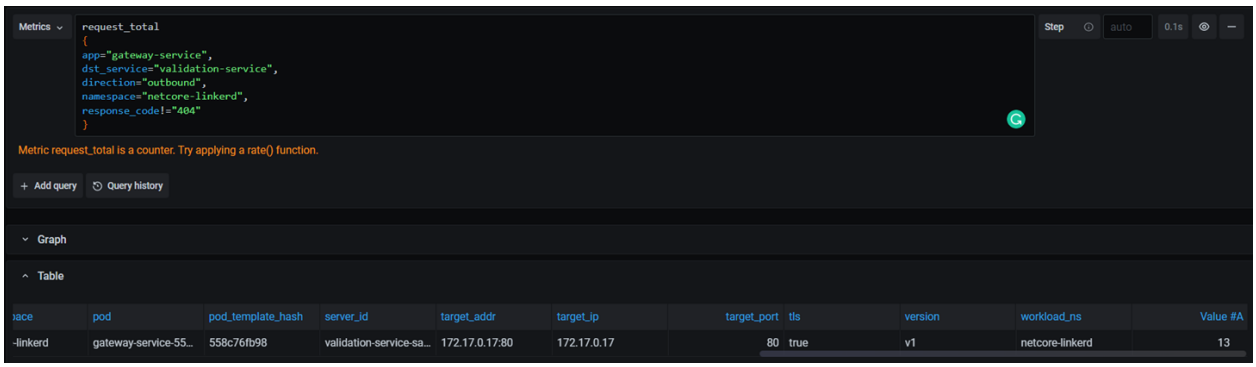
Tačiau šis sprendimas atrodo ne visai optimalus, kadangi tokiu būdu nėra atsižvelgiama į atsako greitį. Gali būti scenarijų kuomet padidinus replikų skaičių pagal RPS vertę, atsako greitis nemažėja arba išlieka toks pat, koks buvo prieš išplėtimą, nepagerinant vartotojų patirties. Taip pat, ribinės RPS reikšmės nustatymas reikalauja išankstinio testavimo tam, kad sužinoti, kokią reikšmę reikėtų pasirinkti, kad sistemos našumas tenkintų nustatytus reikalavimus.

Todėl alternatyvus būdas – sukonfigūruoti HPA nustatant siekiamą atsako greitį. Šiam tikslui „Istio“ siūlo metriką ”istio_request_duration_milliseconds_bucket”. Žemiau esanti „PromQL“ užklausa grąžina P99 atsako greičio duomenis. Siekiant išvengti nereikalingo plėtimo, kuomet servisas patiria trumpą (iki 30 sekundžių) atsako greičio piką, nebūtinai susijusį su apkrovos padidėjimu, buvo pritaikyta papildoma funkcija avg_over_time.

```
avg_over_time(histogram_quantile(0.99, sum(irate(
istio_request_duration_milliseconds_bucket
{namespace="netcore-istio", app="gateway-service"}
[5m])) by (le, app))[30s:])
```

2.3.1.4. Problemos sprendimas – „Linkerd“ paslaugų tinklas

„Linkerd“ taip pat siūlo metrikų API, kurio pagalba yra surenkamos metrikos iš paslaugų tinklo. Viena iš prieinamų – „request_total“, kur pritaikius atitinkamą filtrą yra galimybė sužinoti kiek užklausų buvo siunčiama iš vieno serviso į kitą (7 pav.).



7 pav. ”Prometheus” request_total užklausos rezultatas

Žemiau yra pateikta „PromQL“ užklausa skirta RPS nustatymui vienam iš servisų.

```
sum(irate(request_total{
deployment="gateway-service", namespace="netcore-linkerd",
direction="inbound"}[30s])) by (deployment)
```

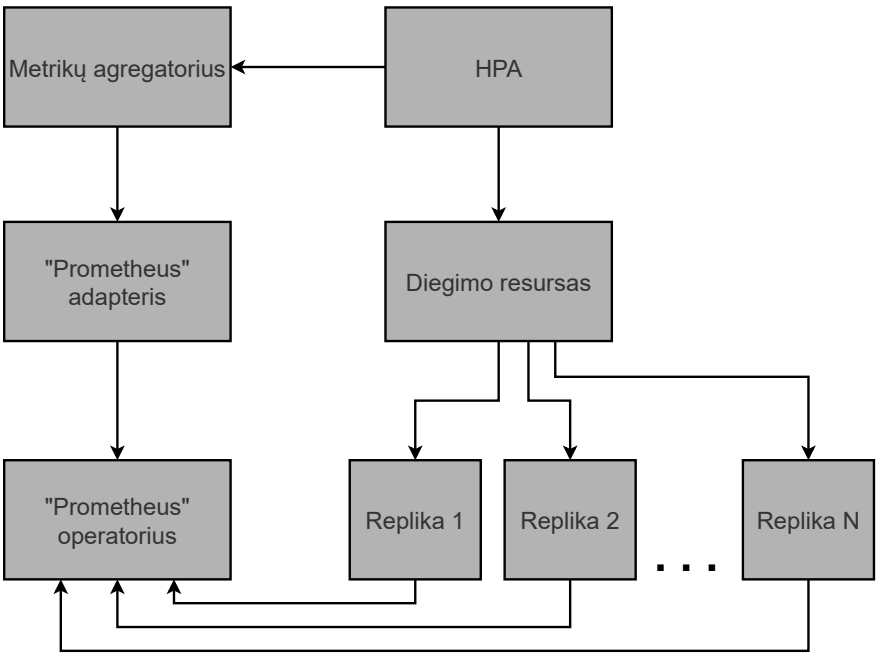
Kita metrika yra skirta atsako greičio matavimui – ”response_latency_ms_bucket”, kuriai taip pat pritaikius atitinkamas Prometheus funkcijas, yra galimybė gauti vidutinį P99 atsako greitį.

```
avg_over_time(histogram_quantile(0.99, sum(irate(
response_latency_ms_bucket{namespace="netcore-linkerd",
deployment="gateway-service",direction="inbound"}[5m]))
by (le, deployment))[30s:])
```

Todėl HPA resurso sukūrimas remiantis šiomis metrikomis yra taip pat yra palaikomas „Linkerd“, panašiai kaip ir „Istio“ paslaugų tinklo platformos.

2.3.1.5. HPA konfigūracijos testavimas

Verta paminėti, jog „Kubernetes“ HPA resursas reikalauja papildomo „Prometheus“ adapterio, kuris įgalina nestandartinių metrikų gavimą t.y. tokių, kurios ateina iš paslaugų tinklo, o ne iš „Kubernetes“ metrikų API. 8 pav. schema iliustruoja tokį scenarijų. Todėl įdiegus „Prometheus“ adapterį, atsiranda galimybė pasinaudoti paslaugų tinklo siūlomomis metrikomis.

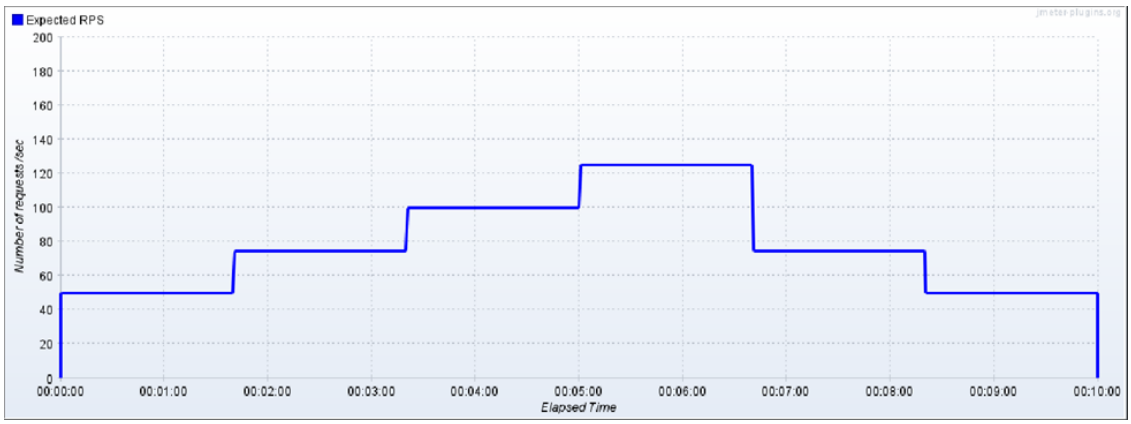


8 pav. Nestandartinių metrikų gavimo schema

Bandymams buvo pasirinktas apkrovos scenarijus, kuomet vartotojų užklausų skaičius didėja kas 60 sekundžių iki tam tikro taško ir vėliau mažėja. Tokiu būdu buvo bandoma simuliuoti atvejį, kuomet sistemą patiria netikėtą apkrovą. Žemiau pateiktoje lentelėje (1 lentelė) yra konfigūracija, kuri buvo nurodyta „JMeter“ įrankiui. 9 pav. iliustruoja šių parametrų vizualizaciją.

RPS intervalo pradžioje	RPS intervalo gale	Trukmė
∼50	∼50	60 s.
∼75	∼75	60 s.
∼100	∼100	60 s.
∼125	∼125	60 s.
∼75	∼75	60 s.
∼50	∼50	60 s.

1 lentelė. JMeter apkrovos testo konfigūracija



9 pav. JMeter apkrovos testo konfigūracijos vizualizacija

Bandymai buvo atlikti 3 scenarijams kartojant kiekvieną 3 kartus siekiant gauti vidutines reikšmes:

- **Scen-HPA1**: Servisas su viena replika (Memory 32Mi-64Mi, CPU: 50m-100m)
- **Scen-HPA2**: Servisas su sukonfigūruotu HPA resursu remiantis RPS metrika, kurios siekiama reikšmė 25 RPS vienai replikai.
- **Scen-HPA3**: Servisas su sukonfigūruotu HPA resursu remiantis P99 atsako greičio metrika, kurios siekiama reikšmė 100 ms.

2.3.2. Autentifikacijos bei autorizacijos taisyklės tarp mikroservisų

2.3.2.1. Problemos apibūdinimas

Komunikacijos saugumo užtikrinimas tarp kelių mikroservisų gali būti užtikrinimas prieigos rakto pagalba (pvz.: JSON Web Token, toliau JWT). Tačiau tapatybės valdymas tokiu būdu reikalauja įsiterpimo į kiekvieno serviso logiką bei kodą. Turint skirtinga programavimo kalba parašytus servisus, programuotojas yra priverstas suprogramuoti M skirtingų įgyvendinimų bei integruoti juos į N servisus, kur M – skirtingų programavimų kalbų skaičius, N – sistemos servisų skaičius.

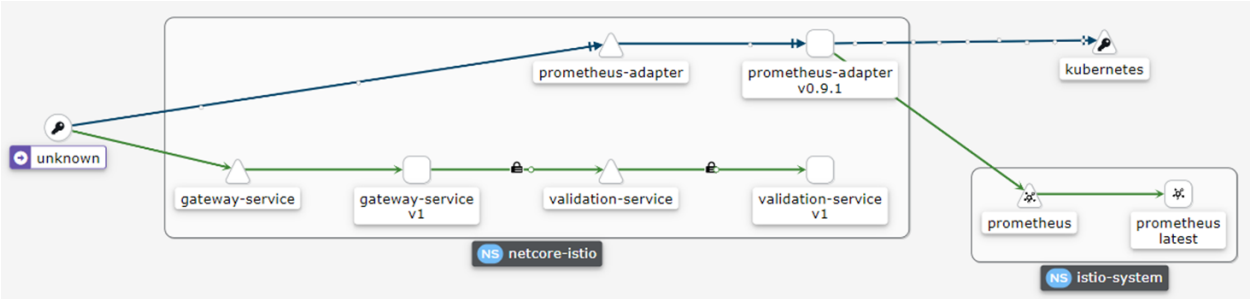
Taip pat, tai galioja atvejams jeigu atsirastų poreikis įvesti tam tikras autorizacijos taisykles, kuomet norėtume servisui A uždrausti kreiptis į servisą B arba jeigu norėtume užšifruoti (angl. *encrypt*) užklausos duomenis, kad tarp serviso A ir B jie nekeliautų paprastu tekstu (angl. *plaintext*).

2.3.2.2. Problemos sprendimas – „Istio“ paslaugų tinklas

„Istio“ siūlo mTLS įgyvendinimą, kur tapatybių valdymo logika būtų pridėta duomenų plokštumoje neliečiant pagrindinio serviso funkcionalumo. Neturint sukonfigūruoto mTLS, yra galimybė kreiptis tiesiai į bet kurį norimą servisą.

Norint uždrausti užklausas į „gateway-service“ „Istio“ leidžia deklaruoti „STRICT“ mTLS scenarijų, kuomet paprasto teksto užklausos nebūtų apdorojamos.

Tačiau kilus poreikiui atskleisti (angl. *expose*) vieną iš norimų servisų klientui, kuris netūrėtų sertifikato, „Istio“ leidžia pridėti „PERMISSIVE“ scenarijų, kas reikštų, kad servisas apdorotų tiek paprasto teksto, tiek užšifruotas užklausas. 10 pav. iliustruoja atvejį kuomet „gateway-service“ buvo priskirtas „PERMISSIVE“ scenarijus, o „validation-service“ – „STRICT“. Tokiu būdu yra leidžiama klientams tiesiai kreiptis į „gateway-service“, o į „validation-service“ galima kreiptis tik turint validžius sertifikatus.



10 pav. mTLS iliustracija tarp skirtingų sistemos servisų

2.3.2.3. Problemos sprendimas – „Linkerd“ paslaugų tinklas

Kalbant mTLS konfigūraciją „Linkerd“ paslaugų platformos atveju, yra reikalaujama papildomų autorizacijos taisyklių, kad mTLS būtų aktyvuotas. Norint sukurti scenarijų parodytą 10 pav., šalia kiekvieno serviso reikėtų pridėti anotaciją, kas leistų užklausas tik tarp tų servisų, kurie yra valdomi paslaugų tinklo:

```
config.linkerd.io/default-inbound-policy: all-authenticated
```

Pabandžius išsiųsti užklausą į validacijų servisą, klientas gauna 403 statusą. „Linkerd“ duomenų plokštumoje ši užklausa yra sustabdoma dėl taisyklės pažeidimo. Todėl išsiuntus paprasto teksto užklausą neturint validaus sertifikato servisui, kuriam buvo sukonfigūruota „all-authenticated“ taisyklė „Linkerd“ tarpinis serveris parodo pranešimą - „Request failed error=unauthorized connection on server default:all-authenticated“

Norint leisti paprasto teksto užklausas, reikėtų pridėti „Server“ ir „ServerAuthorization“ resursus, kur reikėtų specifikuoti taisyklę „unauthenticated: true“. Skirtingai nuo „Istio“, „Linkerd“ leidžia sukonfigūruoti šią taisyklę pasirinktam portui.

11 pav. Sėkmingos mTLS konfigūracijos vizualizacija

2.3.3. Aplikacijų srauto valdymas

2.3.3.1. Problemos apibūdinimas

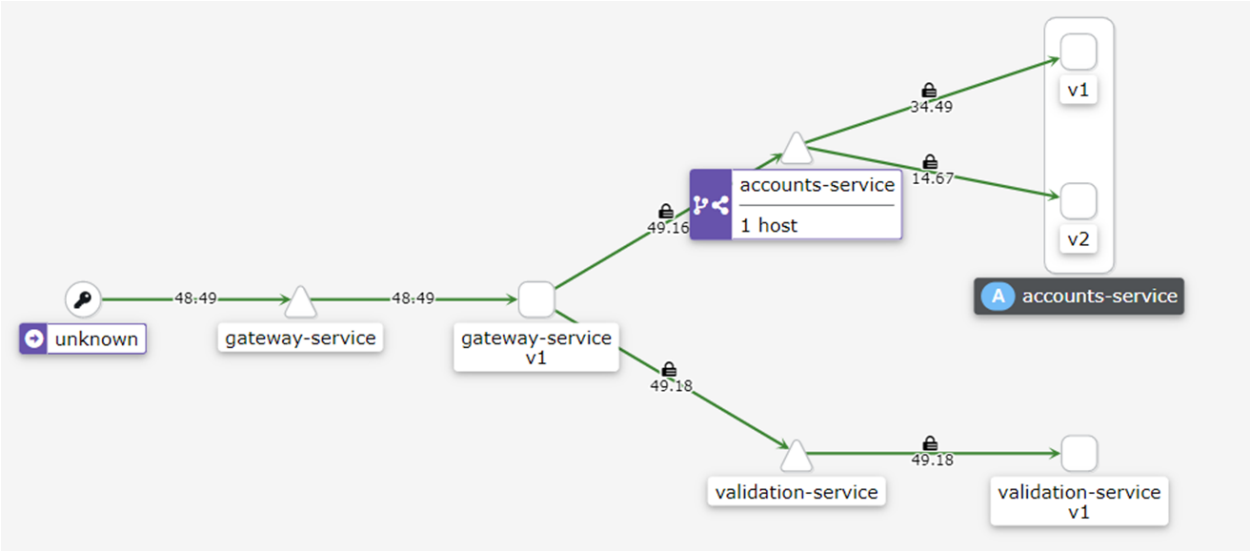
Kalbant apie naujo funkcionalumo įvedimą yra svarbu užtikrinti, jog produkcijoje sistema veiktų sklandžiai. Iš kodo pusės yra galimybė integruoti funkcionalumo žymes (angl. *feature flags*) kuomet naudojantis aplinkos kintamuoju (angl. *environment variable*) yra galimybė įjungti tam tikrą pilotinį funkcionalumą ir išjungti kilus sutrikimams. Tačiau, situacija tampa sudėtingesnė,

kuomet kyla poreikis pakeisti technologijas įvedant naują servisą, kuris ateityje turėtų pakeisti esamą. Pavyzdžiui, jeigu dėl tam tikrų priežasčių buvo nuspręsta migruoti funkcionalumą iš serviso parašyto „.NET“ karkaso pagalba į „Python“. Tokiais atvejais, pakeitimai iš kodo pusės nėra trivialūs ir tokius scenarijus tenka valdyti iš tinklo srauto pusės. Vienas iš būdų – implementuojant „Kubernetes“ siūlomą „canary“ diegimo strategiją, kitas – pasinaudojant paslaugų tinklo platformų teikiamu funkcionalumu.

2.3.3.2. Problemos sprendimas – „Istio“ paslaugų tinklas

„Istio“ siūlo kelis būdus, kaip galima būtų valdyti srautą nukreipiant vartotojų užklausas pagal norimą logiką. Pasirinkus nukreipimą pagal svorius, vartotojo užklausos bus nukreipiamos atitinkamai į servisus, kurie turi didesnį svorį.

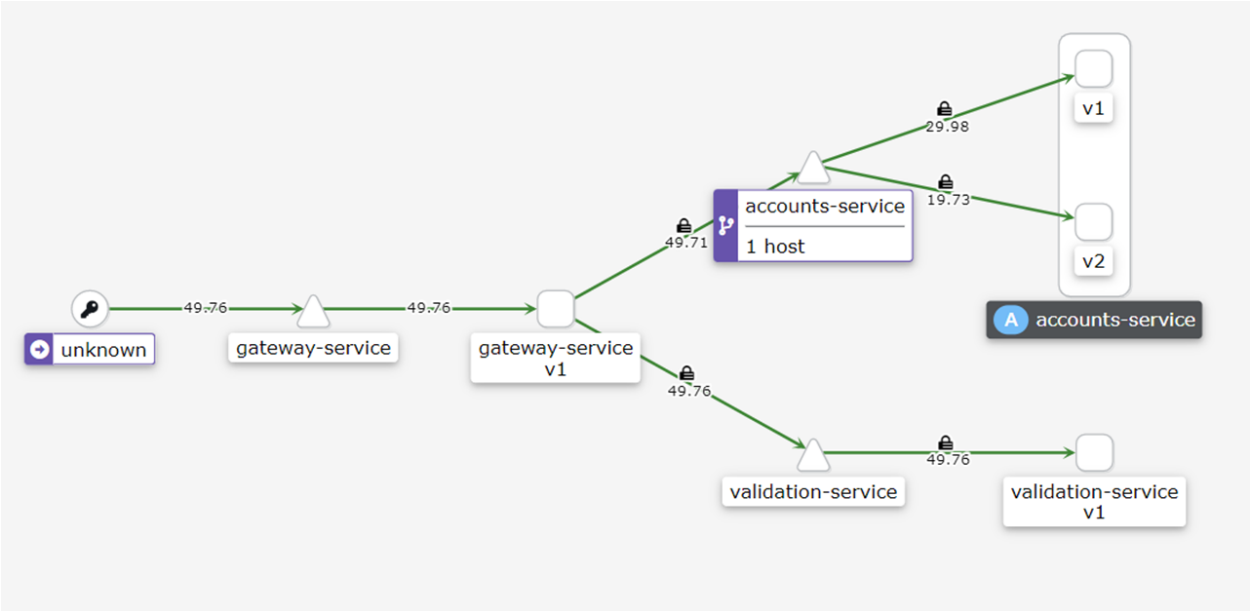
12 pav. iliustruoja minėtos problemos scenarijų, kuomet kilo reikalavimas migruoti „accounts-service“ funkcionalumą iš „.NET“ karkaso į „Python“. Atlikus konfigūraciją, buvo nustatyta, jog 70% sistemos srauto keliaus į dabartinę „accounts-service“ versiją („.NET“) ir 30% į naują. Tokiu būdu tik 30% vartotojų patirs sutrikimų kilus problemoms su nauju servisu. Kuomet bus įsitikinta, jog naujas servisas veikia korektiškai, galima pakeisti konfigūraciją nukreipiant 100% srauto į šį servisą. Verta pastebėti, jog pridėjus papildomus servisus prie esamos sistemos, anksčiau aprašyta mTLS konfigūracija veikia, ką parodo atsiradusios spynelės šalia „accounts-service“ rodyklių.



12 pav. ”Accounts-service” srauto nukreipimas pagal 30% ir 70% svorius (Istio)

Alternatyvus būdas, kurį siūlo „Istio“ yra srauto nukreipimas pagal HTTP užklausos antraštę (angl. *request header*). Ši technika gali būti naudinga, kuomet kyla poreikis nukreipti vartotojų pagal tam tikrą kriterijų, pavyzdžiui, pagal šalį.

13 pav. parodo atvejį, kai 20 RPS ateina iš Danijos vartotojų t.y. tų kurių užklausos antraštėje yra raktas „country“ su reikšme „Denmark“. Šie vartotojai pagal nustatytą konfigūraciją yra sėkmingai nukreipiami į antrą „accounts-service“ versiją, o likusieji – į pirmą.

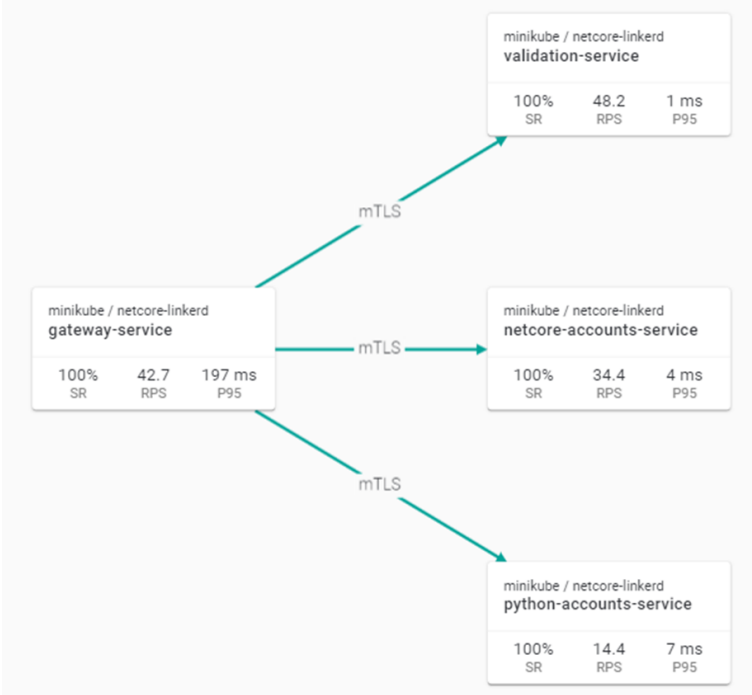


13 pav. ”Accounts-service” srauto nukreipimas pagal ”country” HTTP užklausos antraštę

2.3.3.3. Problemos sprendimas – „Linkerd“ paslaugų tinklas

Kalbant apie „Linkerd“, ši paslaugų tinklo platforma taip pat palaiko užklausų nukreipimą pagal nustatytus svorius. Tačiau skirtingai nuo „Istio“, šis funkcionalumas yra įgyvendintas naudojantis trečios šalies API – „Service Mesh Interface“ (toliau „SMI“).

Pakartojus konfigūraciją parodytą 12 pav., norimas scenarijus taip pat buvo pasiektas. 14 pav. „netcore-accounts-service“ atitinką esamą „accounts-service“ versiją, „python-accounts-service“ – naują.



14 pav. ”Accounts-service” srauto nukreipimas pagal 30% ir 70% svorius (Linkerd)

Kaip ir buvo minėta, „Linkerd“ pasinaudoja „SMI“ API siekiant pasiūlyti srauto nukreipimo funkcionalumą. Dėl šio aspekto atsiranda išorinė priklausomybė nuo trečios šalies produkto, kuris turi tiesioginę įtaką šiam funkcionalumui, kadangi su kiekviena „SMI“ API versija, „Linkerd“ turi atnaujinti savo įskiepio įgyvendinimą. Todėl nors ir „SMI“ API „v1alpha4“ versijoje atsirado galimybė nukreipti srautą pagal HTTP užklausų antraštes panašiai kaip „Istio“ atveju, „Linkerd“ platformoje to padaryti neįmanoma, kadangi paskutinė palaikoma versija yra „v1alpha2“ [Lin21].

Be abejo, „Linkerd“ siūlo „Flagger“ [Fla21] operatorių, kuris šiuo metu turėtų susitvarkyti su šia užduotimi, tačiau lyginamasis tyrimas šio funkcionalumo su skirtingais trečios šalies API nėra visiškai taisyklingas.

2.3.3.4. Srauto nukreipimo konfigūracijos testavimas

Bandymams buvo pasirinktas apkrovos scenarijus, kuomet vartotojų užklausų skaičius išlieka stabilus – 50 RPS ir trunka 180 sekundžių. Bandymų metu buvo siekiama išsiaiškinti kiek efektyviai kiekviena paslaugų tinklo platforma susitvarko su šia užduotimi palyginant kiek padidėja bendras atsako greitis įvedant papildomas konfigūracijas. Bandymai buvo atlikti 3 scenarijams kartojant kiekvieną 3 kartus siekiant gauti vidutines reikšmes:

- **Scen-TF1**: Atsitiktinis srauto nukreipimas turint standartinį apkrovos balansavimo servisą.
- **Scen-TF2**: Srauto nukreipimas remiantis nustatytais svoriais.
- **Scen-TF3**: Srauto nukreipimas remiantis HTTP užklausos antraštės šalies parametru.

Rezultatai

Teorinės dalies rezultatai

1. Atlikta paslaugų tinklo analizė - pristatyta architektūra, įvardintas veikimo principas bei palaikomos galimybės.
2. Išskirta mikroservisų problemų aibė bei pasiūlyti sprendimai naudojantis „Istio“ bei „Linkerd“ siūlomu funkcionalumu:
    - Didelio plečiamumo bei prieinamumo valdymas
    - Autentifikacijos bei autorizacijos taisyklės
    - Aplikacijų srauto valdymas
3. Palygintos paslaugų tinklo platformos išskirtų lyginimo kriterijų bei nagrinėjamo funkcionalumo atžvilgiu.

Praktinės dalies rezultatai

1. Sukurta aplikacija su konfigūracijos failais, kurie leidžia aktyvuoti darbe nagrinėtą „Istio“ bei „Linkerd“ paslaugų tinklo platformų funkcionalumą.
2. Sukurta analitinio „Grafana“ skydelio (angl. *dashboard*) konfigūracija, pritaikyta abejoms platformoms, kurioje yra vizualizuojamos aplikacijos našumo metrikos, panaudotos šiame darbe.
3. Aprašytų bandymo scenarijų (Scen-HPA1, Scen-HPA2, Scen-HPA3) automatinio plečiamumo rezultatai (1 priedas):

Scenarijus	Vidutinis replikų skaičius	Vidutinis P99 atsako greitis	Vidutinis CPU sunaudojimas	Vidutinis atminties sunaudojimas	Klaidų skaičius	Apdorotų užklausų skaičius
Scen-HPA1 (Istio)	1	241 ms	154 m	115 MiB	0.00%	45951
Scen-HPA1 (Linkerd)	1	219 ms	113 m	73 MiB	0.00%	44942
Scen-HPA2 (Istio)	3.03	119 ms	230 m	374 MiB	0.02%	45299
Scen-HPA2 (Linkerd)	3.1	93 ms	172 m	218 MiB	0.02%	44941

Scen-HPA3 (Istio)	4.8	110 ms	267 m	571 MiB	0.04%	45660
Scen-HPA3 (Linkerd)	5.4	115 ms	203 m	379 MiB	0.08%	45835

2 lentelė. Automatinio plečiamumo rezultatai

Gautų rezultatų analizė

Kalbant apie patikimiausią t.y. mažiausiai klaidų turintį sprendimą yra scenarijus Scen-HPA1 su nesikeičiančiu serviso replikų skaičiumi, kadangi pasitaikiusių klaidų skaičius tiek „Linkerd“, tiek „Istio“ atvejais yra lygus 0%. Todėl jeigu sunaudojamų resursų skaičius yra neribotas ir nėra poreikio taupyti, replikų skaičius gali būti padidintas iki tiek, kiek leidžia resursai ir tuomet užklausos bus apdorotos su mažiausia tikimybe klaidų. Tačiau, tokiu atveju sunaudojamų resursų skaičius padidėja N kartų, kur N replikų skaičius. Iš to seka, jog resursai bus iššvaistyti veltui žemos apkrovos atvejais.

Iš Scen-HPA2 rezultatų seka, jog testavimo metu „Linkerd“ platforma pasirodė našesne ir taupesne nei „Istio“. Tą parodo pasiektas atsako greitis abiem atvejais ir sunaudoti procesoriaus bei atminties resursai atitinkamai. Klaidų skaičius abiem atvejais išliko vienodas. Tačiau kalbant apie šį scenarijų, verta paminėti, jog bandymų metu buvo pastebėta, kad renkantis automatinio plečiamumo strategiją pagal RPS yra galimybė suklysti pasirenkant optimalią ribą, kuomet servisai bus išplėsti. Todėl pradžioje ribinės reikšmės nustatymas priminė spėliojimų žaidimą, o vėliau simuliacijos ir testavimo būdu vyksta optimalios reikšmės paieška.

Trečiame scenarijuje Scen-HPA3 „Istio“ pasirodė kiek geriau nei „Linkerd“ atsako greičio bei klaidų skaičiaus prasme. Tačiau resursų sunaudojime, kaip ir praeituose scenarijuose, pirmauja „Linkerd“. Bandymų metu buvo išsiaiškinta, jog nepaisant to, kad resursų yra sunaudojama kiek daugiau nei Scen-HPA2, optimalios reikšmės pasirinkimas yra paprastesnis, kadangi turint nefunkcinį reikalavimą, kad atsako greitis privalo būti apie X ms, HPA resursui galima nurodyti šią metrikos reikšmę pasitikint jo algoritmu.

Todėl neturint poreikio taupyti paprasčiausias variantas - išplėsti serviso replikas iki maksimumo, kas suteiktų daugiausiai stabilumo klaidų prasme, tačiau su tuo išlaikymo kaštai bus didžiausi. Siekiant optimaliai išnaudoti turimus resursus turint nefunkcinį atsako greičio reikalavimą, verta pasirinkti Scen-HPA3 automatinio plečiamumo strategiją. Turint daugiau laiko resursų testavimui ir optimalios RPS reikšmės parinkimui – verta pasirinkti Scen-HPA2 strategiją.

4. „Linkerd“ ir „Istio“ autentifikacijos bei autorizacijos taisyklių įgyvendinimo rezultatai:

Platforma	Palaiko autentifikacijos taisykles	Palaiko autorizacijos taisykles	Autentifikacijos taisyklės privalomos konfigūruojant mTLS	Autorizacijos taisyklės privalomos konfigūruojant mTLS
Istio	✓ (3 tipai)	✓	✓	✗
Linkerd	✓ (5 tipai)	✓	✓	✓

3 lentelė. Palaikomos galimybės saugumo kontekste

Gautų rezultatų analizė

Sprendžiant saugumo klausimą tiek „Istio“, tiek „Linkerd“ atveju mTLS funkcionalumas pasirodė veiksmingas. Tinkamai sukonfigūravus mTLS servisai klasterio viduje yra apsaugoti, kadangi užklausos yra šifruojamos, kas sumažina tarpininko (angl. *man-in-the-middle*) atakos tikimybę. Kilus poreikiui išjungti mTLS, „Istio“ siūlo „disable“ taisyklė, „Linkerd“ – „all-unauthenticated“. Norint leisti komunikaciją tarp servisų tiek užšifruotomis užklausomis, tiek paprasto teksto, „Istio“ leidžia įjungti „permissive“ scenarijų, „Linkerd“ tokios opcijos neturi.

Norint įvesti papildomus komunikacijos ribojimus autorizacijos taisyklių pagalba, abi platformos siūlo tam savo sprendimą. Verta paminėti, jog norint įjungti mTLS tarp servisų, „Linkerd“ reikalauja išreikštinai apibrėžti autorizacijos taisykles tarp servisų. Todėl konfigūracija užtruko kiek ilgiau nei „Istio“ atveju. Iš kitos pusės, „Linkerd“ turi daugiau galimybių autorizacijos taisyklių klausimu, pavyzdžiui, yra leidžiama sukonfigūruoti taisyklę ne tik serviso lygmenyje, bet ir porto.

5. Aprašytų bandymo scenarijų (Scen-TF1, Scen-TF2, Scen-TF3) srauto nukreipimo rezultatai (2 priedas):

Scenarijus	Vidutinis P99 atsako greitis	Vidutinis CPU sunaudojimas	Vidutinis atminties sunaudojimas	Klaidų skaičius	Apdorotų užklausų skaičius
Scen-TF1 (Istio)	249 ms	142 m	115 MiB	0.02%	8646
Scen-TF1 (Linkerd)	229 ms	117 m	61 MiB	0.03%	8781

Scen-TF2 (Istio)	306 ms	140 m	111 MiB	0.08%	8784
Scen-TF2 (Linkerd)	244 ms	124 m	61 MiB	0.06%	8860
Scen-TF3 (Istio)	356 ms	148 m	105 MiB	0.15%	8654
Scen-TF3 (Linkerd) nepalaikoma	-	-	-	-	-

4 lentelė. Srauto nukreipimo rezultatai

Gautų rezultatų analizė

Srauto nukreipimas naudojantis standartine „Kubernetes“ apkrovos balansavimo pagalba (Scen-TF1) dar kartą patvirtino teoriją, jog „Linkerd“ platforma yra taupesnė resursų atžvilgiu bei greitesnė, ką parodo atsako greičio rezultatai.

Norint sukonfigūruoti srauto nukreipimą pagal svorius (Scen-TF2), abi platformos puikiai susitvarko su šia užduotimi, kadangi norimi rezultatai buvo pasiekti. Šiame scenarijuje vienareikšmiškai patrauklesnis pasirodė „Linkerd“ savo greičiu, taupumu bei klaidų skaičiumi. Konfigūracija taip pat buvo paprastesnė, kadangi turint „Kubernetes“ servisus, tereikėjo pridėti dar vieną „TrafficSplit“ resursą ir užduotis buvo pasiekta. „Istio“ atveju, buvo reikalaujama pridėti dar vieną virtualų servisą, kas matomai paveikė našumą.

Kilus poreikiui nukreipti užklausas pagal HTTP užklausos antraštės parametrą (Scen-TF3), uždavinį galėjo išspręsti tik „Istio“. Konfigūracija buvo panaši į (Scen-TF2) apart to, kad teko pridėti papildomą išraišką, kuri galėjo ištraukti antraštę bei palyginti ją su siekiama reikšme. „Linkerd“ savo ruožtu nors ir turi paruošęs integraciją su „SMI“ API, šis funkcionalumas kol kas yra neprieinamas vartotojams, kadangi platforma dar nespėjo atsinaujinti į naujausią „SMI“ API versiją.

Praktinės dalies rezultatai pagal lyginimo kriterijus

Žemiau yra pateiktos lentelės (5 lentelė, 6 lentelė, 7 lentelė) su apibendrintais rezultatais pagal išskirtus lyginimo kriterijus.

Platforma	Vidutinis P99 atsako greitis	Vidutinis CPU sunaudojimas	Vidutinis atminties sunaudojimas	Klaidų skaičius
Istio	205 ms	187 m	257 MiB	0.032%

Linkerd	180 ms	146 m	158 MiB	0.038%

5 lentelė. Greitaveikos rezultatai

Gautų rezultatų analizė

Iš 5 lentelės rezultatų seka, jog „Linkerd“ yra greitesnė bei ženkliai taupesnė platforma nei „Istio“. Tačiau, nors ir su nedideliu skirtumu, „Istio“ yra patikimesnė klaidų atžvilgiu.

Platforma	Resursų skaičius	Konfigūracijos eilučių skaičius	Dokumentacijos aiškumas (1-3)	Apimtis (1-3)
Istio	19 (K8S: 13, CRD: 5, kiti: 1)	364	3	3
Linkerd	22 (K8S: 15, CRD: 6, kiti: 1)	438	2	2

6 lentelė. Įgyvendinimo sudėtingumo rezultatai

Gautų rezultatų analizė

Įgyvendinimo sudėtingumo aspekte (6 lentelė) „Istio“ reikalavo mažiau resursų ir konfigūracijos eilučių nei „Linkerd“, tačiau įgyvendinant nagrinėtą funkcionalumą buvo pastebėta, jog konfigūracijos apimtis laiko atžvilgiu buvo didesnė nei „Linkerd“. Dokumentacijos aiškumas „Istio“ atveju gavo didesnį įvertį nei „Linkerd“ dėl pateikiamos informacijos organizavimo.

Platforma	Stebimumo funkcionalumas	Saugumo funkcionalumas	Srauto valdymo funkcionalumas
Istio	• „Grafana“ • „Prometheus“ • „Kiali“ skydelis	• Autentifikacijos taisyklės • Autorizacijos taisyklės • mTLS	• Nukreipimas pagal svorius • Nukreipimas pagal HTTP antraštę
Linkerd	• „Grafana“ • „Prometheus“ • „Viz“ skydelis • „Buoyant“ skydelis	• Autentifikacijos taisyklės • Autorizacijos taisyklės • mTLS	• Nukreipimas pagal svorius

7 lentelė. Galimybių bei apribojimų rezultatai

Gautų rezultatų analizė

Stebimumo aspekte (7 lentelė), „Linkerd“ leidžia rinktis iš dviejų valdymų plokštumos skydelių, kurie pasirodė patogesni nei „Istio“. Saugumo aspekte, abi platformos siūlo tą patį funkcionalumą. Kalbant apie srauto valdymą, labiau išbaigta platforma yra „Istio“, kadangi „Linkerd“ šiuo metu neturi užklausų nukreipimą pagal HTTP antraštę.

Išvados

Atlikus darbą prieita prie šių išvadų:

1. Abi paslaugų tinklo platformos pasiekia panašius atsako greičio rezultatus, tačiau „Linkerd“ tyrimo metu pasirodė kiek greitesnė bei ženkliai taupesne nei „Istio“ sunaudojamų resursų atžvilgiu.
2. Konfigūruojant automatinį plečiamumą pagal atsako greičio metriką „Istio“ atveju klaidų kiekis buvo mažesnis negu „Linkerd“. Srauto valdyme įverčiai nors ir yra panašūs, paėmus vidurkį „Linkerd“ klaidų buvo mažiau. Pagal bendrus rezultatus „Istio“ platforma yra kiek patikimesnė klaidų atžvilgiu, tačiau skirtumas yra neženklus - 0.06% į „Istio“ naudą.
3. Įgyvendinant darbe nagrinėtą paslaugų tinklo platformų siūlomą funkcionalumą „Istio“ reikalavo mažiau konfigūracijos eilučių nei „Linkerd“, tačiau įgyvendinimo sudėtingumas buvo didesnis dėl sunaudoto laiko sprendžiant iškilusias problemas.
4. Abi paslaugų tinklo platformos leidžia greitai sukonfigūruoti „Prometheus“ bei „Grafana“ resursus paruoštų šablonų dėka, kas ženkliai supaprastina programuotojų darbą užtikrinant sistemos stebimumą. Kalbant apie paslaugų tinklo valdymo plokštumą, „Linkerd“ turėjo du skirtingus skydelius („viz“, „buoyant“), kurie pasirodė patogesni negu „Istio“ („kiali“) dėl intuityvesnio informacijos organizavimo būdo.
5. Srauto valdymo srityje „Istio“ yra labiau išbaigta platforma, kadangi siūlomas funkcionalumas nėra priklausomas nuo trečios šalies API‘ų, kaip tai yra „Linkerd“ atveju.
6. Paslaugų tinklai suteikia gana aukštą abstrakcijos lygį bei siūlo bendrą standartą konfigūruojant sistemos stebimumą, saugumą bei srauto valdymą, kas leidžia standartizuoti bendrą architektūrą atskiriant mikroservisų verslo logiką nuo tarpinio sluoksnio problemų.
7. Turint daugiau infrastruktūros resursų bei labiau patyrusių programuotojų „Istio“ gali būti tinkamesnis sprendimas nei „Linkerd“ dėl savo funkcionalumo išbaigtumo. Tačiau, „Linkerd“ – yra greitesnė, taupesnė bei lengviau konfigūruojama platforma, kas sutaupo nemažai laiko ir infrastruktūros resursų, įdiegiant ją į savo mikroservisų architektūros programų sistemą.

Literatūra

[AF19] J. Atelsek ir W. Fellows. The Role of Service Mesh as a Cloud-Native Enabler is Building Fast. Prieiga per: https://go.451research.com/2019-mi-service-mesh-cloud-native-enabler.html, 2019. Tikrinta 2022-05-21.

[Cal20] L. Calcote. *The Enterprise Path to Service Mesh Architectures, 2nd Edition*. O'Reilly Media, Inc., 2020.

[CB19] L. Calcote ir Z. Butcher. *Istio: Up and Running*. O'Reilly Media, Inc., 2019.

[DHB+21] M. Driss, D. Hasan, W. Boulila ir J. Ahmad. Microservices in IoT Security: Current Solutions, Research Challenges, and Future Directions. *Procedia Computer Science*:2385–2395, 2021. DOI: 10.1016/j.procs.2021.09.007.

[Doc22] Docker. Orientation and setup. Prieiga per: https://docs.docker.com/get-started/, 2022. Tikrinta 2022-05-21.

[Fla21] Flagger. Flagger: Progressive Delivery Operator for Kubernetes. Prieiga per: https://flagger.app/, 2021. Tikrinta 2022-05-21.

[Fou21] Apache Software Foundation. Apache JMeter. Prieiga per: https://jmeter.apache.org/, 2021. Tikrinta 2022-05-21.

[Fou22a] Cloud Native Computing Foundation. Kubernetes documentation. Prieiga per: https://kubernetes.io/docs/home/, 2022. Tikrinta 2022-05-21.

[Fou22b] Python Software Foundation. Python. Prieiga per: https://www.python.org/downloads/, 2022. Tikrinta 2022-05-21.

[Fro21] T. Fromm. Performance Benchmark Analysis of Istio and Linkerd. Prieiga per: https://kinvolk.io/blog/2019/05/performance-benchmark-analysis-of-istio-and-linkerd/, 2021. Tikrinta 2022-05-21.

[Gup21] P. Gupta. Mutual TLS: Securing Microservices in Service Mesh. Prieiga per: https://thenewstack.io/mutual-tls-microservices-encryption-for-service-mesh, 2021. Tikrinta 2022-05-21.

[JVR08] J.Durkovic, V. Vukovic ir L. Rakovic. Open Source Approach in Software Development - Advantages and Disadvantages. *International Scientific Journal of Management Information Systems*:29–33, 2008.

[KBB+21] A. Koschel, M. Bertram, R. Bischof, K. Schulze, M. Schaaf ir I. Astrova. A Look at Service Meshes. *2021 12th International Conference on Information, Intelligence, Systems Applications (IISA)*:1–8, 2021. DOI: 10.1109/IISA52424.2021.9555536.

[KKM+22] A. Kammer, C. Koppelt, J. Müller, H. Prinz, C Schmidt ir Eberhard Wolff. Service mesh comparison. Prieiga per: https://servicemesh.es/, 2022. Tikrinta 2022-05-21.

[KMR+19] P. Kumar, A. Moubayed, A. Refaey, A. Shami ir J. Koilpillai. Performance Analysis of SDP For Secure Internal Enterprises. *2019 IEEE Wireless Communications and Networking Conference (WCNC)*:1–6, 2019. DOI: 10.1109/WCNC.2019.8885784.

[Krz21] P. Krzyzanowski. Remote Procedure Calls. Prieiga per: https://people.cs.rutgers.edu/~pxk/417/notes/rpc.html, 2021. Tikrinta 2022-05-21.

[LAX+20] N. Lazarev, N. Adit, S. Xiang, Z. Zhang ir C. Delimitrou. Dagger: Towards Efficient RPCs in Cloud Microservices With Near-Memory Reconfigurable NICs. *IEEE Computer Architecture Letters*:134–138, 2020. DOI: 10.1109/LCA.2020.3020064.

[Lin21] Linkerd. Linkerd-SMI. Prieiga per: https://github.com/linkerd/linkerd-smi/releases/tag/v0.2.0, 2021. Tikrinta 2022-05-21.

[LLG+19] W. Li, Y. Lemieux, J. Gao, Z. Zhao ir Y. Han. Service Mesh: Challenges, State of the Art, and Future Research Opportunities. *2019 IEEE International Conference on Service-Oriented System Engineering (SOSE)*:122–1225, 2019. DOI: 10.1109/SOSE.2019.00026.

[Man19] S. Manpathak. Kubernetes Service Mesh: A Comparison of Istio, Linkerd, and Consul. Prieiga per: https://platform9.com/blog/kubernetes-service-mesh-a-comparison-of-istio-linkerd-and-consul/, 2019. Tikrinta 2022-05-21.

[Mar03] Robert C. Martin. *Agile Software Development, Principles, Patterns, and Practices*. Prentice Hall PTR, 2003.

[Mic22a] Microsoft. C documentation. Prieiga per: https://docs.microsoft.com/en-us/dotnet/csharp/, 2022. Tikrinta 2022-05-21.

[Mic22b] Microsoft. Download .NET Core 3.1. Prieiga per: https://dotnet.microsoft.com/en-us/download/dotnet/3.1, 2022. Tikrinta 2022-05-21.

[Mic22c] Microsoft. Download Visual Studio Code. Prieiga per: https://code.visualstudio.com/Download, 2022. Tikrinta 2022-05-21.

[Miy21] C. Miyachi. The Rise of Kubernetes. *2021 Cloud Continuum*:1–5, 2021. DOI: 10.1109/CloudContinuum54760.2021.00002.

[Min22] Minikube. Documentation. Prieiga per: https://minikube.sigs.k8s.io/docs/start/, 2022. Tikrinta 2022-05-21.

[Mor17] W. Morgan. What's service mesh? And why do I need one? Prieiga per: https://linkerd.io/2017/04/25/whats-a-service-mesh-and-why-do-i-need-one, 2017. Tikrinta 2022-05-21.

[Mor21] W. Morgan. Benchmarking Linkerd and Istio. Prieiga per: https://linkerd.io/2021/05/27/linkerd-vs-istio-benchmarks/, 2021. Tikrinta 2022-05-21.

[MZ19] A. El Malki ir A. Zdun. Guiding Architectural Decision Making on Service Mesh Based Microservice Architectures. *Bures, T., Duchien, L., Inverardi, P. (eds) Software Architecture. ECSA 2019. Lecture Notes in Computer Science(), vol 11681*:3–19, 2019. DOI: 10.1007/978-3-030-29983-5_1.

[OKe19] M. O'Keefe. The Service Mesh Era: Architecting, Securing and Managing Microservices with Istio. Prieiga per: https://cloud.google.com/blog/products/networking/welcome-to-the-service-mesh-era-introducing-a-new-istio-blog-post-series, 2019. Tikrinta 2022-05-21.

[Pat21] S. Patil. Benchmarking Istio, Consul, and Linkerd. Prieiga per: https://cldcvr.com/news-and-media/blog/benchmarking-istio-consul-and-linkerd/, 2021. Tikrinta 2022-05-21.

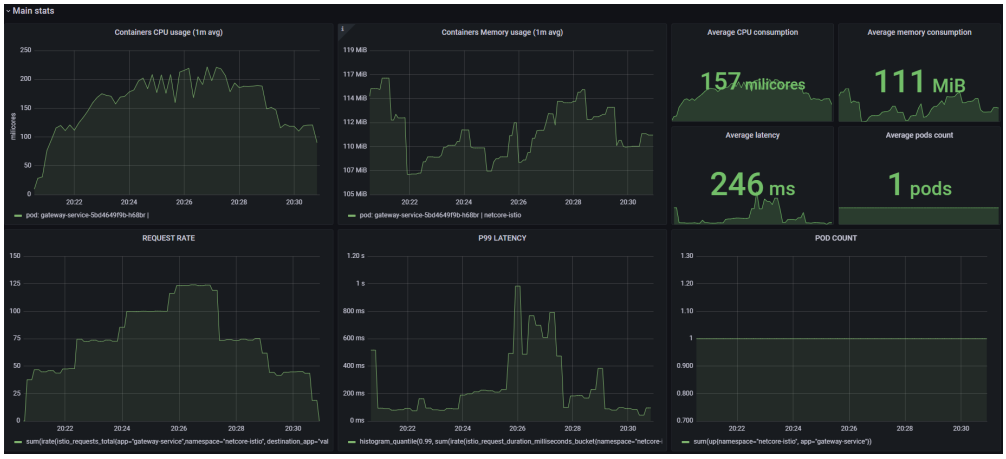
[Por21] Portshift. *The DevOps Guide Essential to Service Meshes*. Portshift, 2021.

[SB19] L. Sun ir D. Berg. *Istio Explained*. O'Reilly Media, Inc., 2019.

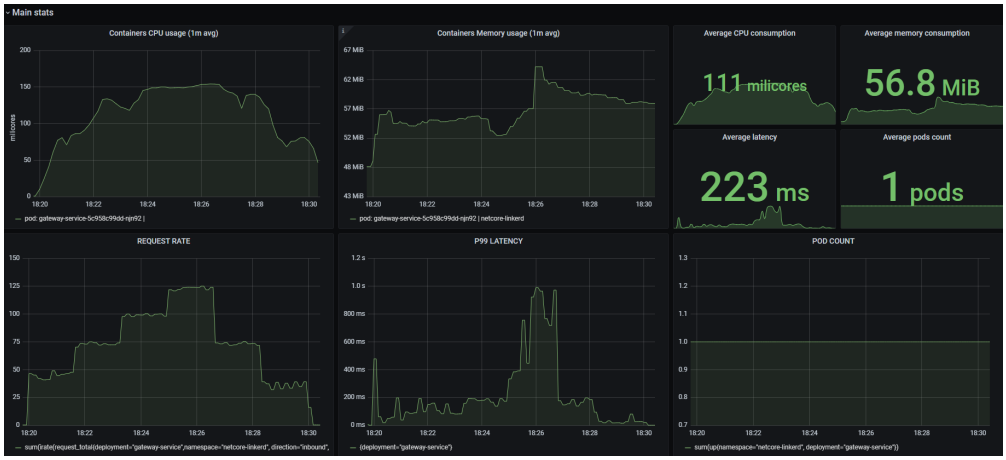
[Tiw17] A. Tiwari. A sidecar for your service mesh. Prieiga per: https://www.abhishek-tiwari.com/a-sidecar-for-your-service-mesh/, 2017. Tikrinta 2022-05-21.

Priedas nr. 1

Automatinio plečiamumo rezultatai

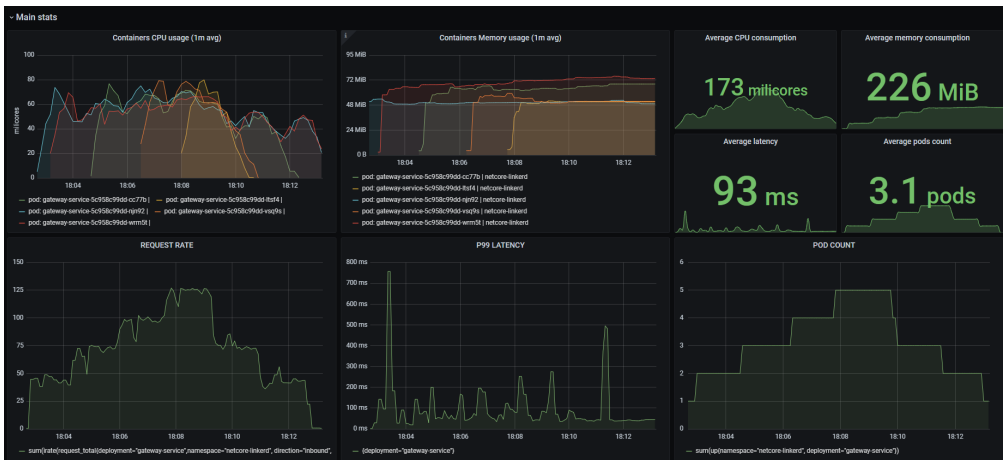


15 pav. Vienas iš Scen-HPA1 (Istio) bandymų

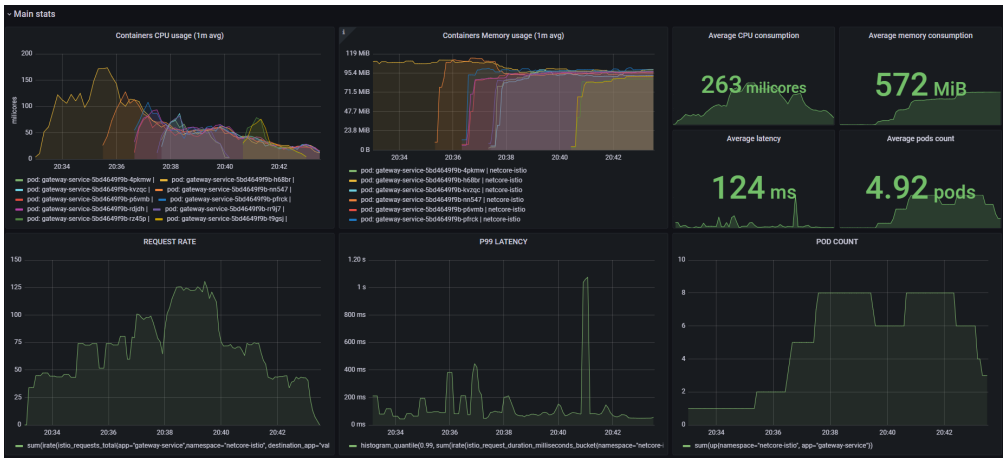


16 pav. Vienas iš Scen-HPA1 (Linkerd) bandymų

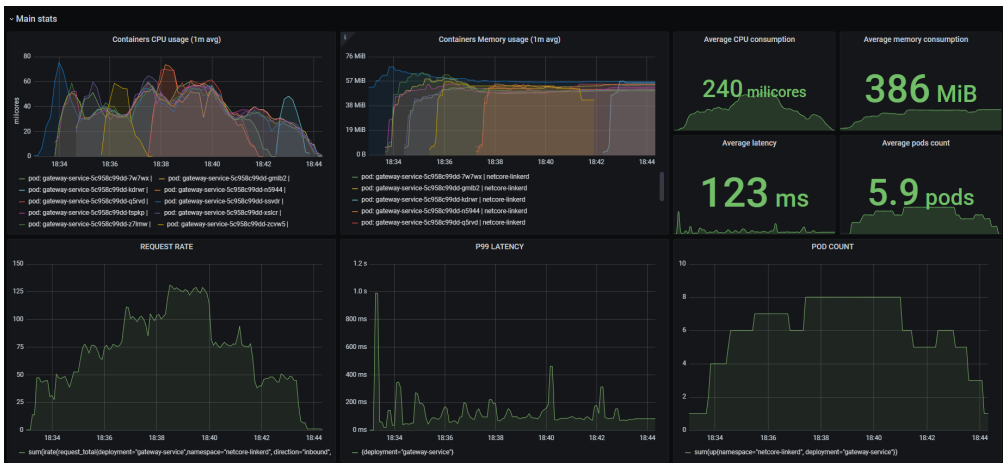
17 pav. Vienas iš Scen-HPA2 (Istio) bandymų



18 pav. Vienas iš Scen-HPA2 (Linkerd) bandymų



19 pav. Vienas iš Scen-HPA3 (Istio) bandymų



20 pav. Vienas iš Scen-HPA3 (Linkerd) bandymų

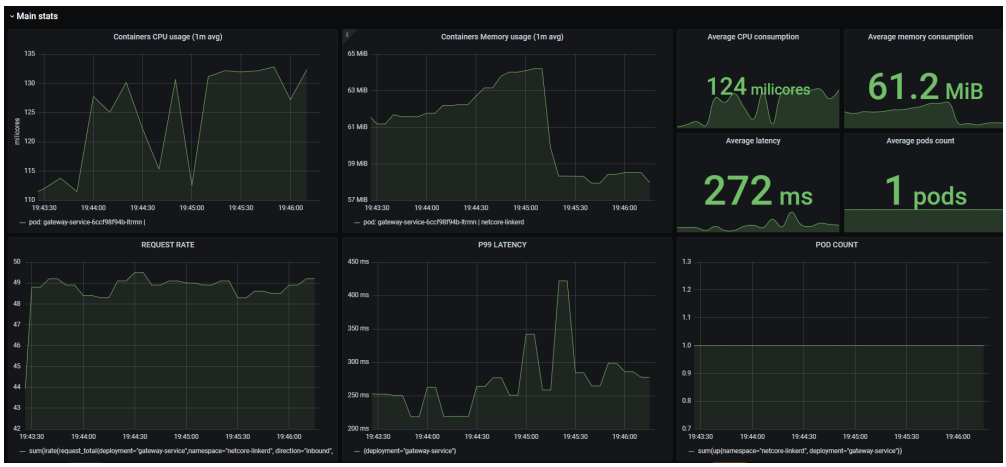
Priedas nr. 2

Srauto nukreipimo rezultatai

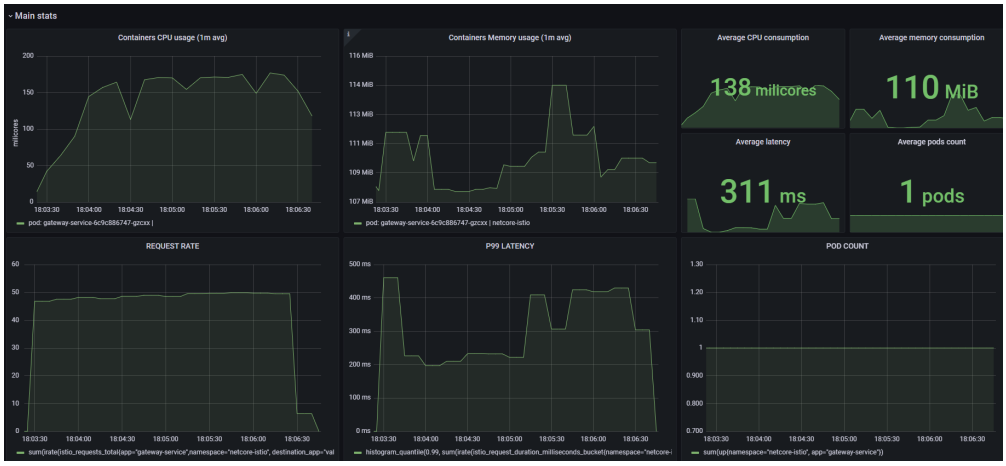
21 pav. Vienas iš Scen-TF1 (Istio) bandymų

22 pav. Vienas iš Scen-TF1 (Linkerd) bandymų

23 pav. Vienas iš Scen-TF2 (Istio) bandymų



24 pav. Vienas iš Scen-TF2 (Linkerd) bandymų



25 pav. Vienas iš Scen-TF3 (Istio) bandymų

Priedas nr. 3
Nuorodą į išeities tekstus

Darbe įgyvendinto prototipo išeities tekstus bei konfigūracijos failus galima rasti čia: `https://github.com/avizen-j/service-mesh-in-microservice-architecture`