Vpeljava sistema za politiko dostopa v avtorizacijski proces obstoječega sistema

Klemen Drev, Mitja Krajnc, Boris Ovčjak

Databox, Ptuj, Slovenija
klemen.drev@databox.com, mitja.krajnc@databox.com, boris.ovcjak@databox.com

V našem podjetju smo se soočali z izzivi pri avtorizaciji poslovne logike, kar je vplivalo na učinkovitost in varnost naših storitev. Da bi izboljšali ta proces, smo raziskali različne pristope in orodja za avtorizacijo ter se osredotočili na enostavnost in hitrost vpeljave. Odločili smo se za rešitev z uporabo Open Policy Agent (OPA), ki jo bomo predstavili v članku. OPA je odprtokodno orodje za enotno politiko avtorizacije, ki omogoča centralizirano izvajanje politik. Politike dostopa v OPA so napisane v jeziku Rego, kar omogoča fleksibilno pisanje pravil za avtorizacijo. Predstavili bomo tudi OPAL (OPA Policy Administration Layer), ki omogoča dinamično posodabljanje politik dostopa brez prekinitev delovanja storitev. OPA v naši infrastrukturi deluje kot Policy Decision Point (PDP), odgovoren za sprejemanje odločitev glede avtorizacije. Uporabljamo decentraliziran pristop, kjer je PDP implementiran kot "sidecar" ob posameznih storitvah, kar zagotavlja hitrejše odzivne čase in boljšo porazdelitev obremenitve. V članku bomo predstavili tudi spremembe in izzive pri razvoju sistema, saj je ločeno pisanje avtorizacijskih politik sprva povzročalo nekaj preglavic, predvsem v obliki pozabljenih politik dostopa in zavrnjenih zahtevkov.

Ključne besede:
Avtorizacija
Open Policy Agent
OPA
Rego
OPAL Server

Univerzitetna založba Univerze v Mariboru
ISBN 978-961-286-894-9
DOI https://doi.org/10.18690/um.feri.4.2024.15

1 Uvod

V hitro razvijajočem se svetu digitalne varnosti je avtorizacija postala ključen, a hkrati zahteven del za organizacije, ki si prizadevajo zaščititi svoje podatke in sisteme. Za razliko od avtentikacije, ki zgolj preveri identiteto uporabnika, avtorizacija določa, kaj je avtoriziran uporabnik dejansko upravičen storiti, kar predstavlja bolj zapleten proces, ki je tesno povezan s poslovno logiko. Zapletenost upravljanja dovoljenj čez različne aplikacije in storitve lahko privede do znatnih varnostnih ranljivosti in operativnih neučinkovitosti.

V podjetju smo se odločili za izboljšavo obstoječega procesa, zato smo izvedli raziskavo o avtorizaciji in naleteli na številne skupne težave, s katerimi se soočajo organizacije. Pogosto smo se soočali s prepletanjem politik, ki so bile težko obvladljive in so zahtevale veliko časa za vzdrževanje. Prav tako je bila problematična heterogenost sistemov, ki so potrebovali različne pristope k avtorizaciji, kar je dodatno zapletlo že tako kompleksno področje.

Želeli smo rešitev, ki bi bila hitra, učinkovita in enostavna za vzdrževanje. Potrebovali smo sistem, ki bi omogočal centralizirano upravljanje politik, a hkrati dovoljeval dovolj fleksibilnosti za prilagajanje specifičnim potrebam posameznih aplikacij in storitev. Naša raziskava nas je pripeljala do inovativnih rešitev, kot je Open Policy Agent (OPA).

OPA je odprtokodni, splošno namenski sistem za upravljanje politik, ki ločuje sprejemanje odločitev o politiki od aplikacijske kode, kar zagotavlja bolj prilagodljiv in razširljiv pristop k avtorizaciji. S pomočjo OPA lahko podjetja uveljavljajo natančne kontrole dostopa, poenostavijo upravljanje politik in izboljšajo svojo splošno varnostno držo. Poleg tega je OPA zasnovan tako, da omogoča hitro implementacijo in enostavno vzdrževanje, kar je bilo ključno za naše potrebe.

Raziskave so pokazale, da OPA omogoča centralizirano upravljanje politik, kar poenostavlja njihovo implementacijo in vzdrževanje [1]. Poleg tega so odprtokodne specifikacije OPA dostopne na GitHub [2], kjer je mogoče najti podrobne informacije o njegovi uporabi in implementaciji.

V tem članku bomo raziskali te težave z avtorizacijo in razložili, kako OPA revolucionira način, kako podjetja obvladujejo nadzor dostopa v sodobnih, oblačno orientiranih okoljih.

2 Open Policy Agent

Open Policy Agent (OPA) omogoča bolj prilagodljiv in razširljiv pristop k avtorizaciji, saj lahko politike definirate in posodabljate neodvisno od aplikacij. OPA omogoča centralizirano upravljanje politik, kar olajša njihovo implementacijo in vzdrževanje. Politike dostopa se znotraj opa definirajo s posebnih jezikom Rego, ki ga bomo predstavili kasneje.

OPA deluje tako, da sprejema zahteve za avtorizacijo od aplikacij in na podlagi definiranih politik vrne odločitve. Politike so zapisane v jeziku Rego, ki omogoča enostavno in razumljivo definiranje kompleksnih pravil. OPA je zasnovan tako, da se lahko integrira z različnimi sistemi in platformami, kar omogoča njegovo uporabo v različnih okoljih, od mikro storitev do monolitnih aplikacij.

2.1. Jezik Rego

Rego jezik je osrednji del Open Policy Agent (OPA) [3], ki omogoča pisanje politik na jasen in strukturiran način. Rego je bil zasnovan z mislijo na fleksibilnost in ekspresivnost, kar omogoča definiranje kompleksnih pravil za avtorizacijo, dostop in druge politike, ki temeljijo na podatkih.

Osnovne Lastnosti Rego Jezika

1. **Deklarativni Pristop**: Rego je deklarativni jezik, kar pomeni, da definirate, kaj želite doseči, ne da bi določali, kako naj se to izvede. Ta pristop poenostavlja pisanje in vzdrževanje politik, saj se osredotočate na želeni rezultat.
2. **Zmogljivost in Fleksibilnost**: Rego omogoča pisanje zelo zmogljivih politik, ki lahko vključujejo kompleksno logiko in več nivojske pogoje. Jezik podpira različne podatkovne tipe, vključno s seznami, nizi, slovarji in množicami, kar omogoča širok spekter uporab.
3. **Vgrajene Funkcije**: Rego vključuje številne vgrajene funkcije za manipulacijo podatkov, kot so filtriranje, združevanje in transformacija podatkov. To omogoča, da politike obdelujejo in analizirajo podatke na različne načine, kar povečuje njihovo uporabnost.
4. **Enostavno Branje in Pisanje**: Sintaksa Rego jezika je zasnovana tako, da je enostavna za branje in pisanje. To omogoča hitrejše pisanje politik in lažje razumevanje obstoječih pravil, kar zmanjšuje možnosti za napake.

Primer Rego Politike

Na spodnjem izseku kode je preprost primer Rego politike, ki preverja, ali ima uporabnik dovoljenje za dostop do določenega vira:

```
package profile.authz
default allow = false
allow {
    input.user == "admin"
}
allow {
    input.user == "user"
    input.action == "read"
}
```

V zgornjem primeru politika določa, da ima dostop uporabnik "admin" za vse akcije, medtem ko ima uporabnik "user" dovoljenje le za branje. Ta politika je preprosta, a prikazuje, kako lahko z Rego jezikom definirate jasna in razumljiva pravila.

Integracija z OPA

Politike napisane z Rego so shranjeni v OPA in se uporabljajo za odločanje v realnem času. Ko aplikacija pošlje zahtevo za avtorizacijo, OPA uporabi Rego politiko za obdelavo zahteve in vrne odločitev. Ta pristop omogoča centralizirano in dosledno upravljanje politik po celotni infrastrukturi.

Rego jezik omogoča organizacijam, da hitro in učinkovito razvijajo ter uvajajo politike, ki so ključne za varnost in skladnost njihovih sistemov. Zahvaljujoč svoji fleksibilnosti in zmogljivosti je Rego postal pomembno orodje za upravljanje politik v sodobnih IT okoljih.

2.2. Podatki

Upravljanje politik z OPA pogosto zahteva ustrezne kontekstualne podatke – informacije o uporabniku, viru, do katerega poskuša dostopati, itd. Brez teh informacij OPA ne bo mogla sprejeti pravih odločitev, ko gre za odločanje o politikah.

Na primer – pravilnik, ki pravi, da »Do te funkcije lahko dostopajo samo uporabniki, ki plačujejo storitev«, zahteva, da ima OPA informacije o vseh uporabnikih sistema in kateri od njih imajo poravnane obveznosti.

Bistveno vprašanje je - kako lahko te podatke prenesemo v OPA in kateri način je najučinkovitejši za to?

Načini zagotavljanja podatkov do politik:

- **Podatki so del veljavnih JWT (JSON Web Token).** To je enostavna in dobro poznana tehnologija, ki je del avtentikacijske plasti. Slabost tega načina je, da ima JWT omejitev velikosti - vsega ni mogoče dekodirati v JWT. Slabost je tudi, da za posodobitev podatkov je potrebno osvežiti žeton oz. ponovna prijava.
- **Predložitev podatkov ob poizvedbi politiki.** Tukaj kršimo najbolj osnovno dobro prakso grajenja avtorizacije - ločevanje politik od izvorne kode.
- **Zbiranje podatkov z uporabo paketov.** To nam omogoča pridobivanje veliko količino podatkov iz centralnega strežnika. Problem nastane ko se nam podatki hitro spreminjajo in moramo zaradi malih sprememb posodobiti celotne pakete.
- **Potiskanje podatkov v OPA z uporabo API (Application Programming Interface)** Podobno kot zbiranje podatkov z uporabo paketov, vendar lahko optimiziramo latenco posodabljanja in imamo veliko možnosti kako pripeljati podatke do politik. Slabost razvoj in je vzdrževanje takšnega sistema, ki skrbi za pravilno posodabljanje podatkov.
- **Pridobivanje podatkov ob evaluaciji politik.** OPA znotraj jezika Rego nam omogoča klicanje eksternih storitev http.send(), kar je solidna možnost pri ogromni količini podatkov. Slabost pa je, da vsako poizvedbo oziroma odločitev dodamo neko latenco omrežja.
- **OPAL (Open Policy Administration Layer).** Omogoča sinhronizacijo podatkov in politik ter odpravi slabosti načina s potiskanjem podatkov preko OPA API. Slabost tega načina je da je relativno nova tehnologija in je potreben določen čas učenja in vpeljave v obstoječ sistem.

3 OPAL Strežnik

OPAL (Open Policy Administration Layer) [4] strežnik deluje kot razširitev za OPA, ki avtomatizira distribucijo in sinhronizacijo politik ter podatkovnih posodobitev. OPAL omogoča, da OPA vedno deluje s posodobljenimi politikami in podatki, kar je ključno za zagotavljanje natančnih in aktualnih avtorizacijskih odločitev.

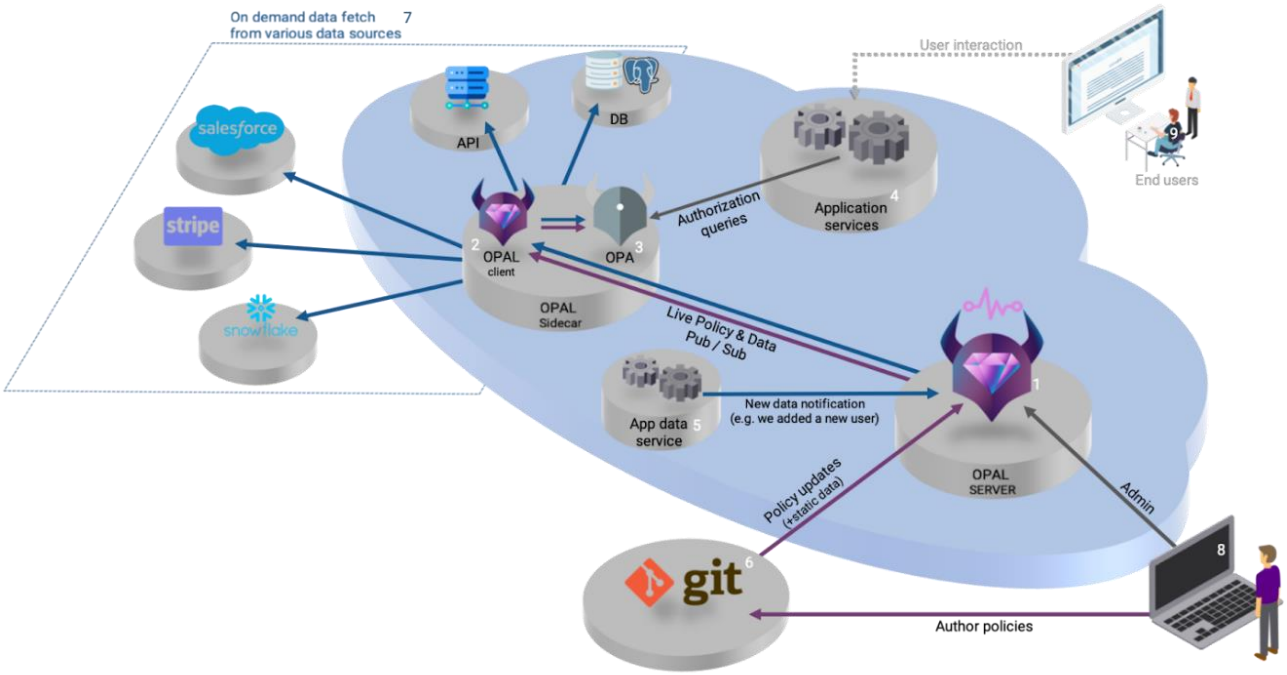
OPAL deluje tako, da spremlja spremembe v virih politik ter podatkih (npr. Git repozitorijih, bazah podatkov) in sproži posodobitve v OPA instancah. To zagotavlja, da so vse OPA instance sinhronizirane z najnovejšimi politikami, brez potrebe po ročnem posodabljanju.

3.1. Arhitektura rešitve z OPAL

OPAL vsebuje dve ključni komponenti, ki delujeta skupaj in sta predstavljeni na spodnji sliki. To sta OPAL strežnik in OPAL odjemalec, ki sta skupaj s preostalimi komponentami vrisani v komponenti diagram arhitekturne rešitve na naslednji sliki (slika 1). Zaradi pomembnosti sta tudi ti komponenti predstavljeni najvidneje. Celotna komunikacija je predstavljena s dvema barvama in sicer predstavljajo modre črte podatkovne tokove, medtem ko so vijolične črte poteki tokov politik.

- **OPAL Strežnik**
  - Ustvarja komunikacijske kanale do odjemalcev.
  - Spremlja spremembe avtorizacijskih politik na Git repozitorijih in potiska posodobitve odjemalcem.
  - Podatke posodablja kot razlike, ne v celoti.

- **OPAL Odjemalec**
  - Leži poleg OPA in skrbi za ažurnost politik ter podatkov.
  - Pridobiva podatke iz različnih virov.
  - Prenaša politike od strežnika.



Slika 1: Komponentni diagram arhitekture OPAL, pri čemer modre črte predstavljajo podatkovne tokove, vijolične črte pa predstavljajo toke politik. [5]

3.2. Možne Implementacije

1. OP **Centralizirana Implementacija**: OPA je lahko nameščen kot centraliziran strežnik, ki sprejema zahteve od več aplikacij. Ta pristop je primeren za manjše sisteme ali sisteme, kjer je latenca manj kritična. Centralizirana implementacija omogoča enostavno upravljanje in distribucijo politik, vendar lahko predstavlja ozko grlo pri visokih obremenitvah.
2. **Vgrajena Implementacija**: OPA se lahko vgradi neposredno v aplikacijsko kodo, kar omogoča tesno integracijo in zmanjšuje potrebo po dodatni infrastrukturi. Vgrajena implementacija je še posebej uporabna v monolitnih aplikacijah, kjer je mogoče hitro in enostavno dodati avtorizacijske kontrole neposredno v obstoječo kodo.
3. **Kubernetes Admission Controller**: V Kubernetes okolju se OPA lahko uporablja kot Admission Controller, ki preverja in uveljavlja politike ob ustvarjanju ali posodabljanju Kubernetes virov. Ta pristop omogoča centralizirano upravljanje politik na ravni celotnega Kubernetes grozda, kar zagotavlja skladnost in varnost na vseh ravneh infrastrukture.

3.3. Implementacija kot Sidecar

V našem podjetju smo se odločili za implementacijo OPA kot sidecar v naši mikrostoritveni arhitekturi. Sidecar vzorec pomeni, da vsaka mikro storitev teče skupaj z OPA instanco, kar omogoča lokalno in hitro odločanje o avtorizaciji.

Ta pristop ima več prednosti:

1. **Izolacija**: Vsaka storitev ima svojo instanco OPA, kar zmanjšuje tveganje napak in povečuje varnost.
2. **Zmanjšana latenca**: Lokalno odločanje pomeni hitrejši odzivni čas, saj ni potrebe po komunikaciji z oddaljenim strežnikom za vsako odločitev.
3. **Enostavna integracija**: Sidecar arhitektura omogoča enostavno integracijo OPA z obstoječimi storitvami brez večjih sprememb v kodi.

Sidecar arhitektura se je izkazala za najboljšo izbiro za naše potrebe zaradi svoje fleksibilnosti, zmogljivosti in enostavnosti integracije. Omogoča nam, da hitro in učinkovito uvajamo in vzdržujemo politike, ki so ključne za varnost in skladnost naših aplikacij.

4 Vpeljava OPAL v obstoječi sistem

Vpeljava takšne arhitekture v obstoječi sistem, je pomenila konkreten premislek glede načina razvoja, glede arhitekture in analize kakšen bo vpliv na obstoječe sisteme. Zato smo morali izvesti konkretno analizo trenutna stanja, izdelati načrt vpeljave in nato tudi poučiti razvijalce kako se bo spremenil način dela.

4.1. Analiza trenutnega stanja

Pred vpeljavo novega sistema za politiko dostopa smo izvedli temeljito analizo trenutnega stanja našega sistema avtorizacije. Ugotovili smo naslednje pomanjkljivosti:

- Heterogenost sistemov: Različni sistemi in aplikacije so uporabljali različne metode za upravljanje avtorizacije, kar je povzročalo nedoslednosti in povečano kompleksnost.
- Težavnost vzdrževanja: Politike dostopa so bile razpršene in težko obvladljive, kar je zahtevalo veliko časa in virov za vzdrževanje.
- Varnostne ranljivosti: Nepooblaščeni dostop je bil možen zaradi neučinkovitih in zastarelih avtorizacijskih politik.
- Vpeljava novega sistema za avtorizacijo bo v nekatere storitve težavna in bo potrebno izvesti postopoma v novejše in kritične sisteme prvo in potem na manj pomembne storitve

4.2. Načrt vpeljave

Da bi rešili zgoraj omenjene težave, smo oblikovali celovit načrt za vpeljavo OPA. Naš načrt je vključeval naslednje korake:

- Izbira ustrezne arhitekture: Odločili smo se za implementacijo OPA kot sidecar v naši mikro storitveni arhitekturi. Ta pristop omogoča lokalno in hitro odločanje o avtorizaciji, kar zmanjšuje latenco in povečuje varnost.

- Vzpostavitev OPAL strežnika: Za avtomatizacijo distribucije in sinhronizacijo politik ter podatkovnih posodobitev smo vzpostavili OPAL strežnik. To je omogočilo, da so vse instance OPA vedno delovale s posodobljenimi politikami in podatki.
- Razvoj in testiranje politik: Politike dostopa smo razvili in testirali s pomočjo Rego jezika. Ustvarili smo skladišče politik v Git repozitoriju, ki so bile centralizirano upravljane in enostavno dostopne za vse naše aplikacije.
- Integracija s sistemom: Implementirali smo OPAL odjemalca (OPA sidecar) poleg vsake mikro storitve in zagotovili, da vse storitve uporabljajo enotno politiko dostopa.

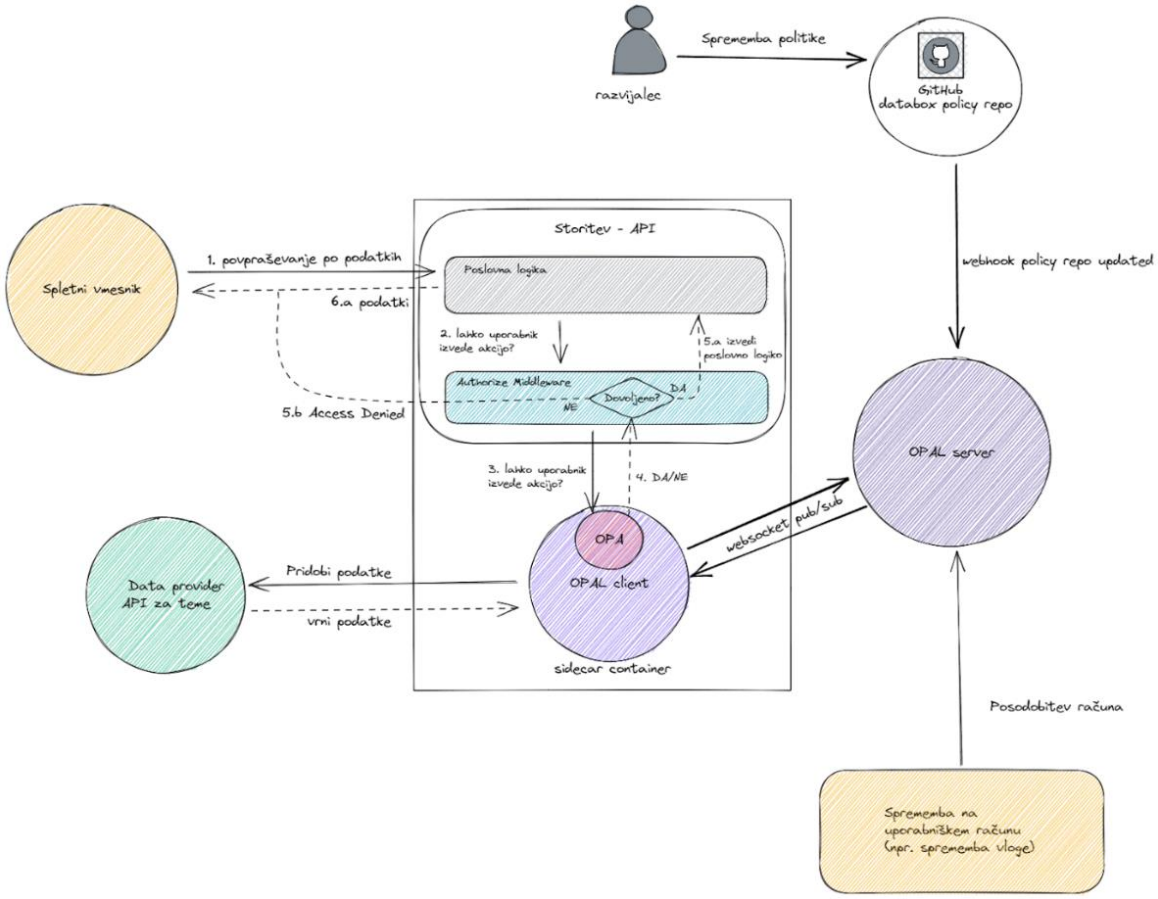
4.3. Implementacija

Vzpostavitev OPAL strežnika in Git repozitorija

Vzpostavili smo OPAL strežnik, ki omogoča sinhronizacijo politik in podatkov. Git repozitorij smo uporabili kot centralizirano mesto za shranjevanje in upravljanje politik. OPAL strežnik spremlja spremembe v Git repozitoriju in avtomatsko posodablja politike na vseh OPA instancah.

Definicija politik z Rego jezikom

Za vsako mikro storitev smo definirali specifične politike v jeziku Rego, ki so ustrezale njenim potrebam.



Slika 2: Diagram komunikacije med komponentami, vključno s posodobitvijo politik ali spremembi na uporabniškem računu.

Integracija OPAL odjemalca

Poleg mikro storitev, kod sidecar smo dodal OPAL odjemalec, ki vsebuje OPA in skrbi za:

- Decentralizirano točko odločitve: Vsaka mikro storitev sprejema avtorizacijske odločitve lokalno, kar zmanjšuje latenco in izboljšuje odzivni čas.
- Ažurnost politik in podatkov.

Spremljanje sprememb podatkov

Mikro storitve, ki so lastniki podatkov (vir resnice) skrbijo, da javljajo o spremembi podatkov OPAL strežniku, ta pa potem poskrbi, da sporoči svojim odjemalcem o spremembi. Odjemalci so naročeni na določene skupine podatkov, tako da lahko pri sebi držijo različne podatke, odvisno kaj potrebujemo za validacijo politike. Diagram komunikacije ja predstavljen na naslednji sliki (slika 2).

5 Zaključek

Vpeljava sistema za politike dostopa je bil eden izmed ključnih varnostnih projektov, ki smo jih naslovili v našem podjetju. S tem smo izboljšali varnost, saj je centralizirano upravljanje politik je omogočilo boljšo skladnost in zmanjšalo možnost napak. Na ta način smo lahko vpeljali tudi "zero trust policy", pri čemer če ni zapisane politike dostopa pomeni, da dostopa ni. Tekom razvoja se je to večkrat pokazalo kot pomemben korak, saj so razvijalci velikokrat na te korake pozabili. Seveda se je potem tekom testiranja hitro ugotovilo, da neka vloga nima pravic, ker politike niso zapisane s čimer smo takoj preprečili uhajanje nezavarovanih storitev na produkcijsko okolje.

Z vpeljavo smo povečali tudi učinkovitost tako iz vidika pisanja politik kot iz vidika izvajanja. Lokalno odločanje je izboljšalo odzivni čas in zmanjšalo latenco. Ker so vse politike zapisane v centralen Git repozitorij, so postale tudi zdaj enostavno dostopne, s čimer se je olajšalo upravljane, ki posledično poenostavlja vzdrževanje in posodobitve. Enostavnost jezika Rego je vplivala tudi na razumevanje politik drugih udeležencev v procesu, ne samo razvijalcev. Uvedli smo tudi testiranje politik, se pravi da se politike, ko se zapišejo tudi pretestirajo, da nas ne čaka kakšno presenečenje na produkcijskem okolju.

S sidecar arhitekturo smo omogočili tudi enostavno prilagajanje in razširitev sistema glede na specifične potrebe posameznih mikro storitev. S tem je bila vpeljava OPA in OPAL bila ključna za izboljšanje našega sistema avtorizacije, kar nam je omogočilo bolj varno, učinkovito in enostavno upravljanje dostopa v naši mikro storitveni arhitekturi.

Literatura

[1] www.openpolicyagent.org, OPA - Policy-based control for cloud native environments, obiskano 20. 7. 2024

[2] github.com/open-policy-agent/opa, Github OPA Repository, obiskano 20. 7. 2024

[3] www.openpolicyagent.org/docs/latest/policy-language/, Rego Policy Language, obiskano 21 .7. 2024

[4] docs.opal.ac/, Welcome to OPAL, obiskano 22. 7. 2024

[5] docs.opal.ac/overview/architecture, OPAL Architevture, obiskano 22.7. 2024