Univerza v Ljubljani
Fakulteta za računalništvo in informatiko
Fakulteta za matematiko in fiziko

Jakob Hostnik

Povezovanje gruč Kubernetes

diplomsko delo

interdisciplinarni univerzitetni
študijski program prve stopnje
računalništvo in matematika

Mentor: izr. prof. dr. Mojca Ciglarič
Somentor: asist. dr. Matjaž Pančur

Ljubljana, 2021

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

TODO Besedilo teme diplomskega dela študent prepiše iz študijskega informacijskega sistema, kamor ga je vnesel mentor. V nekaj stavkih bo opisal, kaj pričakuje od kandidatovega diplomskega dela. Kaj so cilji, kakšne metode uporabiti, morda bo zapisal tudi ključno literaturo.

Na tem mestu bi se zahvalil mentorici izr. prof. dr. Mojci Ciglarič za pripravljenost in mentorstvo. Zahvalil bi se tudi somentorju asist. dr. Matjažu Pančurju za vse nasvete in pomoč pri pisanju diplomske naloge. Zahvala pa gre tudi moji ženi, staršem, bratom, sestram in prijateljem za podporo in spodbudo pri študiju.

Mami Lučki.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
CRD	custom resource definition	definicija tipov po meri
DNS	domain name system	sistem domenskih imen
IP	internet protocol	internetni protokol
HA	high availability	visoka razpoložljivost
GA	general availability	splošna dostopnost
VPN	virtual private network	navidezno zasebno omrežje
TOSCA	topology and orchestration specification for cloud applications	specifikacija topologije in orkestracije za aplikacije v oblaku
WAN	wide area network	prostrano omrežje

Povzetek

Naslov: Povezovanje gruč Kubernetes

Avtor: Jakob Hostnik

Ko na naših strežnikih začne zmanjkovati virov obstajata dva standardna načina za povečanje virov v našem sistemu. Prva možnost je, da nadgradimo naše strežnike, druga pa, da jih kupimo več in jih povežemo v računalniško gručo. V zadnjih letih se je na slednjem področju zgodil preboj s pojavom sistema Kubernetes. Sistem je zaradi svoje popularnosti postal de facto standard za upravljanje gruč in orkestracijo kontejnerjev. A ena sama gruča ni vedno dovolj v primerih, ko imamo težave z dragim prenosom podatkov, preveliko latenco do naših uporabnikov ali pa želimo še bolj povečati stabilnost ali varnost našega sistema. Pogledali si bomo ozadje povezovanja gruč in kakšne pristope lahko uporabimo za reševanje naših problemov. Poseben poudarek pa bomo dali tudi sinhronizaciji podatkov, saj je to eden zahtevnejših delov pri upravljanju več računalniških gruč. Ugotovimo, da nam lahko predstavljene sodobne metode povezovanja gruč zelo olajšajo njihovo upravljanje in preprosto rešijo tudi zahtevnejše probleme sinhronizacije podatkov.

Ključne besede: gruča, oblak, Kubernetes, računalniška gruča, povezovanje gruč, mreža gruč, GitOps.

Abstract

Title: Connecting Kubernetes clusters

Author: Jakob Hostnik

When we are running low on resources in our computer system, there are two standard solutions for increasing them. The first solution is to upgrade our servers and the second one is to buy more servers and connect them in a cluster. There has been a major breakthrough in this field with the release of Kubernetes system in the recent years. The system became de facto standard for cluster management and container orchestration. But when we have problems such as expensive data transfer, too much latency to our users, or we want to further increase the stability or security of our system one cluster is not always enough. We will look at the background of connecting clusters and what approaches we can use to solve our problems. Furthermore, we will also place special emphasis on data synchronization, as this is one of the more difficult parts of managing multiple computer clusters. We find that presented modern methods of connecting clusters can greatly facilitate their management and easily solve even more difficult data synchronization problems.

Keywords: cluster, cloud, Kubernetes, computer cluster, connecting clusters, cluster mesh, GitOps.

Poglavje 1

Uvod

1.1 Motivacija

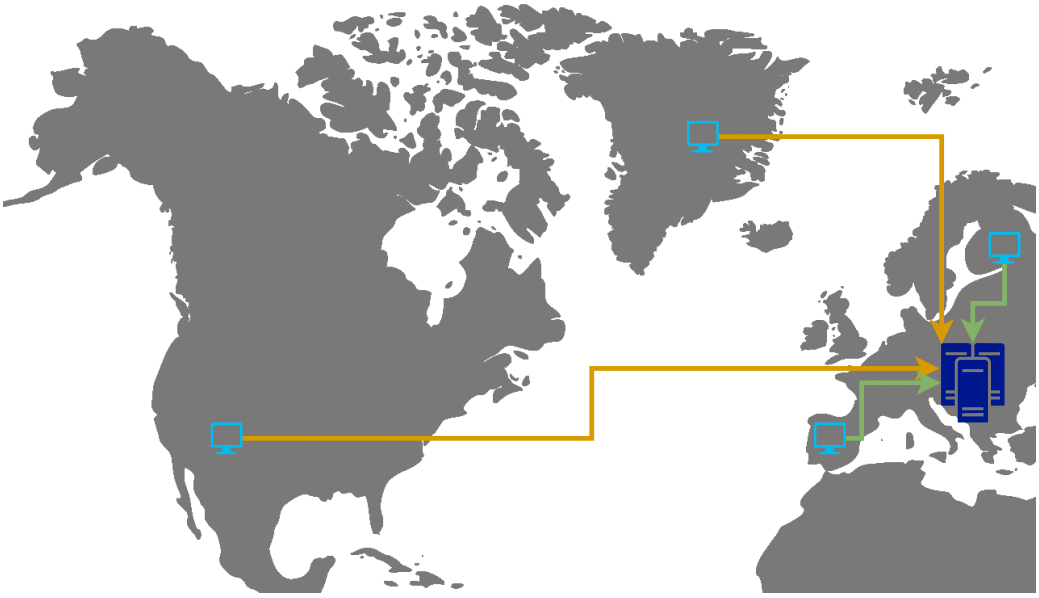
Leta 2014 je Google objavil kodo sistema za orkestracijo kontejnerjev Kubernetes [?]. Kubernetes je univerzalni način, ki nam omogoča, da več računalnikov povežemo v gručo, ki deluje kot ena samostojna enota. Povezovanje strežnikov v gruče nam vsaj teoretično omogoča visoko razpoložljivost (HA) naših storitev [?] in sinhrono delovanje več računalnikov. V primerih obravnavanih v tem delu pa ni dovolj uporaba ene same gruče, ampak moramo med seboj povezati in upravljati več gruč. V zadnjem času so razvijalci Kubernetesa začeli bolj celostno reševati ta problem. Poizkusili so ga rešiti s projektom Federation 1, po njegovi ukinitvi [?], pa razvijalci Federation 2 oziroma KubeFed trdijo, da bo projekt uspešno prešel iz alfa v beta verzijo [?].

1.2 Cilj in vsebina naloge

V tem diplomskem delu bomo obravnavali rešitve problema povezovanja računalniških gruč in primere uporabe, ki izvirajo iz potreb industrije. Vsakemu primeru bomo poiskali rešitev v Kubernetes okolju z uporabo orodja KubeFed, poudarek pa bomo dali na sinhronizacijo podatkov.

1.2.1 Prevelika latenca

Problem prevelike latence se pojavi v primeru počasnega prenosa podatkov iz naše gruče do uporabnikov. Ko problem povzroča velika fizična razdalja, ga rešimo s postavitvijo dodatne gruče bližje naših uporabnikov, denimo na njihovo celino. Zavedati se moramo, da strežniki v gruči zelo veliko komunicirajo, zato je priporočljivo, da so tudi v istem omrežju znotraj istega podatkovnega centra, saj se s tem ponavadi izognemo latenci [?].



Slika 1.1: Problem prevelike latence.

1.2.2 Višja razpoložljivost

Večkrat letno pride do izpada kakšnega večjega podatkovnega centra. To se lahko zgodi iz več razlogov najpogosteje pa gre za napake na programski opremi [?]. Če gre v takšnem primeru za oblačnega ponudnika, kjer imamo nameščeno našo gručo, pomeni, da bo hkrati nedosegljiva tudi ta. V splošnem se problem reši tako, da uporabljamo več gruč in jih namestimo v več različnih

podatkovnih centrov. V primeru izpada enega podatkovnega centra pa naše uporabnike preusmerimo v drug podatkovni center.

1.2.3 Izolacija aplikacije

Ko govorimo o izolaciji aplikacije se nanašamo na varnost pri vdoru, ali pa na večjo razpoložljivost. Glede izolacije spletnih aplikacij smo z uporabo Kubernetesa naredili že kar nekaj korakov k dobri rešitvi problema. Na primer vsaka aplikacija lahko teče v svojem kontejnerju, lahko pa jo celo izoliramo samo na določena vozlišča. A vseeno se v Kubernetesu dogajajo problemi zaradi katerih postane cela gruča nedosegljiva. Kaj takšnega se pogosto zgodi med posodabljanjem cele gruče. Z varnostnega vidika pa mislimo na dejstvo, da če nekomu uspe serija napadov in se uspešno polasti enega samega vozlišča, si lahko začne lastiti celo gručo. Ker ima administratorske pravice na vozlišču, ima posledično tudi popolno kontrolo nad vsemi drugimi aplikacijami, ki tečejo na tem vozlišču. Aplikacije lahko pripadajo istemu uporabniku ali pa celo drugim uporabnikom. Če imamo vsako od naših aplikacij v svoji gruči pa se temu lahko izognemo.

1.2.4 Drag prenos podatkov

Če se spustimo iz jedra računalniškega oblaka na njegov rob pa tam srečamo veliko zanimivih problemov. Na robu oblaka smo takrat, ko govorimo o delu naše aplikacije, ki se izvaja stran od centralnih aplikacij, ki so vedno doseljive. Takšen primer so na primer mikro podatkovni centri in gruče na majhnih računalnikih kot na primer Raspberry PI. Če naša aplikacija uporablja takšne gruče je potrebno tudi njihovo sinhrono delovanje. Takšne majhne gruče so poleg že znanega problema prevelike razdalje pogosto obsojene, da s centralnimi strežniki komunicirajo minimalno, saj zelo pogosto za prenos podatkov uporabljajo draga mobilna omrežja.

1.2.5 Razdeljevanje dela po različnih lokacijah

Na robu oblaka pa se pogosto srečamo ne samo z omejenim komuniciranjem s centralnim strežnikom ampak tudi zmanjšano razpoložljivostjo in zmogljivostjo naprav. Torej če ima ena gruča manj dela kot drugi lahko delež tega prenese na druge gruče.

Poglavje 2

Problem povezovanja gruč

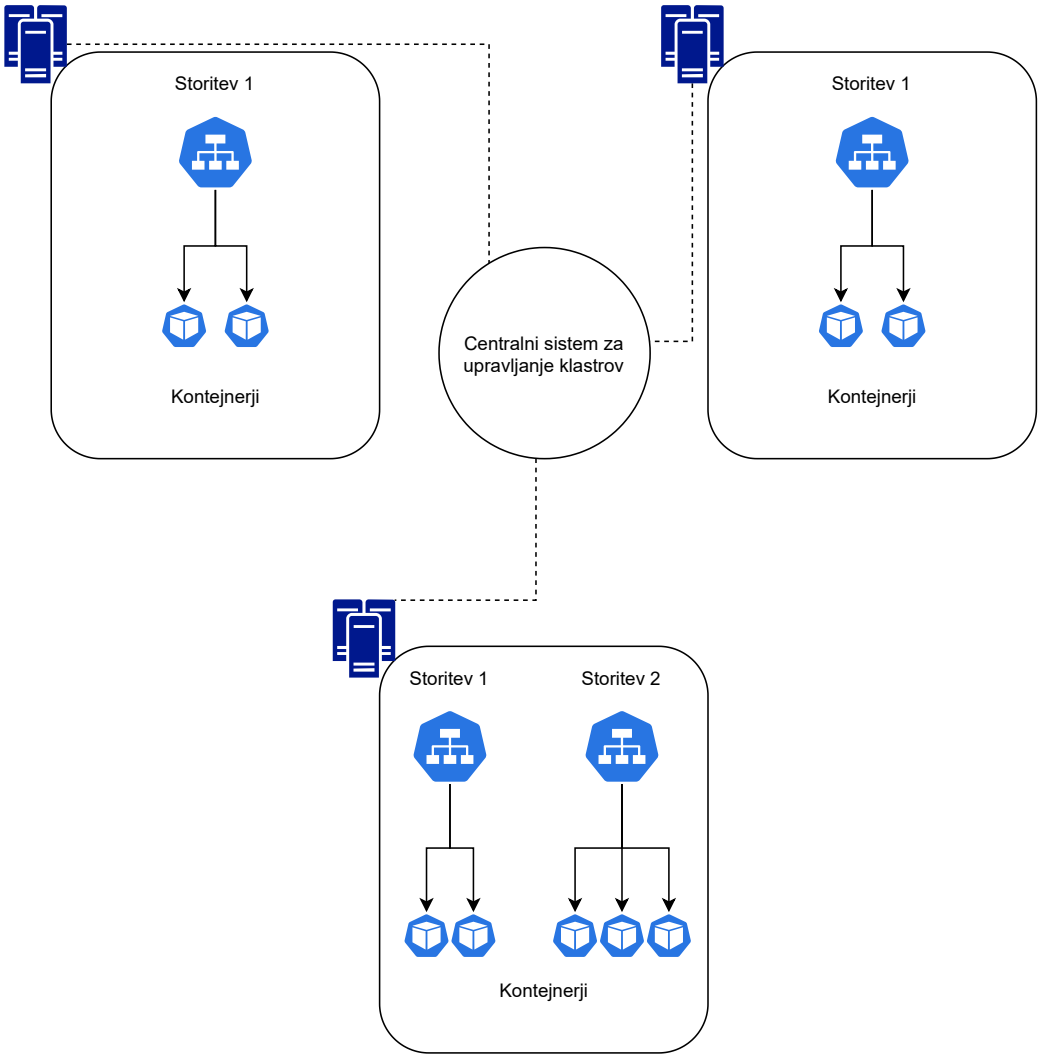
Računalniška gruča je skupina računalnikov, ki zaradi večje zanesljivosti in zmogljivosti skupaj opravlja določene storitve. Zaradi praktičnosti pa te storitve pogosto napišemo tako, da vsako storitev sestavlja več programov, ki tečejo v kontejnerjih.

Problem povezovanja več računalniških gruč je v svetu prisoten že kar nekaj časa. Ko postavimo gručo, želimo da več računalnikov deluje kot celota, a z veliko večjo zanesljivostjo, stabilnostjo in zmogljivostjo. Obstajajo primeri, ko bi radi med seboj povezali računalnike, a jih zaradi razdalje ali druačnih ovir ne moremo povezati v eno tesno povezano gručo. V takšnih primerih pa pogosto lahko vsaj računalnike na isti lokaciji povežemo v gručo. Te gruče pa potem na različne načine povežemo šibkeje.

Ko govorimo o tesni povezanosti znotraj gruče večinoma pričakujemo, da vsako vozlišče vidi vsako drugo, da vsak kontejner lahko komunicira z vsakim kontejnerjem, da so vozlišča v istem omrežju in da uporabljamo hitro interno omrežje podatkovnega centra. Pričakujemo, da sistem, ki ga uporabljamo za gručenje omogoča razporejanje zaželenih storitev in kontejnerjev med vozlišči in v primeru izpada vozlišča to vozlišče odstrani iz sistema in storitve s tega vozlišča prerazporedi na preostala vozlišča.

Ko pa govorimo o šibki povezanosti med različnimi gručami pa zaradi omejitev redko pričakujemo komunikacijo vsakega vozlišča z vsakim. Zelo

pogosto je povezava med vozlišči počasna, nezanesljiva in draga. Po navadi je vsaka gruča v svojem omrežju in je to omrežje direktno nedosegljivo ostalim gručam. Pričakujemo, da vsaka gruča skrbi za svoja vozlišča, ohranja svoje storitve in kontejnerje v delovanju. Od sistema za povezovanje gruč pa si želimo, da nam omogoča centralni nadzor nad storitvami v gručah, prerazporejanje teh storitev med gručami, dinamično odkrivanje drugih gruč in njihovih storitev, izločanje nedosegljivih gruč, povezljivost med vsemi vozlišči in kontejnerji, četudi so vozlišča v različnih omrežjih.



Slika 2.1: Primer povezanih več gruč.

Poglavje 3

Kubernetes

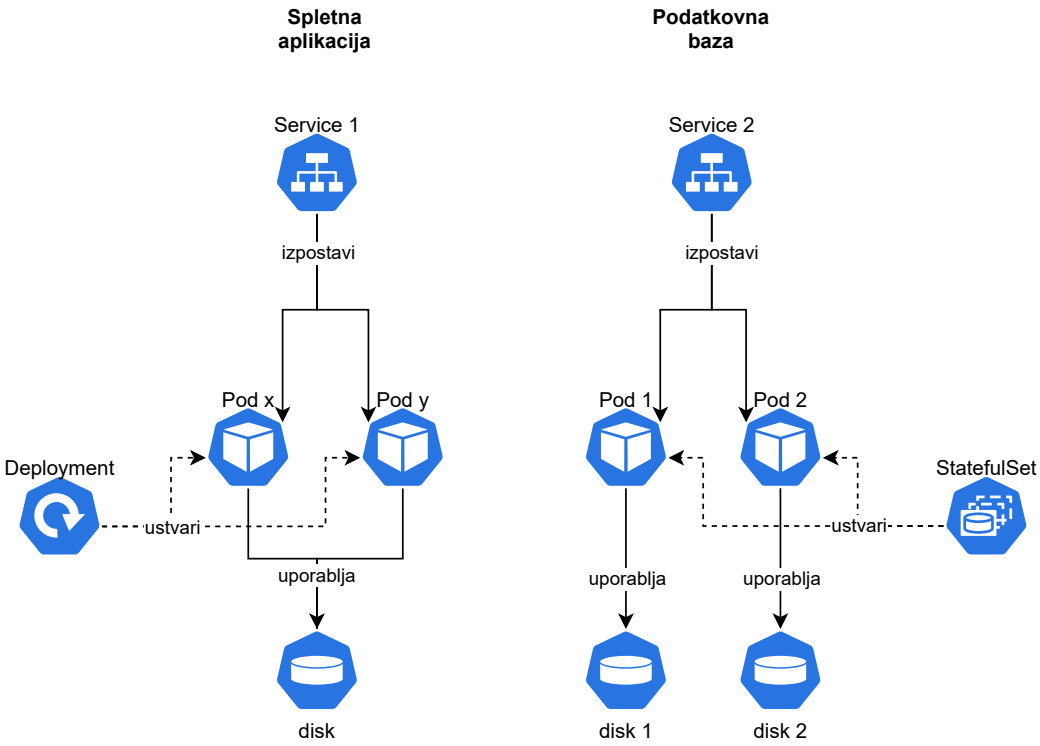
Kubernetes definira javno dostopen REST API in trenutno obstaja že več kot 70 distribucij [**?**]. Ko se bomo v tem dokumentu sklicevali na Kubernetes bomo imeli v mislih njegovo distribucijo.

3.1 Zgodovina

Leta 2014 je Google objavil in odprl kodo projekta Kubernetes [**?**]. Gre za program, ki je bil ustvarjen z namenom, da poenostavi upravljanje kontejnerjev in večjih računalniških gruč v produkcijskih okoljih. A to vseeno niso pravi začetki Kubernetesa. Začelo se je leta 2003, ko je Google začel z razvojem sistema za upravljanje njihovih internih gruč Borg. Kasneje leta 2013 je Google predstavil sistem Omega. Leta 2014 pa je Google objavil odprtokodni projekt Kubernetes. Projekt je bil zasnovan na podlagi dobrih praks upravljanja s kontejnerji, ki so se jih pri Googlu naučili skozi leta. Kasneje je upravljanje nad projektom prevzela organizacija Cloud Native Computing Fundation.

3.2 Osnovni pojmi

Kubernetes API nam omogoča, da v Kubernetes shranjujemo najrazličnejše tipe objektov. Takšne, ki so del standardnega kubernetesovega API-ja ali pa smo jih definirali sami (CRD). Najpogostejši tipi objektov, ki se pojavijo v Kubernetesu so pod, service, deployment, statefulset in objekti za delo z diski.



Slika 3.1: Primer delovanja Kubernetes objektov.

3.2.1 Pod [?]

Objekt pod običajno predstavlja nek primerek mikrostoritve. Gre za najmanjšo enoto v Kubernetesu, ki lahko teče v gruči. Sestavljen je iz enega ali več kontejnerjev, ki si delijo diske in omrežni vmesnik. To pomeni, da imajo skupen IP in se obnašajo podobno kot izolirani procesi na istem računalniku.

3.2.2 Service [?]

Objekt service večinoma označuje vse pode ene mikrostoritve. Kubernetes iz objekta v internem DNS ustvari domeno za mikrostoritev in dinamično razvršča promet med našimi podi. Service objekte uporabljamo tako, da namesto pošiljanja zahtevkov direktno na IP naslov Poda, delamo klice na ustvarjeno domensko ime storitve na primer `curl ime-storitve`. Takšen zahtevek potem dobi en izmed označenih podov v objektu service.

3.2.3 Deployment [?]

Deployment je objekt, ki mu podamo število željenih objektov pod in predlogo za njihovo izdelavo. Potem pa interne storitve Kubernetesa zagotavljajo, da bo obstajalo toliko takšnih objektov tipa pod kot smo navedli v deploymentu. Takšno stanje se poizkuša ohranjati tudi ob raznih težavah in izpadih vozlišč.

3.2.4 StatefulSet [?]

Objekt zelo podoben deploymentu, le da statefulset vsaki replikaciji poda dodeli unikatno številko. Pod, ki se ustvari s to številko ohranja diske, mrežni vmesnik, IP naslov in domensko ime. Pomembna razlika med objektoma deployment in statefulset pa je tudi v polju `volumeClaimTemplate`. Statefulset omogoča vsakemu podu, da si ustvari in uporablja svoj disk. Statefulset se najpogosteje uporablja za podatkovne baze in podobne storitve, ki morajo ohranjati stanja.

Poglavje 4

Pregled področja in literature

V tem poglavju si bomo pogledali nekaj ključnih del in literature na področju povezovanja gruč Kubernetes. V delih je pogosto za federacijo izbran sistem Federation 1 ali Federation 2, pogosto prav zaradi tesne povezanosti s sistemom Kubernetes [?] [?] [?].

V članku [?] se avtorji posvetijo federaciji z namenom povezovanja gruč pri različnih oblačnih ponudnikih. Pri tem posebno pozornost posvečajo avotmatskemu horizontalnemu skaliranju aplikacij. Za uporabo in postavitev pri več oblačnih ponudnikih so uporabili standard TOSCA, ki jim omogoča enoten deklarativni zapis strukture njihove strukture v različnih oblakih. V svoji študiji so uporabili sistem Cloudify, ki pa jim z dodatkom za Kubernetes omogoča tudi enoten način namestitve kubernetesa pri različnih oblačnih ponudnikih. Svoje gruče so še povezali v federacijo s sistemom Kuberenetes federation. Iz članka pa ni povsem razvidno ali so uporablili prvo ali drugo iteracijo sistema Kubernetes federation. V testne namene pa so v federacijo namestili še strežnik spletne igre in pokazali uspešnost avtomatskega horizontalnega skaliranja.

Lorenzo Martino je v svoji magisterski nalogi [?] v uvodu pojasnil pomembnost pristopa mikrostoritev pri razvoju aplikacij in pokazal prednosti uporabe Kubernetesa v oblaku. Kot glavno prednost je izpostavil neodvisnost od platforme in možnost uporabe v oblaku ali pa v svojem podatkov-

nem centru. Omenil je tudi hibridne rešitve, ki pa zahtevajo povezovanje in upravljanje večih gruč.

V nadaljevanju je podanih nekaj predlogov za uporabo več gruč Kubernetes, kot so: izolacija med produkcijskim in testnim okoljem, težave z latenco zaradi prevelikih fizičnih razdalj, povečevanje razpoložljivosti aplikacije, uporaba dodatne gruče v oblaku zaradi lažjega avtomaskega skaliranja vozlišč, omejitve lokacije obdelovanja podatkov. V delu je predlaganih tudi nekaj programov za upravljanje gruč. V rešitvi svojega problema pa je uporabil sistem KubeFed. Avtor omeni, da je pri svojem delu reševal problem v podjetju, ki se ukvarja s civilnimi in vojaškimi aeronavtičnimi sistemi. Ključna zahteva v podjetju pa je bila obdelava podatkov v lokalnih gručah. Nadadljevanje dela je vezano na reševanje konkretnega problema. Sinhronizaciji podatkov je v delu posvečena posebna pozornost, saj imajo v podjetju posebej označene podatke, ki se ne smejo obdelovati v oblaku in podatke, ki se lahko. Ker je KubeFed še v razvojni fazi alfa, kar pa predstavlja oviro za podjetje. Tako avtor poleg rešitve s KubeFed pripravi še svojo rešitev, kjer implementira samo potrebne funkcionalnosti.

Vir [**?**] pa se posveti področju upravljanja aplikacij na robu oblaka, kjer zaradi večjega števila gruč centralno upravljanje pride še bolj do izraza. V poročilu je postavljena Kubernetes gruča v prostrano omrežje (WAN). Avtorji so primerjali delovanje ene gruče preko prostranega omrežje z delovanjem iste gruče preko lokalnega omrežja, izpostavijo pa pomembnost previdnosti pri takšnem pristopu v gručah na robu oblaka, saj lahko pride do nepredvidljivih rezultatov. V poročilu je predstavljen tudi sistem KubeEdge, kjer pa imajo vozlišča tako kot v prvem primeru še vedno premalo avtonomnosti v primeru izpada iz omrežja. Izpostavljeno je, da ima te slabosti rešimo s federacijo in sistemom KubeFed, ki je v naaljevanju porobneje opisan.

Poglavje 5

Povezovanje Kubernetes gruč

Ko postavimo več različnih gruč imamo vedno možnost, da upravljamo vsakega posebej [**?**]. A takšen pristop zelo kmalu odpove, če imamo takšnih gruč res veliko. Ko govorimo o sistemu, ki ga uporabljamo za upravljanje več gruč, najpogosteje pričakujemo možnost prenašanja objektov med gručami. Tako lahko objekt definiramo samo enkrat in bo naš sistem ta objekt ustvaril v izbranih gručah. Odvisno od naših potreb pa nam lahko prav pride tudi dinamično odkrivanje storitev z enako definicijo v različnih gručah, komunikacijo med storitvami v različnih gručah, dinamično odkrivanje podov med gručami in komunikacijo med podi v različnih gručah. Te funkcionalnosti znotraj ene gruče nudi že Kubernetes sam. Je pa seveda odvisno od našega primera, katere funkcionalnosti želimo uporabiti in kako kompleksno postavitev potrebujemo. V nadaljevanju si bomo ogledali različne sisteme za povezovanje Kubernetes gruč, njihove glavne prednosti in značilnosti.

5.1 ArgoCD in drugi GitOps sistemi

5.1.1 Sinhronizacija objektov z uporabo GitOps sistemov

GitOps pristop pri postavljanju strukture aplikacij v Kubernetes gručah se je izkazal za dober pristop za upravljanje gruč. Osnovna ideja GitOpsa je to, da imamo našo strukturo aplikacij v Kubernetes gruči napisano v repozitoriju Git in potem je kontroler GitOps tisti, ki iz teh definicij postavi strukturo gruče. Takšen pristop se je v zadnjih letih zelo razširil in obstaja veliko podjetji, ki pri razvoju svojih spletnih aplikacij uporablja pristop GitOps. Pogosto za GitOps sistem uporabljajo kar ArgoCD.

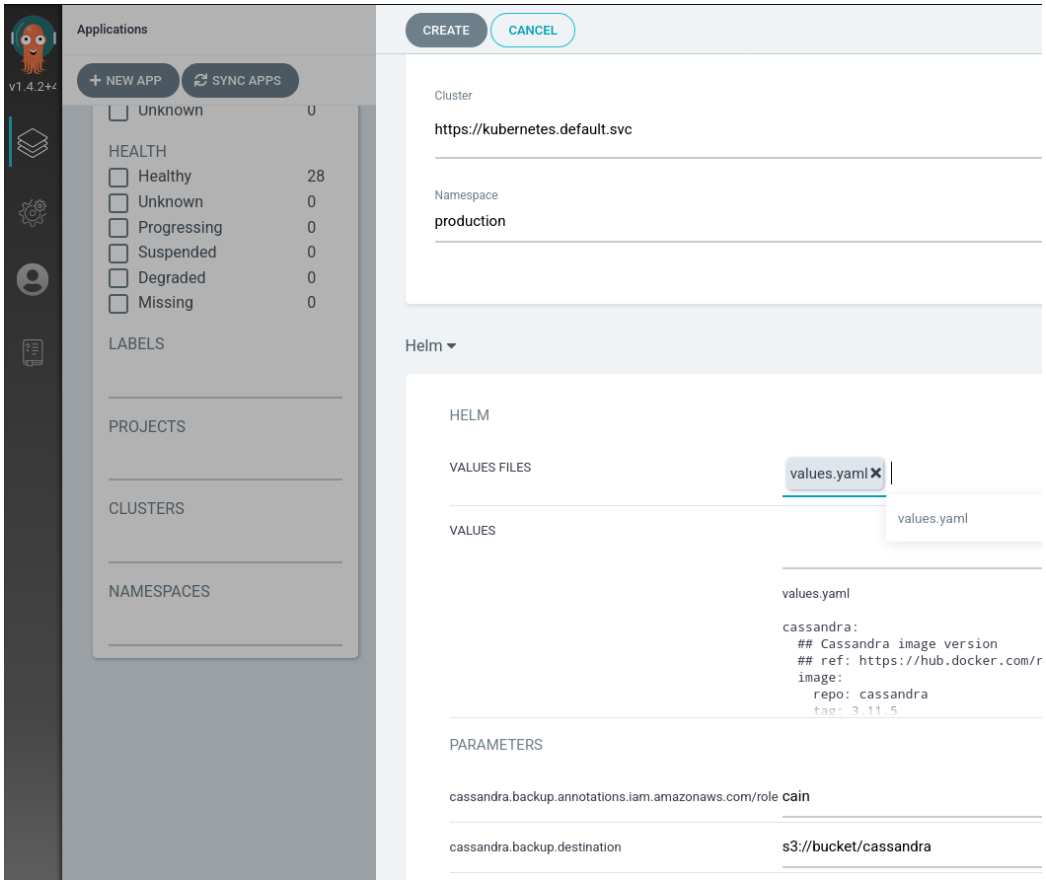
Če uporabljamo kakšnega od sistemov GitOps lahko potem iz enakega repozitorija postavimo več gruč. V osnovi takšen pristop avtomatsko pomeni, da bomo imeli na voljo samo sinhronizacijo infrastrukture in nam takšen pristop ne omogoča naprednih funkcionalnosti kot so komunikacija med podi v različnih gručah ali pa odkrivanje storitev ali podov. V nadaljevanju si bomo izbrali sistem ArgoCD in si pogledali kako bi si postavili zgoraj opisano infrastrukturo.

5.1.2 Sinhronizacija objektov z ArgoCD

ArgoCD podpira več različnih formatov konfiguracije gruče [**?**]. Najpreprostejše je, če uporabimo YAML format datoteke z definicijami objektov, ki jih želimo namestiti v vsako gručo. Za nameščanje te konfiguracije na več kot eno gručo imamo na voljo dva pristopa. Prvi način je, da v vsako gručo namestimo ArgoCD in uporabimo enak repozitorij Git v vseh. Drugi način, ki pa ga ponuja ArgoCD pa je, da lahko konfiguracijo pošljemo tudi v oddaljene gruče [**?**]. To pomeni, da moramo imeti samo v eni gruči nameščen ArgoCD kontroler.

Pogosto pa ne želimo, da imajo vse gruče popolnoma enako infrastrukturo in želimo vsaj malo prilagoditi konfiguracijo posameznege gruče. V tem pri-

meru bi uporabili format zapisa konfiguracije, ki podpira predloge. ArgoCD nam ponuja možnost, da ročno določimo spremenljivke predlogam. Tako lahko uporabimo na primer predloge HELM in ArgoCD nam bo omogočil, da vsaki gruči izberemo svojo datoteko s spremenljivkami. Glede na preprostost delovanja takšnega sistema se moramo zavedati, da od njega ne moremo pričakovati nikakršnih naprednih funkcionalnosti kot sta dinamično odkrivanje storitev ali komunikacija podov med gručami. Takšen sistem nam omogoča samo sinhronizacijo infrastrukture.



Slika 5.1: Primer uporabe helm predloge v ArgoCD.

5.2 KubeFed

9. 1. 2018 je bil po ukinjenem projektu Kubernetes Federation V1 ustvarjen Kubernetes Federation V2 ali KubeFed [**?**]. Oba projekta sta želela poenostaviti upravljanje več gruč in razporejanje Kubernetes objektov. V projektu Federation V1 je bil ubran pristop, ki je skupino gruč ali federacijo uporabniku predstavil kar kot novo Kubernetes gručo [**?**]. Uporabljal je svoj API in API kontroler, ki pa je bil združljiv s Kubernetesovim, kar pa je omogočalo tudi uporabo orodja kubectl [**?**]. Objekti, ki jih je federacija podpirala so bili kompatibilni s standardnimi Kubernetes objekti [**?**]. Takšne objekte je potem Federation V1 ustvaril tudi v ostalih gručah. Zanimiv pristop, ki pa zaradi mnogih pomanjkljivosti in pomankanja možnosti naprednejših konfiguracij ni uspel pridobiti statusa GA. GA faza v Kubernetesu pomeni, da se uporabniki lahko zanašajo na projekt, ga uporabljajo in se bo vsaj do neke mere ohranjala združljivost za nazaj. Pred dosegom te stopnje naj bi se projekt uporabljalo samo v testne namene.

Tako se je kasneje rodil projekt Federation V2 [**?**]. Glavna razlika s prvo verzijo z uporabniškega stališča je v tem, da za federacijo ne poizkuša imitirati Kubernetesovega API-ja, ampak uporablja obstoječi Kubernetesov API. Federation V2 samo predstavi nove objekte, ki pa so razširitev standardnih, kot na primer federateddeployment. Federated objekte je treba najprej vklopiti z ukazom `kubefedctl` enable.

```
kubefedctl enable deployment
```

Orodje kubefedctl si moramo namestiti na naš računalnik. Takšen Federated objekt vsebuje tri glavne lastnosti: definicija predloge primarnega objekta, postavitev v gruče in prepis lastnosti originalnega objekta za posamezne gruče. Takšen pristop je zelo široko zastavljen in omogoča tudi federacijo CRD objektov.

```
apiVersion: types.kubefed.io/v1beta1
kind: FederatedDeployment
spec:
```

```
placement:
  clusterSelector:
    # izbira gruč
    matchLabels: {}
    ...
template:
  # specifikacije deployment objekta
  spec:
  ...
overrides:
  # prepis konfiguracije za posamezne gruče
  - clusterName: gruca-1
    clusterOverrides:
        # nastavi polje replicas na vrednost 5
      - path: "/spec/replicas"
        value: 3
...
```

Federation V2 podpira poleg sinhronizacije infrastrukture tudi odkrivanje storitev v drugih gručah prek DNS zapisov. Omenja pa se možnost odstranitve te funkcionalnosti [**?**], ki je že sedaj privzeto izklopljena. Preden pa uporabimo KubeFed pa se moramo zavedati, da je projekt v času pisanja diplomske naloge še vedno v razvojni fazi alfa in lahko mine še nekaj časa preden doseže status GA, če slučajno ne bo šel po stopinjah svojega predhodnika.

5.3 Cilium

Cilium je odprtokodni program, ki nam omogoča napredne varnostne in omrežne nastavitve v naši gruči [**?**]. Program na tretji in četrti omrežni plasti zagotavlja osnovne principe varnosti in zaščite, kot na primer zapiranje portov in omejevanje komunikacije. Poleg tega pa Cilium zagotavlja tudi

naprednejšo varnost na sedmi omrežni plasti, saj nam omogoča omejevanje in filtriranje HTTP zahtevkov in podobne varnostne funkcionalnosti na popularnih protokolih aplikacijskega nivoja [**?**]. Zaradi naprednih možnosti, ki jih Cilium ponuja, se podjetja velikokrat odločijo za uporabo Ciliuma v svojih Kubernetes gručah, tudi ko ne povezujejo več gruč med seboj.

Ker Cilium implementira precejšni del mreženja in povezovanja v Kubernetesu, pa nam s tem lahko ponudi tudi nekaj zelo naprednih možnosti, ko med seboj povezujem več različnih Kubernetes gruč. Tako nam kot ključno prednost Cilium omogoča tudi komunikacijo med podi v različnih gručah [**?**] in uporabo globalnih objektov service, ki so sposobni delati razporejanje prometa med različnimi gručami. Takšne objekte definiramo z anotacijo io.cilium/global-service [**?**]. Omogoča nam tudi omejevanje povezovanja med gručami z njihovim objektom CiliumNetworkPolicy [**?**]. Ko postavljamo mrežo gruč, pa se moramo še vedno zavedati, da Cilium ne rešuje problema, če so naše gruče skrite v različnih zasbenih omrežjih. Ključno pri uporabi Ciliuma za povezovanje gruč je, da so vsa naša vozlišča dosegljiva med seboj. A četudi so naše gruče v med seboj nedosegljivih zasebnih omrežjih, pa je problem z lahkoto rešljiv z uporabo sistema VPN, ki nam omogoča, da vsa vozlišča povežemo v eno virtualno omrežje [**?**].

Kljub naprednim funkcijam, ki nam jih Cilium ponuja, pa se moramo zavedati, da se Cilim ukvarja samo s povezovanjem gruč na omrežnem nivoju. Ne omogoča enotnega upravljanja in sinhroniziranja objektov med gručami zato moramo objekte sinhronizirati sami. Ampak zaradi dovolj široke zasnove Kubernetesovega vmesnika so rešitve med seboj kompatibilne. Torej lahko uporabimo napredno mreženje Ciliuma in objekte sinhroniziramo s KubeFed in GitOps pristopom.

Poglavje 6

Priprava sistema gruč za testiranje

6.1 Raspberry PI 4

Za namene testiranja različnih načinov povezovanja Kubernetes gruč moramo najprej postaviti nekaj gruč. Zaradi preprostosti in nizke cene, predvsem pa ker se koncepti zaradi tega ne spremenijo, bomo za naša Kubernetes vozlišča uporabili Raspberry PI 4. Na višjem nivoju pa gre še vedno za Kubernetes gručo in je delo zelo podobno, če uporabimo nekaj 1000 vozlišč v gruči v oblaku ali pa lokalno gručo z enim vozliščem. Raspberry PI je zelo majhen in manj zmogljiv računalnik na eni sami plošči. Ključni prednosti takšnih računalnikov pa sta prav velikost in cena. Na vsak Raspberry PI se bo namestila Kubernetes gruča z enim samim vozliščem. Fizična postavitev gruč je prikazana na sliki 6.1. Takšna postavitev pa je lahko tudi primer gruče na robu oblaka, kar je bolj podrobno opisano v poglavju 9.

6.2 K3S in K3OS

Obstaja več implementacij Kubernetesa in mi bomo uporabili z viri varčno odprtokodno implementacijo K3S od podjetja Rancher [**?**]. Hkrati so v pod-

Slika 6.1: Postavitev Raspberry PI gruč.

jetju Rancher pripravili distribucijo operacijskega sistema Linux K3OS, ki jo lahko namestimo na katerikoli računalnik [**?**]. Majhna težava se pojavi, ker še ni pripravljene uradne verzije operacijskega sistema za ploščice Raspberry PI. A k sreči se je v ta namen začel odprtokodni projekt PiCl k3os image generator, ki nam iz slik operacijskih sistemov K3OS in Raspberry OS in konfiguracijskih datotek zgradi novo sliko operacijskega sistema za naš Raspberry PI [**?**]. Konfiguracijske datoteke, ki jih moramo priložiti so standardne YAML datoteke, ki jih podpira K3OS. Vanje zapišemo nastavitve kot so SSH javni ključi za dostop, podatki od WiFi omrežja na katerega se povezujemo, geslo, žeton za povezavo s Kubernetes gručo in način v katerem želimo zagnati K3S na sistemu [**?**]. V našem primeru smo vse K3S programe zagnali v strežniškem načinu in nobenega v načinu delovnega vozlišča, saj želimo, da vsak Raspberry PI predstavlja svojo gručo.

```
ssh_authorized_keys:
- ssh-rsa ...
hostname: gruca-1
k3os:
  ntp_servers:
  - ...
  password: ...
  token: ...
  dns_nameservers:
  - ...
  wifi:
  - name: ...
    passphrase: ...
  k3s_args:
  - server
```

6.3 Demonstracijska spletna aplikacija

Za potrebe testiranja je bilo potrebno naredili novo testno mikrostoritev. Ker se v tem diplomskem delu želimo osredotočiti na resnične probleme v industriji, mora ta aplikacija omogočati tudi shranjevanje podatkov v podatkovno bazo.

Koda, ki je javno objavljena v Git repozitoriju [**?**], je napisana v programskem jeziku Go. Iz kode je bil generiran kontejner, ki je objavljen v javnem Docker repozitoriju [**?**]. Ob tem velja opozoriti, da Raspberry PI uporablja ARM arhitekturo procesorja, kar je zahtevalo posebno pozornost.

Aplikacija deluje preprosto. Na mrežnih vratih podanih s spremenljivko okolja izpostavi vmesnik REST z dvema preprostima HTTP klicema. GET klic na pot `/users` nam bo vrnil vse uporabnike, ki so zapisani v tabeli v bazi, s klicem POST na isto pot pa poskrbimo, da se podatki uporabnika iz našega zahtevka shranijo v tabelo v bazo.

```
# ukaz za dodajanje uporabnika
curl -X POST localhost/users \
  --data '{"name": "John", "lastname": "Doe"}'
# ukaz za prikaz vseh uporabnikov
curl localhost/users
```

Za shranjevanje podatkov bomo uporabili 2 različni SQL bazi podatkov. Postgres, ki je preprosta za lokalni razvoj, a ne omogoča napredne sinhronizacije podatkov med strežniki in CrateDB, ki je bil zasnovan kot SQL baza na več vozliščih in nam omogoča napredne sinhronizacije tudi med različnimi strežniki in gručami. K sreči pa CrateDB implementira PostgreSQL vmesnik in nam kode za prehod med bazami ni potrebno spreminjati.

6.4 Namestitev KubeFed

Kot ena izmed ključnih komponent složnega delovanja več gruč je njihovo upravljanje. V te namene bomo uporabili program KubeFed, ki ga moramo namestiti na eno izmed gruč, ki jih želimo povezati skupaj. Ker je izdelek še v razvoju in še ni prišel iz alfa faze, nimajo objavljene verzije za procesorje ARM. Zato je bilo iz kode KubeFed potrebno zgraditi novo sliko kontejnerja, ki je javno objavljena [**?**]. Potem pa smo uporabili originalno Helm predlogo, kjer smo samo zamenjali originalno sliko kontejnerja z našo. Za delo s KubeKed pa moramo na svoj računalnik namestiti orodje kubefedcli. Z uporabo ukaza `kubefedctl join` povežemo vse tri gruče v kubefed sistem.

```
kubefedctl join gruca-1
kubefedctl join gruca-2
kubefedctl join gruca-3
```

S tem smo uspešno povezali več Kubernetes gruč v sistem KubeFed. Seznam vseh povezanih gruč pa lahko preverimo tako, da izpišemo seznam objektov tipa kubefedclusters. V našem primeru imamo povezane tri gruče, kar se vidi iz sledečega izpisa.

```
kubectl get kubefedclusters

NAME        AGE   READY
gruca-1     1d    True
gruca-2     1d    True
gruca-3     1d    True
```

Sedaj lahko z uporabo ukazov `kubefedctl enable` in `kubefedctl federate` naše objekte dodajamo v vse gruče hkrati. Več o tem je napisano v poglavjih, kjer ukaze tudi uporabljamo.

Poglavje 7

Povezovanje med podatkovnimi centri

7.1 Problem velike latence

V industriji je zelo malo primerov spletnih aplikacij, ki jim ni potrebno hraniti stanja. Ko neko podjetje, lastnik aplikacije poseže po globalnem trgu zelo hitro ugotovi, da stranke, ki niso blizu podatkovnega centra precej dlje čakajo pred ekrani, da se naloži njihova spletna aplikacija in njihovi podatki. Takšen problem se v splošnem rešuje tako, da našo aplikacijo postavimo še na dodaten strežnik bližje uporabniku. Rešitev se sliši preprosta, a vseeno se tu srečamo z zelo zahtevnimi problemi v računalništvo. Najbolj očiten primer je sinhronizacija podatkov. V našem primeru bomo uporabili podatkovno bazo CrateDB [**?**], novejšo alternativo standardnim SQL podatkovnim bazam. CrateDB ima v primerjavi s tradicionalnimi podatkovnimi bazami boljšo podporo za sinhronizacijo podatkov med vozlišči. Poleg vsega pa nam za uporabo podatkovne baze CrateDB ni potrebno konceptualno spreminjati naše aplikacije, saj podpira vmesnik od podatkovne baze PostgreSQL.

7.2 Povečanje razpoložljivosti aplikacije

Če je čim višja razpoložljivost za našo aplikacijo kritičnega pomena in smo že poskrbeli za visoko razpoložljivost (HA) aplikacije v naši gruči, še vedno lahko pride do situacije, ko iz omrežja izpade cel podatkovni center. Spomnimo se, da Kubernetes najbolj učinkovito deluje, če naša vozlišča uporabljajo hitro interno omrežje podatkovnega centra. V primeru naravnih nesreč ali hujših vremenskih pogojev pomeni, da je nedosegljiv cel podatkovni center in s tem gruča v njem. Če uporabljamo oblak, pa gremo lahko še korak dlje z zagotavljanjem razpoložljivosti. Če nam ni dovolj niti to, da uporabimo različne razpoložljivostne cone in podatkovne centre oblačnih ponudnikov, lahko postavimo naše gruče pri več različnih ponudnikih. Tu Kubernetes pride zelo do izraza, saj kljub nekaj neenakostim skozi implementacije ohranja enak vmesnik in je zato takšna postavitev precej lažja, kot bi bila brez uporabe Kubernetesa.

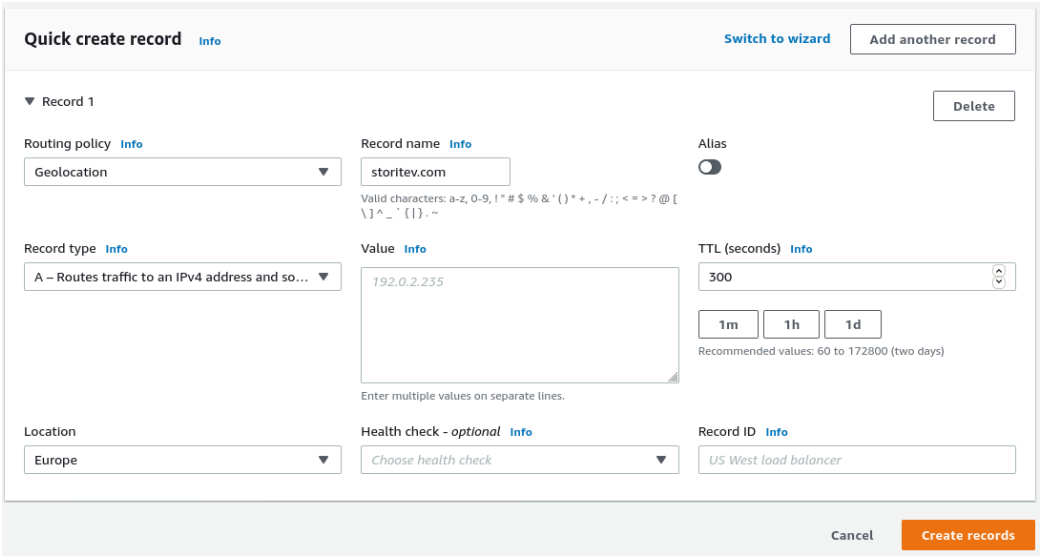
7.3 Povezovanje med podatkovnimi centri

Rešitev za oba problema je identična. Postaviti moramo gruče v več različnih podatkovnih centrih in jih nastaviti, da bodo delovali usklajeno. Odvisno od problema bodo te podatkovni centri morda bližje uporabniku, morda v lasti različnih oblačnih ponudnikov ali pa oboje. Ampak princip ostaja enak.

7.4 Razporeditev uporabnikov po gručah

Ko imamo na vsaki gruči javno izpostavljen Kubernetesov service objekt in postavljene primerne ingress objekte, moramo še vedno uporabnike preusmeriti na njim najbližjo gručo. Uporabnike lahko mi usmerimo avtomatsko z DNS zapisi, ki omogočajo usmerjanje na podlagi geolokacije. Lahko uporabimo in namestimo zunanji DNS skozi Kubernetes ali pa to opravimo kar mimo Kubernetesa. V naših lokalnih testnih gručah bomo ta korak preskočili

in jih ne bomo usmerjali preko javnih DNS strežnikov, saj v lokalnem okolju to ni smiselno.



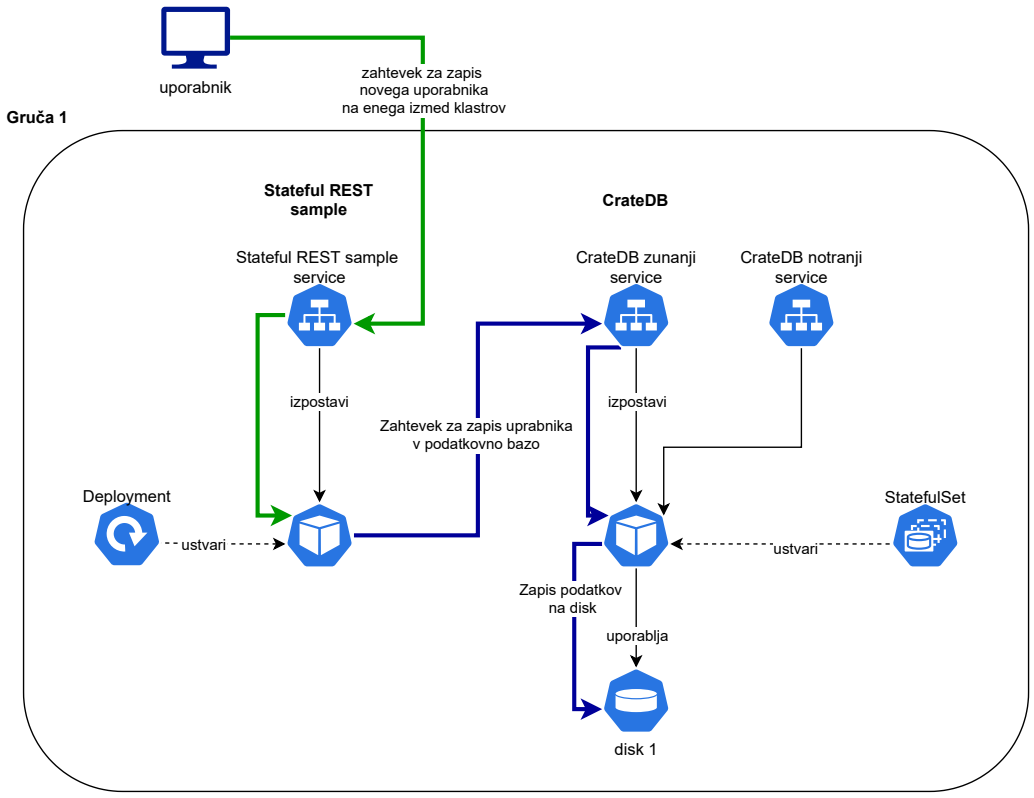
Slika 7.1: Ustvarjanje geolokacijskega DNS zapisa v storitvi ROUTE53.

Naslednja možnost pa je rešitev, ki se jo zelo pogosto poslužujejo internetne računalniške igre, da so naši strežniki popolnoma ločeni in se vsak uporabnik sam odloči na kateri gruči ali strežniku želi igrati. V takšnih primerih se lahko tudi izognemo problemu sinhronizacije podatkov med strežniki, kar zelo poenostavi upravljanje naših gruč.

7.5 Definicija infrastrukture za naš primer

V našem primeru spletne aplikacije bomo imeli v vsaki gruči eno postavitev aplikacije „Stateful rest sample“ z deployment objektom. Da pa aplikacijo izpostavimo izven gruče, pa bomo uporabili objekt service. Aplikacija bo za shranjevanje uporabljala podatkovno bazo CrateDB, ki bo postavljena z objektom statefulset, diskom na lokalni SD kartici, in dvema objektoma service. Prvi objekt service je zunanji in se bo uporabljal za dostop do baze, drugi pa je notranji in ga bomo uporabljali za prepoznavo ostalih primerkov

CrateDB v gruči. Vsa konfiguracija je javno objavljena na Git repozitoriju[?].



Slika 7.2: Infrastruktura vsake gruče v primeru demonstracijske aplikacije

Postavimo jo z ukazom `kubectl apply -f diploma-demo-1` Takoj preverimo, če aplikacija deluje in če lahko podatke zapisujemo v bazo, tako da prek demo aplikacije poizkusimo dodati uporabnika in izpisati vse uporabnike. To naredimo z naslednjima `curl` ukazoma.

```
curl -X POST gruca-1/users \
    --data '{"name": "John", "lastname": "Doe"}'
curl gruca-1/users
```

7.6 Implementacija s KubeFed

Najprej se moramo odločiti za katere tipe objektov bomo vklopili federacijo oziroma za katere bomo želeli univerzalno upravljanje. V našem primeru gre za service, deployment in statefulset. Vklopimo jih z naslednjim ukazom, ki za nas ustvari nove federated tipe objektov na izbranih tipih.

```
kubefedctl enable <ime tipa>
```

Ko smo si vklopili federacijo na vseh potrebnih tipih pa moramo še vklopiti avtomatsko upravljanje na specifičnih objektih. V našem primeru želimo za to uporabiti ukaz `kubefedctl federate`.

```
kubefedctl federate deployment stateful-rest-sample
kubefedctl federate service stateful-rest-sample
kubefedctl federate statefulset crate
kubefedctl federate service crate-internal
kubefedctl federate service crate-external
```

Izvršeni ukazi ustvarijo federated objekte, ki uporabijo postavitev v vse gruče in za predlogo kar podane objekte. Tako je za nas rezultat izvršenih ukazov kreiranje federated objektov in posledično kopiranje objektov v vse naše povezane gruče.

Po preizkusu delovanje s `curl` ukazom opazimo, da podatki med gručami še vedno niso sinhronizirani. Uporabniki, ki jih vnesemo v eno gručo se še ne sinhronizirajo v ozadju. Na tej točki se ustavijo nekatere spletne aplikacije in prepustijo izbiro strežnika oziroma gruče kar uporabniku.

7.7 Sinhronizacija podatkov

Če želimo pred uporabnikom skriti, da uporabljamo več gruč, moramo poleg geolokacijskih DNS zapisov, urediti tudi avtomatsko sinhronizacijo podatkov. Sicer v našem primeru res uporabljamo samo en primerek CrateDB baze na gručo, a vseeno smo na nivoju sinhronizacije znotraj gruče to stvar že

uredili. Zopet se moramo spomniti, da so tudi gruče podatkovnih baz pogosto narejene tako, da najbolje delujejo, če so vozlišča v hitrem lokalnem omrežju. Zaradi tega mnoge baze, ki podpirajo sinhronizacijo podatkov znotraj gruče, podpirajo tudi sinhronizacijo med različnimi podatkovnimi centri.

7.7.1 Uporaba primerne podatkovne baze

Najlažje je sinhronizirati podatke, če uporabimo podatkovno bazo, ki ima sinhronizacijo med različnimi gručami že podprto. CrateDB podpira sinhronizacijo tudi preko razpoložljivostnih con. Vseeno pa je mišljeno, da vsa vozlišča povežemo v enako podatkovno gručo. To pomeni, da morajo vsa vozlišča imeti dostop do vseh. Zelo elegantna rešitev bi bila uporaba sistema Cilium in uporaba globalnih storitev, saj nam Cilium že omogoča komunikacijo vsakega poda z vsakim, tudi če so ti v različnih gručah. Druga možnost pa je, da izpostavimo vsak pod s svojim javnim IP naslovom in jih ročno povežemo v gručo.

Potem pa moramo nastaviti še nastavitve, ki jih baza podpira za zmanjšanje prometa in zagotavljanje željene razpoložljivosti med gručami [**?**]. Podobne načine sinhronizacije podpira tudi na primer podatkovna baza Cassandra [**?**].

7.7.2 Podatke sinhroniziramo sami

Sinhronizacija podatkovne baze je težak problem. Če ne uporabimo primerne podatkovne baze ali pa želimo sinhronizirati samo določene stvari preko gruč bomo sinhronizacijo podatkov verjetno morali napisati sami. To pomeni, da bomo ustvarili novo mikrostoritev, ki bi v ozadju kopirala ključne podatke med podatkovnimi centri. Ker samo mi poznamo naš konkreten primer uporabe, je takšen pristop lahko najbolj učinkovit.

V našem primeru bomo s preprosto skripto kopirali uporabnike iz ene aplikacije v drugo kar z uporabo našega REST vmesnika. To bomo storili v drugem Ubuntu kontejnerju z uporabo ukazov `curl` za izvajanje REST klicev

in ukazom `jq` [?] za razčlenjevanje podatkov. Podatki se sinhronizirajo vsakih 10 sekund. Primer še testiramo in dobimo spodnji izhod, kar potrdi, da so se podatki uspešno sinhronizirali.

```
curl -s -X POST gruca-1/users|jq \
  --data '{"name": "John", "lastname": "Doe"}'
curl -s gruca-2/users|jq
[{"Name":"John","Lastname":"Doe"}]
```

Poglavje 8

Upravljanje izoliranih aplikacij

8.1 Zmanjševanje posledic vdorov in izpadov

Računalniška stroka si je že nekaj časa nazaj priznala, da popolnega sistema ne more ustvariti: sistema, ki se ne more sesuti, sistema, ki bo ves čas razpoložljiv in sistema, v katerega ne bo mogoče vdreti. To vsake toliko časa potrdijo tudi najbolje upravljani veliki sistemi kot so AWS, Google, Facebook z izpadi ali vdori na njihovih storitvah. Vseeno pa kljub vdorom in napakam, zaradi katerih postanejo naši sistemi nedosegljivi, vedno lahko poizkusimo zmanjšati posledice ob morebitnem vdoru ali napadu.

8.1.1 Izpadi aplikacije

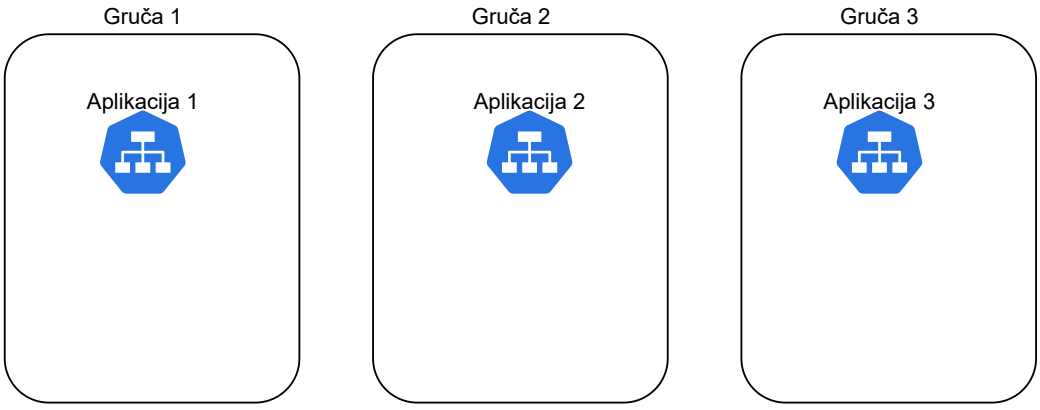
Kljub temu, da smo naše aplikacije namestili na različne gruče in je s tem aplikacija odporna na izpad ene gruče, pa lahko ob hujših nepravilnostih delovanja ene aplikacije in napaki pri nastavitvi gruč kaskadno izpadejo tudi vse gruče na katerih imamo aplikacijo nameščeno. Takšen primer bi bil, če ena aplikacija ali mikrostoritev zavzame vse vire v gruči hkrati pa odpovejo ostale varovalke, ki jih ponuja že sam Kubernetes. V takšnih primerih bo odpovedal cel naš sistem namesto samo del sistema. Zato se lahko odločimo, da bomo nekatere bolj kritične aplikacije ali mikrostoritve postavili v gručo, kjer napake drugih aplikacij ne bodo vplivale na naše delovanje. A vseeno se

moramo zavedati, da je ta korak smiseln šele ko smo opravili že vse predhodne preventivne ukrepe, kot so razdelitev aplikacije na mikrostoritve, kontejnerijzacija, izolacija na posamezno Kubernetes vozlišče, pravilna nastavitev omejitev avtomatskega povečevanja in še mnoge druge.

8.1.2 Vdori

Podobno kot pri izpadih aplikacije je tudi pri preprečevanjih posledic vdorov. Najprej moramo poskrbeti za primerno zaščito Kubernetes vozlišč, naše aplikacije, kriptiranje komunikacije med mikrostoritvami, uporabo nepriviligiranih in neadministratorskih kontejnerjev. Če pa nam vsi zgoraj našteti in ostali priporočeni ukrepi niso dovolj ali pa se zavedamo, da imamo v gručah manj varne aplikacije in napadalec prek teh aplikacij ne sme dostopati do podatkov kritičnih aplikacij, potem pa je smiselno kritične aplikacije izolirati v svoje gruče.

8.2 Implementacija s Kubefed



Slika 8.1: Primer izoliranih aplikacij.

Ena izmed treh glavnih lastnost federiranih objektov je možnost izbire gruč na katerih se bo določen objekt ustvaril. S tega stališča je naš primer

zelo preprost. Samo določimo da se naša aplikacija izvaja na gruči 3 namesto na vseh. Tokrat za federacijo ne moremo uporabiti ukaza `kubefedctl federate`, ampak moramo spisati konfiguracijo federiranih objektov sami. Najprej bomo z ukazom `kubectl tag` označili našo izolirano gručo (ali več njih). Potem pa bomo lastnosti `.clusterSelector.matchLabels` vsakega federiranega objekta, ki ga želimo izolirati, dodali označbe vseh izoliranih gruč. V takšnih primerih se nam ni potrebno posebej ukvarjati s sinhronizacijo podatkov, saj smo ali vse podatke obdržali v isti gruči ali pa sinhroniziramo na enak način kot v poglavju 7.

Poglavje 9

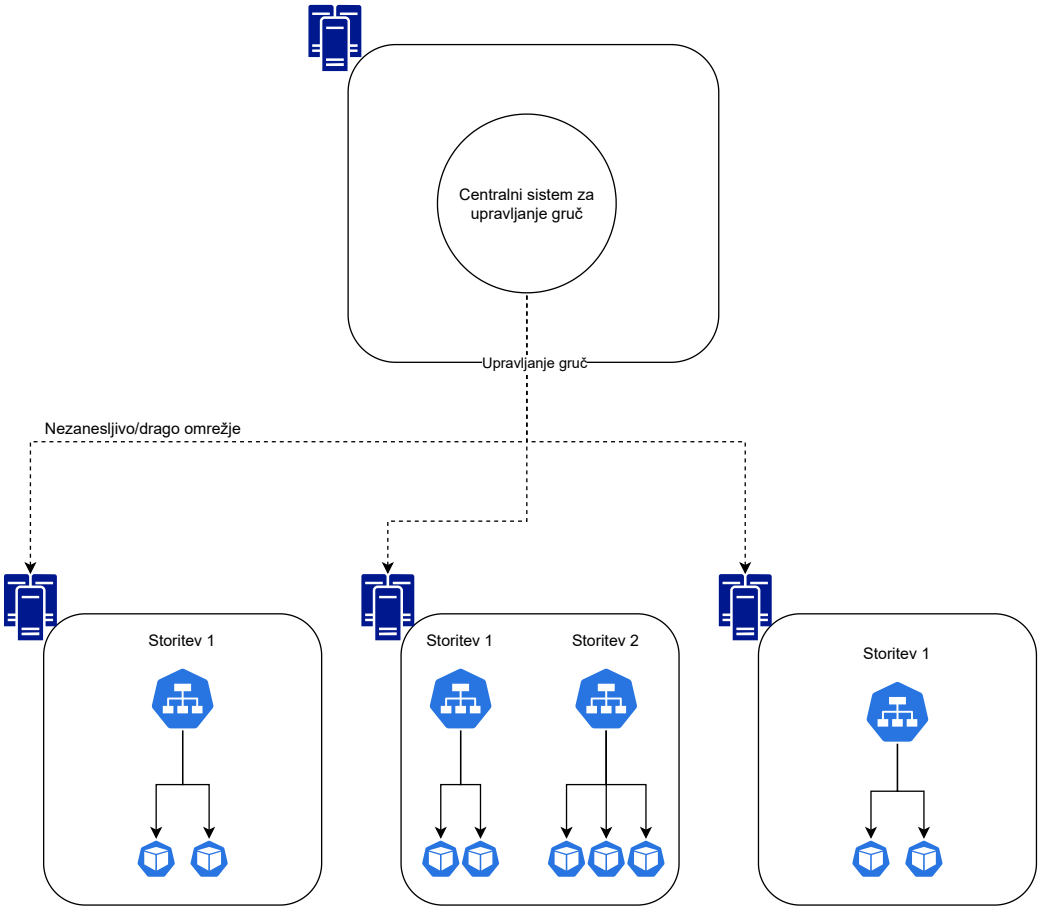
Upravljanje gruč na robu oblaka

9.1 Gruče na robu oblaka

Razlogov zakaj gruče postavljamo na rob oblaka oziroma fizično bližje končnemu uporabniku je več. Pogosto ne želimo na glavni strežnik pošiljati vseh podatkov, ampak želimo podatke obdelati že lokalno, da lahko po omrežju pošiljamo samo agregirane podatke. Smiselnost takšne postavitve pride še posebej do izraza, če podatke prenašamo po dragem mobilnem omrežju. A možnih postavitev in razlogov zanje pa je več. Na primer lahko gre za zahteve strank, da se podatki obdelujejo lokalno, lahko gre za zakonske omejitve ali pa želimo operacije izvajati na napravah, ki si jih ne lastimo direktno. V našem primeru se bomo osredotočali na upravljanje takšnih gruč.

9.2 Implementacija s KubeFed

Ko enkrat povežemo vse gruče s kubefedctl join je njihovo upravljanje preprosto. Samo nastavimo v kateri gruči želimo katere objekte in naša naloga je končana. Zavedati se sicer moramo, da nekaj komunikacije porabi tudi KubeFed za sinhronizacijo.



Slika 9.1: Primer izoliranih aplikacij.

Nam pa KubeFed omogoča še eno lepo možnost s svojo strukturo in lahko s svojim kontrolerjem in KubeFed vmesnikom implementiramo še dodatne funkcionalnosti kot so razporejanje obremenjenosti med lokalnimi strežniki in po potrebi povečujemo število primerkov ali pa kar razporejamo opravila s Kubernetes Job objekti.

Z zelo preprosto integracijo v Kubernetes nam KubeFed vmesnik tu res omogoča zelo preprosto implementacijo katerekoli naše rešitve.

9.3 Sinhronizacija podatkov

V primeru gruč na robu oblaka bomo sinhronizacijo verjetno implementirali sami, saj le mi vemo kakšen problem rešujemo in zakaj smo sploh postavljali gruče na robu oblaka.

Za primer vzemimo hipotetični varnostni sistem korporacije, ki centralno spremlja varnost v posameznih podružnicah. Sistem ima eno nadzorno kamero pri vhodu v vsako podružnico. Želimo, da naša kamera prepoznava obraze in na podlagi tega dovoljuje zaposlenim vstop. V našem centralnem sistemu pa želimo, hraniti seznam vstopov. En način reševanja tega problema je z gručami na robu oblaka. V vsako podružnico bi postavili gručo računalnikov Raspberry PI, ki so dovolj zmogljivi, da obdelujejo posnetke kamer in prepoznavajo obraze. Če posnetke obdelujemo lokalno, se izognemo pošiljanju veliki količini podatkov na centralne strežnike, posledično pa bo hitrejše tudi preverjanje zaposlenih. Tako bi na centralni strežnik pošiljali samo številko zaposlenega in čas vstopa. Takšen pristop bi prišel še toliko bolj do izraza, če imajo podružnice dostop do interneta samo prek dragega mobilnega omrežja, kjer lahko z zmanjšanjem prometa zelo zmanjšamo tudi stroške podjetja. Vse te gruče na podružnicah bi imele zelo podobno strukturo in jih je smiselno centralo upravljati s kakšnim sistemom za povezovanje. Tu bi lahko uporabili KubeFed ali GitOps pristop. Pošiljanje podatkov na centrali strežnik pa bi morali napisati sami in ga vgraditi v naš program za prepoznavo obrazov.

Poglavje 10

Sklepne ugotovitve

V diplomskem delu smo si pogledali teoretično ozadje povezovanja več računalniških gruč in osnove Kubernetesa. Predstavljenih je bilo tudi nekaj popularnih orodij za delo z več Kubernetes gručami. V praktičnem delu pa smo se posvetili predvsem reševanju pogostih problemov v industriji, ki zahtevajo povezovanje več gruč. Zato pa je bilo potrebo postaviti tudi ustrezno okolje za preizkušanje naših rešitev.

Z razvojem Kubernetesa se je razvilo tudi zelo veliko odprtokodnih orodij, ki omogočajo lažje upravljanje in povezovanje več gruč. Tako so napredne tehnologije prišle v roke širšemu krogu ljudi in jim omogočajo preprostejše reševanje težav. Kubernetes pa je s standardizacijo orkestracije zelo olajšal tudi možnost gostovanja aplikacije pri več različnih oblačnih ponudnikih, kjer se zopet pojavi problem povezovanja več gruč.

Področje orkestracije in povezovanja gruč se bo še zelo razvijalo in tema bo zagotovo zahtevala še veliko diplomskih del.