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:

Preučite področje povezovanja gruč Kubernetes. Analizirajte problematiko povezovanja in preglejte nekaj primerov uporabe, ki izvirajo iz potreb podjetij. Pripravite tudi prototipno rešitev povezovanja, pri čemer se osredotočite na rešitve, nastale v odprtokodni skupnosti Kubernetes.

Na tem mestu bi se zahvalil mentorici izr. prof. dr. Mojci Ciglarič in somentorju asist. dr. Matjažu Pančurju za pripravljenost, mentorstvo, 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 strežnikih začne zmanjkovati virov, obstajata dva standardna načina za povečanje virov v našem sistemu. Prva možnost je, da strežnike nadgradimo, druga pa, da jih kupimo več in jih povežemo v računalniško gručo. V zadnjih letih se je na tem 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, ko želimo še bolj povečati stabilnost ali varnost našega sistema. Predstavili bomo ozadje povezovanja gruč in pristope, ki jih lahko uporabimo za reševanje naših problemov. Poseben poudarek bomo dali tudi na sinhronizacijo podatkov, saj je to eden zahtevnejših delov pri upravljanju več računalniških gruč. Ugotavljamo, 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 to a cluster. There has been a major breakthrough in this field with the release of the Kubernetes system in 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

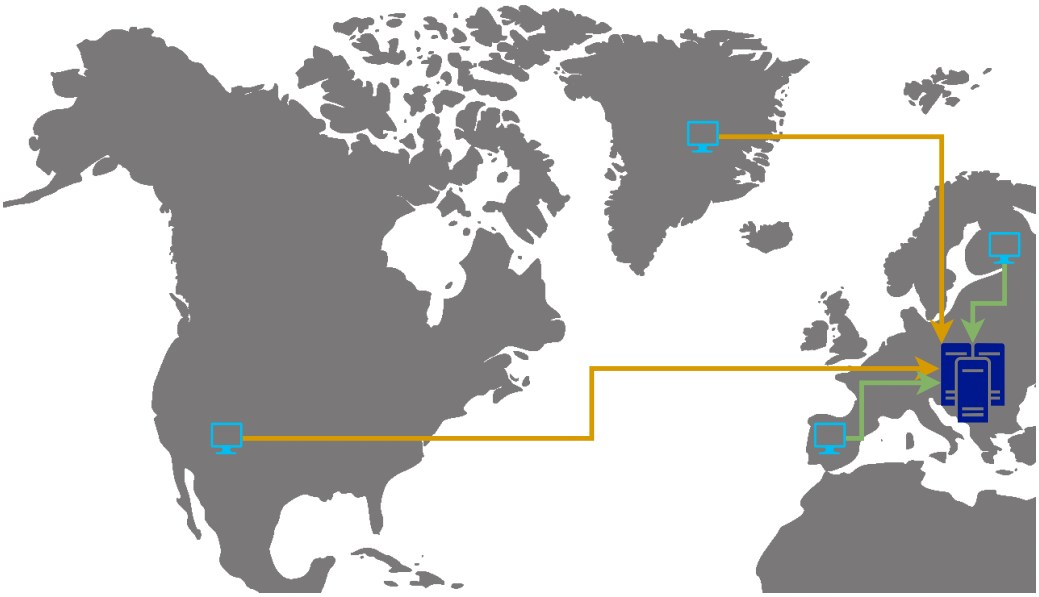
Leta 2014 je Google objavil kodo sistema za orkestracijo kontejnerjev Kubernetes [47] [1]. Kubernetes je univerzalni način, ki omogoča, da več računalnikov povežemo v gručo, ki deluje kot samostojna enota. Povezovanje strežnikov v gruče omogoča visoko razpoložljivost (HA) naših storitev [8], deljenje virov in preprostejšo sinhronizacijo podatkov. 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. Poskusili so ga rešiti s projektom Federation 1, po njegovi ukinitvi [15] pa razvijalci Federation 2 oziroma KubeFed trdijo, da bo projekt uspešno prešel iz alfa v beta verzijo [40].

1.2 Cilj in vsebina naloge

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

1.2.1 Prevelika latenca

Problem prevelike latence se pojavi, ko je čas od poslanega zahtevka uporabnika do prejema odgovora s strežnika prevelik [24]. Ko problem povzroča velika fizična razdalja, ga rešimo tako, da postavimo dodatne gruče bližje naših uporabnikov, denimo na njihovo celino. Zavedati se moramo, da strežniki v gruči zelo veliko komunicirajo, zato je priporočljivo, da se nahajajo v istem omrežju znotraj istega podatkovnega centra, saj se s tem izognemo veliki latenci [25].



Slika 1.1: Problem prevelike latence.

1.2.2 Visoka razpoložljivost

Večkrat letno pride do izpada kakšnega večjega podatkovnega centra [10]. To se lahko zgodi iz več razlogov najpogosteje pa gre za napake na programski opremi [19]. Če gre v takšnem primeru za oblačnega ponudnika, pri katerem imamo nameščeno svojo gručo, to 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 svoje uporabnike preusmerimo v drug podatkovni center.

1.2.3 Potreba po izolaciji aplikacij

Ko govorimo o izolaciji aplikacije, se navezujemo na varnost pri vdoru ali pa na večjo razpoložljivost. Uporaba Kubernetesa od nas zahteva izolacijo v kontejnerje. Prav tako pa nam že sam Kubernetes omogoča izolacijo na posamezne strežnike [42] ali nastavitev pravil komunikacije v gruči [37]. Kljub tem postopkom se v Kubernetesu pojavljajo problemi, zaradi katerih postane nedosegljiva celotna gruča, s tem pa vse aplikacije v tej gruči. Če postavimo del neodvisnih aplikacij v drugo gručo, s tem preprečimo njihov izpad ob napaki v prvi gruči. Izolacija pa je pomembna tudi z varnostnega vidika. Če se napadalec polasti enega samega vozlišča, ima posledično tudi popolno kontrolo nad vsemi drugimi aplikacijami, ki tečejo na tem vozlišču [16]. Aplikacije lahko pripadajo istemu uporabniku ali pa celo drugim uporabnikom. Če imamo vsako aplikacijo v svoji gruči, pa se temu lahko izognemo.

Poglavje 2

Problem povezovanja gruč

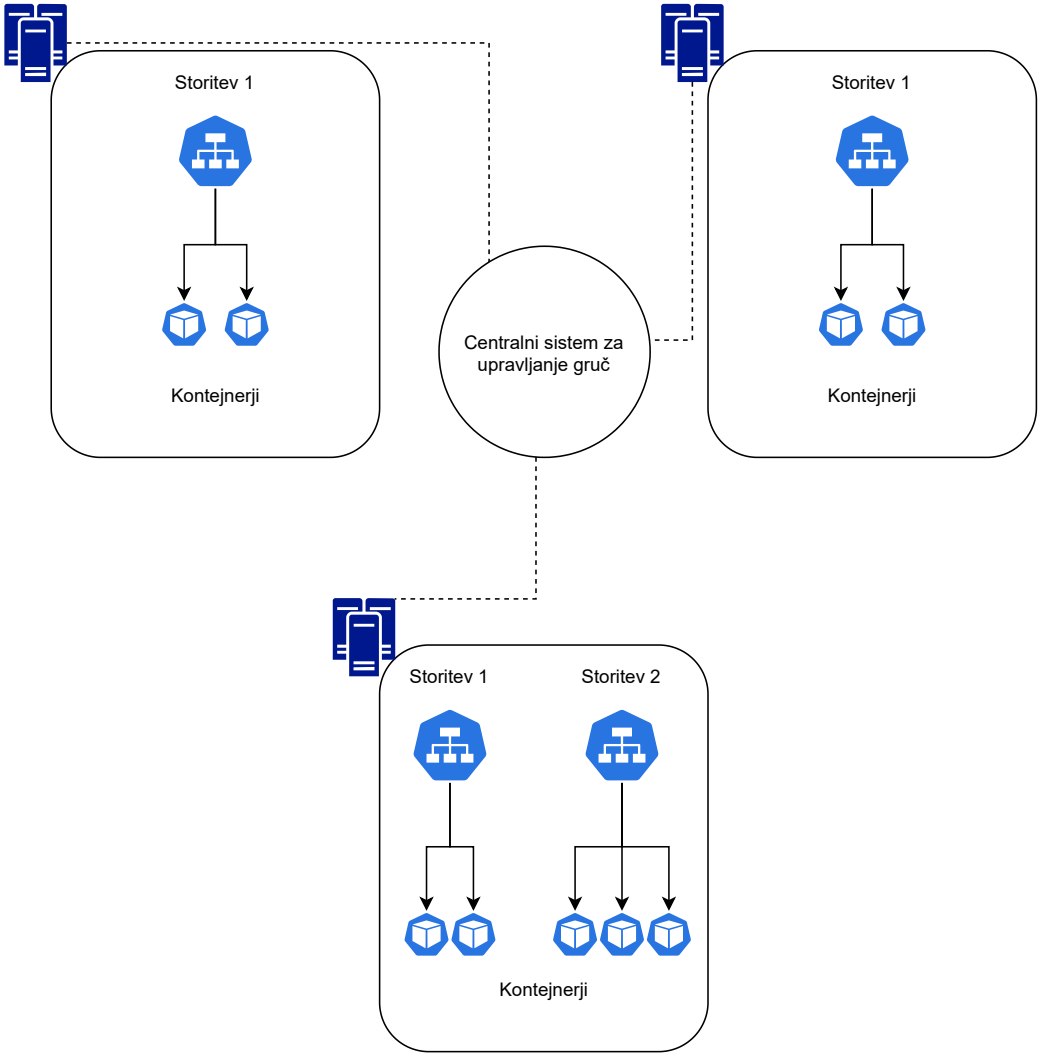
Računalniška gruča je skupina računalnikov, ki zaradi večje zanesljivosti in zmogljivosti opravlja določene storitve.

Zaradi prevelike latence ali drugih ovir računalnikov ne moremo povezati v eno tesno povezano gručo [25]. Računalnike lahko vedno povežemo v več različnih gruč, četudi to pomeni, da je v nekaterih gručah samo po en strežnik oziroma vozlišče. Ob prisotnosti povezave pa lahko te gruče med seboj povežemo, a šibkeje.

Ko govorimo o tesni povezanosti znotraj gruče, velja, da ima vsako vozlišče dostop do vsakega, da vsak kontejner lahko komunicira z vsakim in da so vozlišča v istem hitrem notranjem omrežju podatkovnega centra. Pričakujemo, da sistem, ki ga uporabljamo za gručenje, omogoča razporejanje zaželenih storitev in kontejnerjev med strežniki in v primeru izpada vozlišča to odstrani iz sistema, kontejnerje s tega vozlišča pa prerazporedi na preostala vozlišča.

Šibka povezanost med gručami pomeni, da je povezava med različnimi gručami počasna, nezanesljiva ali draga. Zaradi omejitev moramo sprejemati kompromise na podlagi zmožnosti povezave. Skladno z našimi potrebami se lahko odrečemo komunikaciji med vozlišči v različnih gručah. Pričakujemo, da vsaka gruča skrbi za svoja vozlišča ter svoje storitve in kontejnerje ohranja v delovanju. V tem delu bomo kot naloge sistemov za povezovanje gruč

obravnavali: omogočanje centralnega nadzora 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 več povezanih gruč.

Poglavje 3

Kubernetes

Kubernetes definira javno dostopen vmesnik REST. Trenutno obstaja že več kot 70 distribucij Kubernetesa [26].

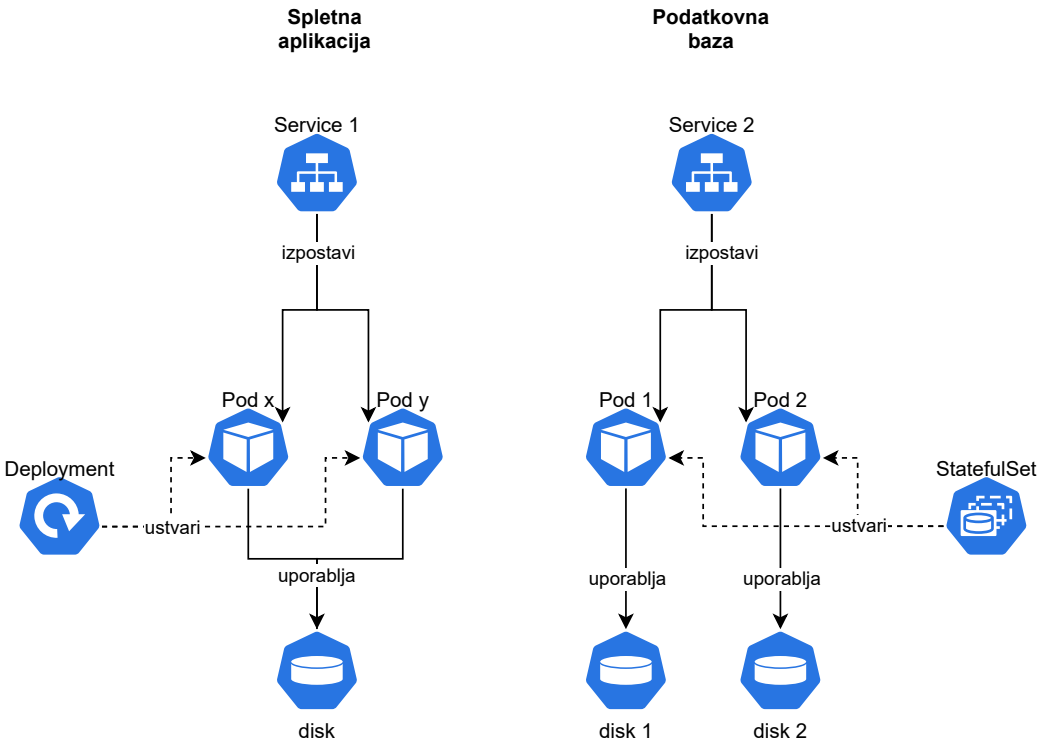
3.1 Zgodovina

Leta 2014 je Google objavil in odprl kodo projekta Kubernetes [8] [1]. Gre za program, ki je bil ustvarjen z namenom, da poenostavi upravljanje kontejnerjev in večjih računalniških gruč v produkcijskih okoljih [47]. A to niso pravi začetki Kubernetesa. Začelo se je leta 2003, ko je Google začel z razvojem sistema za upravljanje svojih notranjih gruč Borg. Pozneje, leta 2013, je Google predstavil sistem Omega, leta 2014 pa je objavil odprtokodni projekt Kubernetes. Projekt je bil zasnovan na podlagi dobrih praks upravljanja s kontejnerji, ki so se jih pri Googlu naučili skozi dolga leta upravljanja velikega števila kontejnerjev v produkcijskem okolju. Pozneje je upravljanje nad projektom prevzela organizacija Cloud Native Computing Fundation.

3.2 Osnovni pojmi

Kubernetesov vmesnik REST nam omogoča, da v sistem shranjujemo najrazličnejše tipe objektov. Takšne, ki so del standardnega Kubernetesovega

vmesnika, 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 objektov Kubernetes.

3.2.1 Pod

Objekt Pod je najmanjša enota v Kubernetesu, ki lahko teče v gruči [43]. Sestavljen je iz enega ali več kontejnerjev, ki si delijo diske in omrežni vmesnik. To pomeni, da imajo skupen naslov IP in se obnašajo podobno kot izolirani procesi na istem računalniku.

3.2.2 Service

Objekt Service označuje vse objekte Pod ene storitve [44]. Kubernetes iz objekta v notranjem DNS ustvari domeno za storitev in dinamično porazdeljuje promet med našimi objekti Pod. Objekte Service uporabljamo tako, da namesto pošiljanja zahtevkov neposredno na naslov IP objekta Pod, zahtevke pošiljamo na ustvarjeno domensko ime storitve na primer z ukazom `curl ime-storitve`. Takšen zahtevek nato dobi eden izmed označenih objektov Pod v objektu Service.

3.2.3 Deployment

Deployment je objekt, ki mu podamo število želenih objektov Pod in predlogo za njihovo izdelavo [38]. Notranje storitve Kubernetesa nato zagotavljajo, da bo obstajalo toliko takšnih objektov tipa Pod, kot smo navedli v definiciji objekta Deployment. Takšno stanje se poskuša ohranjati tudi ob raznih težavah in izpadih vozlišč.

3.2.4 StatefulSet

Ta objekt je zelo podoben objektu Deployment, le da StatefulSet vsakemu objektu Pod dodeli unikatno številko [46]. Pod, ki se ustvari s to številko, ohranja diske, mrežni vmesnik, naslov IP in domensko ime. Pomembna razlika med objektoma Deployment in StatefulSet pa je tudi v polju `volume ClaimTemplate`. StatefulSet omogoča vsakemu objektu Pod, da ustvari in uporablja svoj disk. Uporablja pa se za podatkovne baze in podobne storitve, ki morajo ohranjati stanja.

Poglavje 4

Pregled področja in literature

V tem poglavju bomo predstavili nekaj ključnih del in literature s področja povezovanja gruč Kubernetes. V delih sta pogosto za federacijo izbrana sistema Federation 1 ali Federation 2 zaradi tesne povezanosti s sistemom Kubernetes [18] [22] [25].

V članku [18] se avtorji lotijo povezovanja gruč pri različnih oblačnih ponudnikih. Pri tem pozornost namenijo tudi avtomatskemu horizontalnemu skaliranju aplikacij. Za uporabo in postavitev pri več oblačnih ponudnikih so uporabili standard TOSCA, ki jim omogoča enoten deklarativni zapis 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. Svoje gruče so povezali v federacijo s sistemom Federation. Iz članka pa ni povsem razvidno ali so uporabili prvo ali drugo iteracijo tega sistema. V testne namene so v federacijo namestili še strežnik spletne igre in pokazali uspešnost avtomatskega horizontalnega skaliranja.

Lorenzo Martino je v magistrski nalogi [22] 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 sistema Kubernetes v oblaku ali pa v lokalnem podatkovnem centru. Omenil je tudi hibridne rešitve, ki pa zahtevajo povezovanje in upravljanje več 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 avtomatskega skaliranja vozlišč, omejitve lokacije obdelovanja podatkov. V delu je predlaganih tudi nekaj programov za upravljanje gruč, v rešitvi problema pa je avtor 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 je bila obdelava podatkov v lokalnih gručah. Nadaljevanje dela je vezano na reševanje konkretnega problema podjetja. V delu je posebna pozornost namenjena sinhronizaciji podatkov, saj imajo v podjetju označene podatke, ki se ne smejo obdelovati v oblaku, in podatke, ki se lahko. KubeFed je še v razvojni fazi alfa, kar pa je za podjetje predstavljalo oviro. Tako je avtor poleg rešitve s KubeFed pripravil še svojo rešitev, v kateri implementira samo potrebne funkcionalnosti.

Vir [25] pa se posveti področju upravljanja aplikacij na robu oblaka, kjer pride centralno upravljanje zaradi večjega števila gruč še bolj do izraza. V poročilu je gruča Kubernetes postavljena v prostrano omrežje (WAN). Avtorji so primerjali delovanje ene gruče preko prostranega omrežja z delovanjem iste gruče preko lokalnega omrežja. Pri takšnem pristopu v gručah na robu oblaka izpostavijo pomembnost previdnosti, saj lahko pride do nepredvidljivih rezultatov. V poročilu je predstavljen tudi odprtokodni sistem KubeEdge. Projekt je namenjen razširitvi aplikacij v kontejnerjih na vozlišča na robu oblaka [34]. Avtorji izpostavijo, da imata tako pristop z eno gručo v omrežju WAN kot pristop s sistemom KubeEdge pomembno omejitev, saj imajo vozlišča v primeru izpada iz omrežja še vedno premalo avtonomnosti. Izpostavljeno je, da te slabosti rešimo s federacijo in sistemom KubeFed, ki je v nadaljevanju podrobneje opisan. Če je vsako vozlišče v svoji gruči, potem je vozlišče v primeru izpada omrežja še vedno avtonomno in omogoča lokalno upravljanje.

Poglavje 5

Povezovanje gruč Kubernetes

Ko postavimo več različnih gruč, imamo vedno možnost, da upravljamo vsako posebej. Toda takšen pristop ni učinkovit, če imamo takšnih gruč res veliko [27]. Osnovna lastnost sistema za upravljanje več gruč Kubernetes je možnost prenašanja Kubernetesovih objektov med gručami. Tako lahko objekt definiramo samo enkrat in naš sistem ga bo ustvaril v izbranih gručah. Glede na potrebe pa lahko uporabimo sistem, ki omogoča tudi dinamično odkrivanje storitev z enako definicijo v različnih gručah, komunikacijo med storitvami v različnih gručah, dinamično odkrivanje objektov Pod med gručami in komunikacijo med njimi v različnih gručah. Te funkcionalnosti znotraj ene gruče nudi že Kubernetes. Od našega primera pa je odvisno, katere funkcionalnosti želimo uporabiti in kako kompleksno postavitev potrebujemo. V nadaljevanju bomo predstavili različne sisteme za povezovanje gruč Kubernetes, njihove glavne prednosti in značilnosti.

5.1 ArgoCD in drugi sistemi GitOps

5.1.1 Sinhronizacija objektov s sistemi GitOps

Osnovna ideja pristopa GitOps je, da imamo strukturo aplikacij v gruči Kubernetes napisano v repozitoriju Git in potem je kontroler GitOps tisti, ki iz teh definicij postavi strukturo gruče. Takšen pristop se je v zadnjih letih

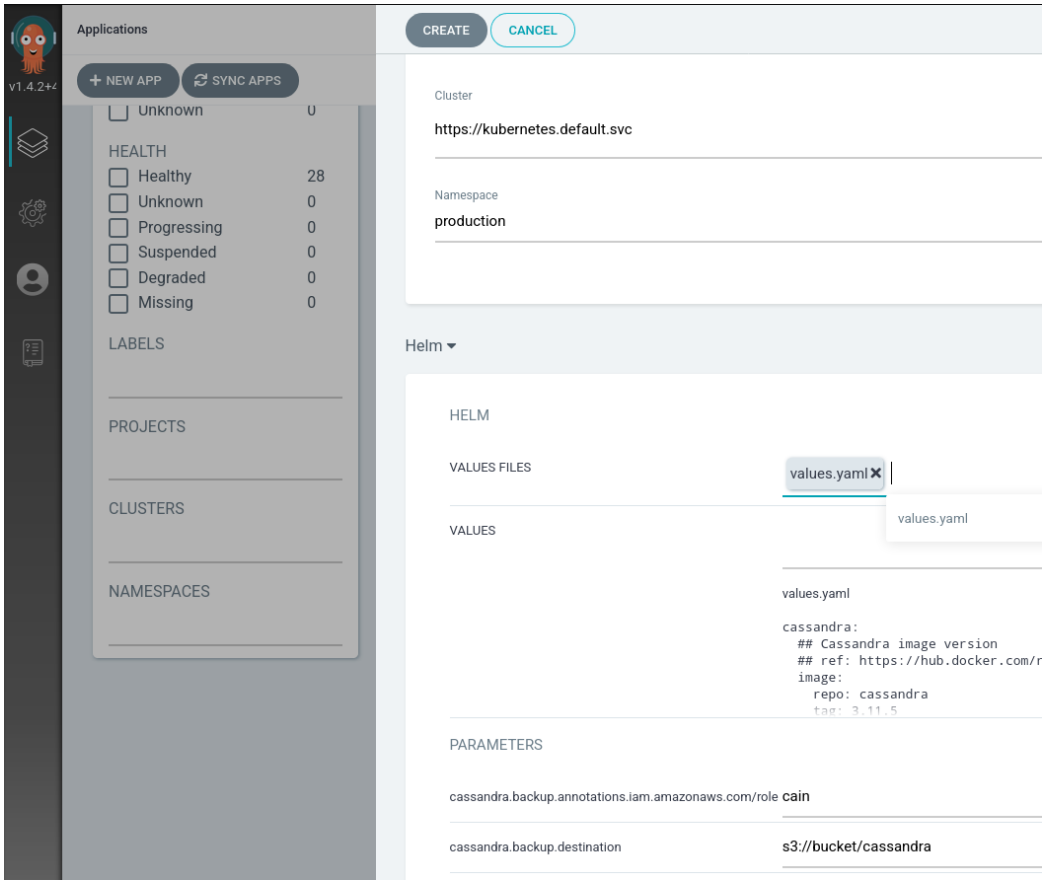
zelo razširil in ArgoCD navaja več kot sto podjetij, ki ga uporabljajo pri razvoju [30].

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

5.1.2 Sinhronizacija objektov z ArgoCD

ArgoCD podpira več različnih formatov konfiguracije gruče [28]. Uporabimo lahko format datoteke YAML z definicijami objektov, ki jih želimo namestiti v vsako gručo. Za nameščanje te konfiguracije v več kot eno gručo imamo na voljo dva pristopa. Prvi je, da v vsako gručo namestimo ArgoCD in uporabimo enak repozitorij Git v vsaki izmed njih. ArgoCD pa nam omogoča tudi pošiljanje konfiguracije v oddaljene gruče [29]. To pomeni, da imamo lahko kontroler ArgoCD nameščen samo v eni gruči.

Če ne želimo, da imajo vse gruče popolnoma enako infrastrukturo in nameravamo prilagoditi konfiguracijo posamezne gruče, lahko uporabimo format zapisa konfiguracije, ki podpira predloge. ArgoCD nam ponuja možnost, da v predlogah sami določimo vrednosti spremenljivk. Tako lahko uporabimo na primer predloge Helm in ArgoCD nam bo omogočil, da vsaki gruči izberemo lastno 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 in komunikacija kontejnerjev med gručami. Takšen sistem nam omogoča samo sinhronizacijo infrastrukture.



Slika 5.1: Primer uporabe predloge Helm v ArgoCD.

5.2 KubeFed

9. 1. 2018 je bil po ukinjenem projektu Kubernetes Federation V1 ustvarjen Kubernetes Federation V2, imenovan tudi KubeFed [15]. Cilj obeh projektov je bil poenostavljeno upravljanje več gruč in razporejanje Kubernetes objektov. V projektu Federation V1 je bil ubran pristop, ki je skupino gruč ali federacijo uporabniku predstavil kot novo gručo Kubernetes [45]. Uporabljal je svoj vmesnik, ki pa je bil združljiv s Kubernetesovim, kar je omogočalo tudi uporabo orodja kubectl [11]. Objekti, ki jih je federacija podpirala, so bili kompatibilni s standardnimi objekti Kubernetes [39]. Objekte, ki so bili poslani kontrolerju federacije, je Federation V1 nato ustvaril tudi v

pripadajočih gručah. Pristop zaradi mnogih pomanjkljivosti in pomankanja možnosti naprednejših konfiguracij ni uspel pridobiti statusa GA. Faza GA 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 pozneje razvil projekt Federation V2 [15]. Glavna razlika s prvo različico je z uporabniškega stališča v tem, da za federacijo ne poizkuša imitirati Kubernetesovega vmesnika, ampak uporablja obstoječega. Federation V2 samo predstavi nove objekte, ki pa so razširitev standardnih, kot na primer FederatedDeployment [41]. Federirane objekte je treba najprej vklopiti z ukazom `kubefedctl enable`.

```
kubefedctl enable deployment
```

Orodje kubefedctl moramo namestiti na svoj računalnik. Takšen federiran 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 objektov CRD.

```
apiVersion: types.kubefed.io/v1beta1
kind: FederatedDeployment
spec:
  placement:
    clusterSelector:
      # izbira gruč
      matchLabels: {}
      ...
  template:
    # specifikacije objekta deployment
    spec:
    ...
  overrides:
```

```
    # prepis konfiguracije za posamezne gruče
    - clusterName: gruca-1
      clusterOverrides:
          # nastavi polje replicas na vrednost 5
        - path: "/spec/replicas"
          value: 3
...
```

Federation V2 poleg sinhronizacije infrastrukture podpira tudi odkrivanje storitev v drugih gručah prek zapisov DNS [41]. Omenja pa se možnost odstranitve te funkcionalnosti [36], ki je že sedaj privzeto izklopljena. Preden uporabimo KubeFed pa se moramo zavedati, da je projekt v času pisanja diplomske naloge še vedno v razvojni fazi alfa in lahko preteče še nekaj časa preden doseže status GA.

5.3 Cilium

Cilium je odprtokodni program, ki nam omogoča napredne varnostne in omrežne nastavitve v gruči [31]. Program na tretji in četrti omrežni plasti zagotavlja osnovne principe varnosti in zaščite, kot sta, na primer, zapiranje portov in omejevanje komunikacije. Poleg tega pa Cilium zagotavlja tudi naprednejšo varnost na sedmi omrežni plasti, saj omogoča omejevanje in filtriranje zahtevkov HTTP in podobne varnostne funkcionalnosti na popularnih protokolih aplikacijskega nivoja [31].

Ker Cilium implementira precejšen del mreženja in povezovanja v Kubernetesu, pa nam s tem lahko ponudi tudi nekaj zelo naprednih možnosti, ko med seboj povezujemo več različnih gruč Kubernetes. Tako nam Cilium kot ključni prednosti omogoča tudi komunikacijo med kontejnerji v različnih gručah [32] in uporabo globalnih objektov Service, ki razporejajo promet med različnimi gručami. Takšne objekte definiramo z anotacijo `io.cilium/global-service` [33]. Omogoča nam tudi omejevanje povezovanja med gručami z njihovim objektom CiliumNetworkPolicy [33]. Ko po-

stavljamo mrežo gruč, pa se moramo še vedno zavedati, da Cilium ne rešuje problema, če so naše gruče skrite v različnih zasebnih omrežjih. Ključno pri uporabi Ciliuma za povezovanje gruč je, da so vsa naša vozlišča med seboj dosegljiva. A četudi so naše gruče v med seboj direktno nedosegljivih zasebnih omrežjih, je problem rešljiv z uporabo sistema VPN, ki omogoča, da vsa vozlišča povežemo v eno navidezno omrežje [33].

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

Poglavje 6

Priprava sistema gruč za testiranje

6.1 Raspberry PI 4

Za namene testiranja različnih načinov povezovanja gruč Kubernetes moramo najprej postaviti nekaj gruč. Zaradi preprostosti in nizke cene, predvsem pa, ker se koncepti zaradi tega ne spremenijo, bomo za naša vozlišča Kubernetes uporabili Raspberry PI 4. Na višjem nivoju gre še vedno za gručo Kubernetes in delo je zelo podobno, če uporabimo nekaj tisoč vozlišč v gruči v oblaku ali pa lokalno gručo z enim vozliščem. Raspberry PI 4 je majhen (85x56x20mm) in manj zmogljiv računalnik na eni sami plošči [23]. Ključni prednosti takšnih računalnikov sta velikost in cena. Na vsak Raspberry PI se bo namestila gruča Kubernetes z enim samim vozliščem. Fizična postavitev gruč je prikazana na sliki 6.1. Takšna postavitev pa je lahko tudi primer gruče na robu oblaka.

6.2 K3S in K3OS

Obstaja več implementacij Kubernetesa. Mi bomo uporabili z viri varčno odprtokodno implementacijo K3S od podjetja Rancher [21] [7] [9]. Hkrati

Slika 6.1: Postavitev gruč Raspberry PI.

so v podjetju Rancher pripravili distribucijo operacijskega sistema Linux K3OS [20]. Gre za minimalen operacijski sistem s prednameščenim sistemom K3S. Težava se pojavi, ker uradna verzija operacijskega sistema za ploščice Raspberry PI še ni pripravljena. K sreči pa se je v ta namen začel odprtokodni projekt *PiCl k3os image generator*, ki iz slik operacijskih sistemov K3OS in Raspberry OS ter konfiguracijskih datotek zgradi novo sliko operacijskega sistema za naš Raspberry PI [5]. Konfiguracijske datoteke, ki jih moramo priložiti so standardne datoteke YAML, ki jih podpira sistem K3OS. Vanje zapišemo nastavitve kot so javni ključi za dostop SSH, podatki omrežja Wi-Fi na katerega se povezujemo, geslo, žeton za povezavo z gručo Kubernetes in način, v katerem želimo zagnati K3S na sistemu [20]. V našem primeru smo vse programe K3S zagnali v strežniškem načinu (server) 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 treba narediti novo testno storitev. Ker se želimo v tem diplomskem delu osredotočiti na realne probleme, s katerimi se srečujejo podjetja, mora ta aplikacija omogočati tudi shranjevanje podatkov v podatkovno bazo.

Koda, ki je javno objavljena v repozitoriju Git [13], je napisana v programskem jeziku Go. Iz kode je bil generiran kontejner, ki je objavljen v javnem registru Docker [14]. Ob tem velja opozoriti, da Raspberry PI uporablja arhitekturo procesorja ARM, kar je zahtevalo dodatno pozornost pri nastavitvah gradnje ustrezne slike Docker.

Aplikacija na mrežnih vratih, podanih s spremenljivko okolja, izpostavi vmesnik REST, ki podpira dva preprosta klica HTTP. Klic `GET` na pot `/users`

vrne seznam vseh uporabnikov, ki so zapisani v tabeli v podatkovni bazi, s klicem `POST` na isto pot pa poskrbimo, da se podatki uporabnika iz našega zahtevka shranijo v tabelo v podatkovni bazi.

```
# 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 dve različni bazi podatkov SQL: Postgres, ki je preprosta za lokalni razvoj, a ne omogoča napredne sinhronizacije podatkov med strežniki in CrateDB, ki je bil zasnovan kot baza SQL na več vozliščih in nam omogoča napredne sinhronizacije tudi med različnimi strežniki in gručami. K sreči pa CrateDB implementira vmesnik PostgreSQL in nam kode za prehod med bazami ni potrebno spreminjati [4].

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, še ni objavljene uradne verzije programa za arhitekturo ARM. Zato je bilo iz kode KubeFed treba zgraditi novo sliko kontejnerja, ki je javno objavljena [35]. Potem pa smo uporabili uradno predlogo Helm, pri čemer smo samo zamenjali originalno sliko kontejnerja z našo. Za delo s KubeFed pa moramo na svoj računalnik namestiti še orodje Kubefedctl. Z uporabo ukaza `kubefedctl join` povežemo vse tri gruče v sistem KubeFed.

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

S tem smo uspešno povezali več gruč Kubernetes 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.

Poglavje 7

Povezovanje med podatkovnimi centri

7.1 Problem velike latence

Za primer vzemimo preprosto spletno aplikacijo, ki mora hraniti stanje in jo namestimo v eno gručo Kubernetes. Če to aplikacijo ponudimo vsem uporabnikom na globalnem trgu, se nam bo pojavil problem velike latence [24]. To pomeni, da bo naša aplikacija za uporabnike, ki so bolj oddaljeni od naše gruče, delovala počasneje oziroma se bodo ob enaki pasovni širini podatki do uporabnika prenašali dalj časa.

Takšen problem v splošnem rešimo tako, da našo aplikacijo postavimo še v dodatno gručo, bližje uporabniku [49]. Če moramo podatke med gručami še sinhronizirati, pa to zahteva dodaten trud. V našem primeru bomo uporabili podatkovno bazo CrateDB [4], novejšo alternativo standardnim podatkovnim bazam SQL. CrateDB ima v primerjavi s tradicionalnimi podatkovnimi bazami boljšo podporo za sinhronizacijo podatkov med vozlišči [17]. Za uporabo podatkovne baze CrateDB ni treba niti konceptualno spreminjati naše aplikacije, saj podpira vmesnik podatkovne baze PostgreSQL.

7.2 Povečanje razpoložljivosti aplikacije

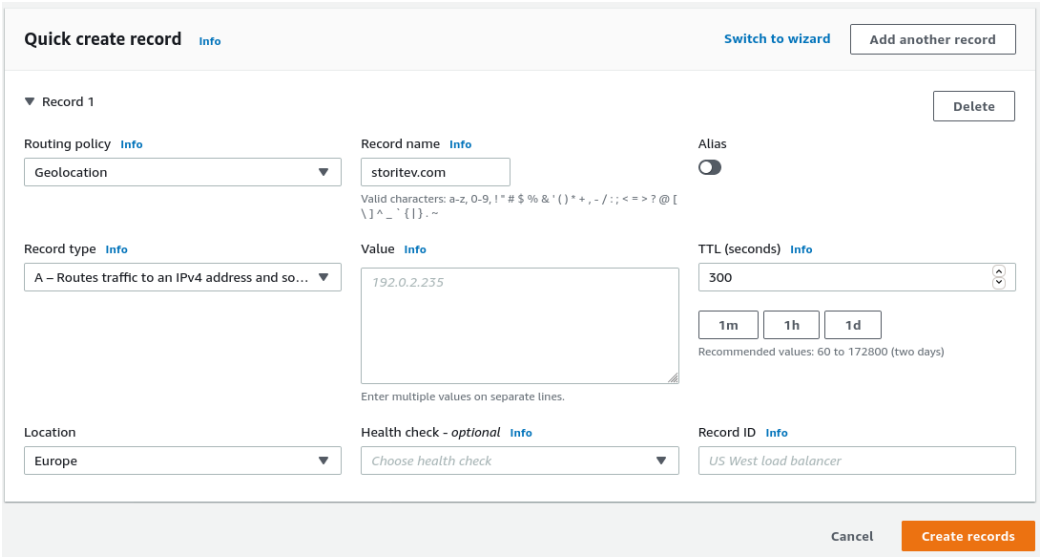
Če je čim višja razpoložljivost za našo aplikacijo kritičnega pomena in smo že poskrbeli za visoko razpoložljivost (HA) aplikacije v naši gruči, še vedno lahko pride do situacije, ko iz omrežja izpade ves podatkovni center [6]. Spomnimo se, da Kubernetes najbolj učinkovito deluje, če naša vozlišča uporabljajo hitro notranje omrežje podatkovnega centra. V primeru napake v podatkovnem centru ali hujših vremenskih pogojev, ki bi prekinili povezave do podatkovnega centra, to pomeni, da je nedosegljiv ves podatkovni center in s tem gruča v njem. Pri uporabi strežnikov v oblaku lahko še povečamo razpoložljivost tako, da gruče namestimo v različne razpoložljivostne cone (availability zones) in podatkovne centre pri enem ali celo več različnih oblačnih ponudnikih. Takšen pristop je opisan v članku [18], omogoča pa ga predvsem neodvisnost Kubernetesa od platform.

7.3 Povezovanje med podatkovnimi centri

Rešitev za oba omenjena problema je enaka. Postaviti moramo gruče v več različnih podatkovnih centrih in jih nastaviti, da bodo delovale usklajeno. Ti podatkovni centri bodo bližje uporabniku ali pa v lasti različnih oblačnih ponudnikov – odvisno od problema. Princip pa ostaja enak.

7.4 Razporeditev uporabnikov po gručah

Ko imamo v vsaki gruči javno izpostavljen Kubernetesov objekt Service in postavljene primerne objekte Ingress, moramo uporabnike še vedno usmeriti na njim najbližjo gručo. Lahko jih usmerimo avtomatsko z zapisi DNS, 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 uporabnikov ne bomo usmerjali preko javnih strežnikov DNS, saj v lokalnem okolju to ni smiselno.

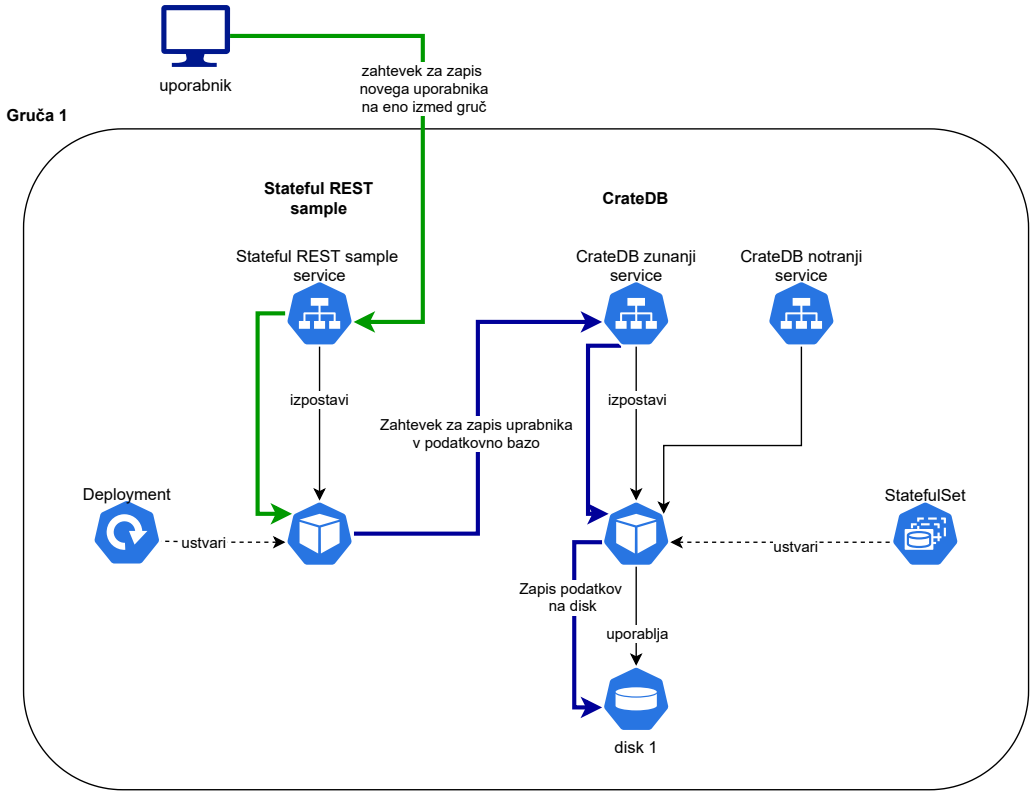


Slika 7.1: Ustvarjanje geolokacijskega zapisa DNS v storitvi Route53.

Naslednja možnost je rešitev, ki se je poslužujejo nekatere internetne računalniške igre (npr. Among Us), in sicer, da so naši strežniki popolnoma ločeni in se vsak uporabnik sam odloči, na kateri gruči ali strežniku želi igrati. V takšnih primerih se lahko 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 objektom Deployment. Da 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 kartici SD, 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 repozitoriju Git[12]. Postavimo jo z ukazom `kubectl apply -f diploma-demo-1`. Ta-



Slika 7.2: Infrastruktura vsake gruče v primeru demonstracijske aplikacije

koj preverimo, če aplikacija deluje in če lahko podatke zapisujemo v bazo. To storimo tako, da prek demonstracijske aplikacije poskusimo dodati uporabnika in izpisati vse uporabnike. To naredimo z naslednjima ukazoma `curl`.

```
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 federirane tipe objektov na izbranih tipih.

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

Ko vklopimo federacijo na vseh potrebnih tipih, pa moramo vklopiti še 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 federirane objekte, ki uporabijo postavitev v vse gruče in za predlogo kar podane objekte. Tako je za nas rezultat izvršenih ukazov kreiranje federiranih objektov in posledično kopiranje objektov v vse naše povezane gruče.

Po preizkusu delovanja z ukazom `curl` 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 zapisov DNS urediti tudi avtomatsko sinhronizacijo podatkov. V našem primeru res uporabljamo samo en primerek baze CrateDB na gručo, a vseeno smo na nivoju sinhronizacije znotraj gruče to že uredili. Moramo se zavedati, da tudi podatkovna gruča CrateDB najbolje deluje, če so vozlišča v hitrem lokalnem omrežju z visoko pasovno širino in nizko latenco. CrateDB podpira tudi sinhronizacijo med različnimi razpoložljivostnimi conami in podatkovnimi centri [3].

7.7.1 Uporaba primerne podatkovne baze

Za sinhronizacijo podatkov lahko uporabimo podatkovno bazo, ki ima sinhronizacijo med različnimi gručami že podprto. CrateDB podpira sinhronizacijo tudi preko razpoložljivostnih con. Vseeno pa moramo vsa vozlišča povezati v enako podatkovno gručo [3]. To pomeni, da morajo biti primerki CrateDB dostopni med seboj. Problem lahko rešimo z uporabo sistema Cilium in uporabo globalnih storitev, saj nam Cilium že omogoča komunikacijo vsakega kontejnerja z vsakim, tudi če so ti v različnih gručah. Druga možnost pa je, da izpostavimo vsak objekt Pod s svojim javnim naslovom IP in jih v gručo povežemo ročno.

Določiti moramo še nastavitve za zmanjšanje prometa med podatkovnimi centri in nastavitve za zagotavljanje prisotnosti podatkov v vsakem podatkovnem centru [3]. Podobne načine sinhronizacije podpira tudi na primer podatkovna baza Cassandra [2].

7.7.2 Ročno usklajevanje podatkov

Sinhronizacija podatkovne baze ni trivialen problem. Če ne uporabimo primerne podatkovne baze ali pa preko gruč želimo sinhronizirati samo del podatkov, bomo sinhronizacijo podatkov verjetno morali implementirati sami. To pomeni, da bomo ustvarili novo storitev, ki bo v ozadju kopirala ključne podatke med podatkovnimi centri. Ker samo mi poznamo naš konkreten primer uporabe, je takšen pristop lahko najbolj učinkovit, a tudi najbolj kompleksen in časovno potraten.

V našem primeru bomo s preprosto skripto kopirali uporabnike iz ene aplikacije v drugo kar z uporabo vmesnika REST. To bomo storili v drugem kontejnerju Ubuntu z uporabo ukazov `curl` za izvajanje klicev REST in ukazom `jq` [48] za razčlenjevanje podatkov. Podatki se sinhronizirajo vsakih deset sekund. Primer še testiramo in dobimo spodnji izhod, kar potrdi, da so se podatki uspešno sinhronizirali.

```
curl -s -X POST gruca-1/users \
```

```
  --data '{"name": "John", "lastname": "Doe"}'
curl -s gruca-2/users
[{"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 [19]. Vseeno pa lahko kljub vdorom in napakam, zaradi katerih postanejo naši sistemi nedosegljivi, vedno poskusimo zmanjšati posledice ob morebitnem vdoru ali izpadu.

8.1.1 Izpadi aplikacije

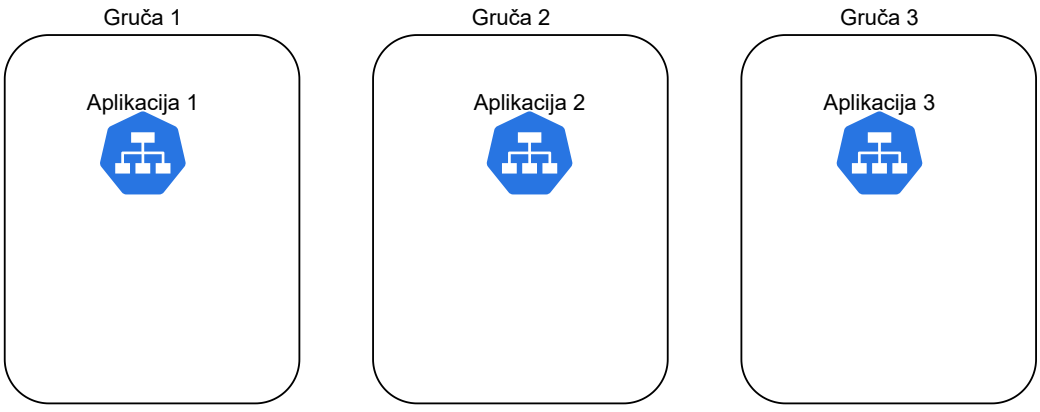
Kljub temu, da smo naše aplikacije namestili na različne gruče, s čimer je aplikacija odporna na izpad ene gruče, pa lahko ob hujših nepravilnostih delovanja ene aplikacije in napaki pri nastavitvi gruč kaskadno izpadejo tudi vse gruče, na katerih imamo aplikacijo nameščeno. Takšen primer je, ko ena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

moramo zavedati, da je ta korak smiseln šele po tem, ko opravimo že vse predhodne preventivne ukrepe, kot so razdelitev aplikacije na mikrostoritve, kontejnerizacija, pravilna nastavitev omejitev skaliranja aplikacije in druge.

8.1.2 Vdori

Podobno kot pri izpadih aplikacije je tudi pri preprečevanju posledic vdorov. Najprej moramo poskrbeti za primerno zaščito vozlišč Kubernetes, naše aplikacije, zaščito komunikacije med storitvami, uporabo nepriviligiranih kontejnerjev [16]. 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 njih ne sme dostopati do podatkov kritičnih aplikacij, potem 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 konfiguracijo federiranih objektov spisati 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 treba posebej ukvarjati s sinhronizacijo podatkov, saj smo vse podatke obdržali v isti gruči ali pa sinhronizirali na enak način kot v poglavju 7.

Poglavje 9

Upravljanje gruč na robu oblaka

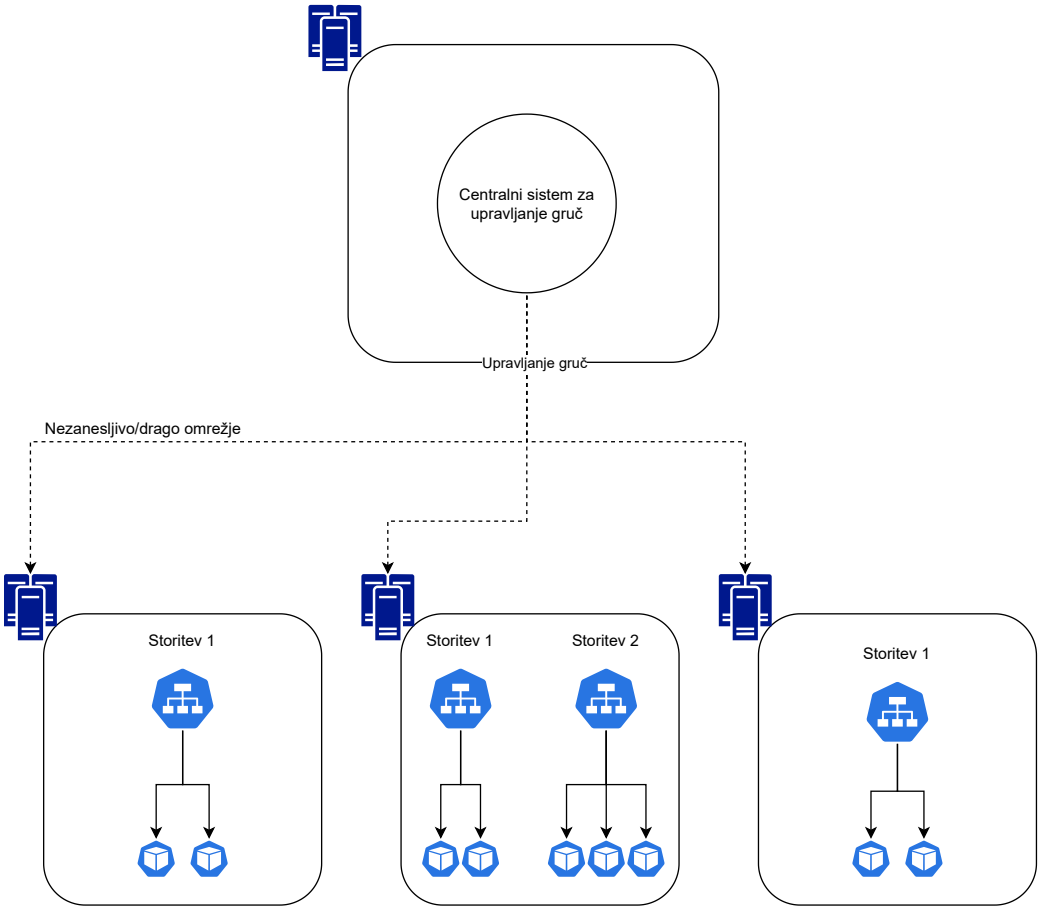
9.1 Gruče na robu oblaka

Razlogov, zakaj gruče postavljamo na rob oblaka oziroma fizično bližje končnemu uporabniku, je več. Za primer vzemimo zahtevo podjetja, da se morajo njihovi podatki obdelovati lokalno – v njihovem podjetju. V našem primeru se bomo osredotočili na upravljanje takšnih gruč.

9.2 Implementacija s KubeFed

Ko enkrat povežemo vse gruče z ukazom `kubefedctl join`, je njihovo upravljanje preprosto. Samo nastavimo v kateri gruči želimo katere objekte, in naša naloga je končana. Zavedati se moramo, da nekaj prenosa podatkov porabi tudi KubeFed za sinhronizacijo, zato moramo biti pozorni, če se podatki prenašajo prek dragih mobilnih omrežij.

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



Slika 9.1: Primer upravljanja gruč na robu oblaka.

z objekti Kubernetes Job.

Z zelo preprosto integracijo v sistem Kubernetes nam vmesnik KubeFed tu 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 zaposlenim dovoljuje 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 velike količine 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 stroške podjetja. Vse te gruče na podružnicah bi imele zelo podobno strukturo in jih je smiselno centralo upravljati z nekakšnim sistemom za povezovanje. Tu bi lahko uporabili pristopa KubeFed ali pristop GitOps. Pošiljanje podatkov v centralno gručo pa bi morali implementirati sami in ga vgraditi v naš program za prepoznavo obrazov.

Poglavje 10

Sklepne ugotovitve

V diplomskem delu smo predstavili teoretično ozadje povezovanja več računalniških gruč Kubernetes. Predstavljenih je bilo tudi nekaj popularnih orodij za delo z njimi. V praktičnem delu smo se posvetili predvsem reševanju pogostih problemov, s katerimi se srečujejo podjetja v praksi, ki zahtevajo povezovanje več gruč. Razvili smo primer preproste aplikacije za shranjevanje uporabnikov, s pomočjo katere smo reševali problem povezovanja več gruč Kubernetes. Pokazali smo, da je ob uporabi primernih orodij upravljanje več gruč Kubernetes preprosto, bolj zahtevna pa je sinhronizacija podatkov med različnimi gručami, kar smo pokazali tudi na primeru. Če je možno, se je sinhronizaciji podatkov smiselno izogniti. Takšen pristop smo uporabili tudi v podjetju, kjer sem zaposlen.

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 problemov. 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.

Literatura

[1] Brendan Burns, Brian Grant, David Oppenheimer, Eric Brewer, and John Wilkes. Borg, omega, and kubernetes: Lessons learned from three container-management systems over a decade. *Queue*, 14(1):70–93, January 2016.

[2] Apache Cassandra. Initializing a multiple node cluster (multiple datacenters). Dosegljivo: `https://docs.datastax.com/en/cassandra-oss/2.2/cassandra/initialize/initMultipleDS.html`, 2020. [Dostopano: 29. 12. 2020].

[3] CrateDB. Cratedb multi-zone setup. Dosegljivo: `https://crate.io/docs/crate/howtos/en/latest/clustering/multi-zone-setup.html`, 2020. [Dostopano: 29. 12. 2020].

[4] CrateDB. Cratedb: the distributed sql database for iot and time-series data. Dosegljivo: `https://crate.io/`, 2020. [Dostopano: 29. 12. 2020].

[5] odprtokodna skupnost Dennis Brentjes, Sjors Gielen. Picl k3os image generator. Dosegljivo: `https://github.com/sgielen/picl-k3os-image-generator`, 2020. [Dostopano: 16. 12. 2020].

[6] P. T. Endo, G. L. Santos, D. Rosendo, D. M. Gomes, A. Moreira, J. Kelner, D. Sadok, G. E. Gonçalves, and M. Mahloo. Minimizing and managing cloud failures. *Computer*, 50(11):86–90, November 2017.

[7] Halim Fathoni, Chao-Tung Yang, Chih-Hung Chang, and Chin-Yin Huang. Performance comparison of lightweight kubernetes in edge devices.

In Christian Esposito, Jiman Hong, and Kim-Kwang Raymond Choo, editors, *Pervasive Systems, Algorithms and Networks*, pages 304–309, Cham, 2019. Springer International Publishing.

[8] Sayfan Gigi. *Mastering Kubernetes*. Packt Publishing Ltd, 2017.

[9] Tom Goethals, Filip De Turck, and Bruno Volckaert. Fledge: Kubernetes compatible container orchestration on low-resource edge devices. In Ching-Hsien Hsu, Sondès Kallel, Kun-Chan Lan, and Zibin Zheng, editors, *Internet of Vehicles. Technologies and Services Toward Smart Cities*, pages 174–189, Cham, 2020. Springer International Publishing.

[10] J. Gray and D. P. Siewiorek. High-availability computer systems. *Computer*, 24(9):39–48, 1991.

[11] Lukasz Guminski. Cluster federation in kubernetes 1.5. Dosegljivo: `https://kubernetes.io/blog/2016/12/cluster-federation-in-kubernetes-1-5/`. [Dostopano: 23. 11. 2020].

[12] Jakob Hostnik. Connecting kubernetes clusters. Dosegljivo: `https://github.com/hostops/connecting-kubernetes-clusters`, 2020. [Dostopano: 16. 12. 2020].

[13] Jakob Hostnik. Stateful rest sample. Dosegljivo: `https://github.com/hostops/stateful-rest-sample`, 2020. [Dostopano: 16. 12. 2020].

[14] Jakob Hostnik. Stateful rest sample. Dosegljivo: `https://hub.docker.com/r/hostops/stateful-rest-sample`, 2020. [Dostopano: 16. 12. 2020].

[15] Shashidhara T D Irfan Ur Rehman, Paul Morie. Kubernetes federation evolution. Dosegljivo: `https://kubernetes.io/blog/2018/12/12/kubernetes-federation-evolution/`, 2018. [Dostopano: 20. 12. 2020].

[16] M. S. Islam Shamim, F. Ahamed Bhuiyan, and A. Rahman. Xi commandments of kubernetes security: A systematization of knowledge related to kubernetes security practices. In *2020 IEEE Secure Development (SecDev)*, pages 58–64, 2020.

[17] Lauren Milechin Siddharth Samsi William Arcand David Bestor William Bergero Chansup Byun Matthew Hubbell Michael Houle Michael Jones Anne Klein Peter Michaleas Julie Mullen AProut Antonio Rosa Charles Yee Albert Reuther Jeremy Kepner, Vijay Gadepally. A billion updates per second using 30, 000 hierarchical in-memory D4M databases. *CoRR*, abs/1902.00846, 2019.

[18] Dongmin Kim, Hanif Muhammad, Eunsam Kim, Sumi Helal, and Choonhwa Lee. Tosca-based and federation-aware cloud orchestration for kubernetes container platform. *Applied Sciences*, 9(1), 2019.

[19] V. B. Mendiratta L. Fiondella, S. S. Gokhale. Cloud incident data: An empirical analysis. In *2013 IEEE International Conference on Cloud Engineering (IC2E)*, pages 241–249, 2013.

[20] Rancher Labs. K3os. Dosegljivo: `https://github.com/rancher/k3os`, 2020. [Dostopano: 16. 12. 2020].

[21] Rancher Labs. K3s: Lightweight kubernetes. Dosegljivo: `https://k3s.io/`, 2020. [Dostopano: 16. 11. 2020].

[22] Marino Lorenzo. Dynamic application placement in a kubernetes multicluster environment. Magisterska naloga, Politecnico di Torino, 2020.

[23] Raspberry Pi Trading Ltd. Raspberry pi 4 computer model b. Dosegljivo: `https://static.raspberrypi.org/files/product-briefs/200521+Raspberry+Pi+4+Product+Brief.pdf`. [Dostopano: 20. 1. 2021].

[24] Marzieh Malekimajd, Ali Movaghar, and Seyedmahyar Hosseinimotlagh. Minimizing latency in geo-distributed clouds. *The Journal of Supercomputing*, 71(12):4423–4445, Dec 2015.

[25] Karim Manaouil and Adrien Lebre. Kubernetes and the edge? Raziskovalno poročilo RR-9370, Inria Rennes - Bretagne Atlantique, 2020.

[26] Cloud native computing foundation. Cncf cloud native interactive landscape. Dosegljivo: `https://landscape.cncf.io/`, 2020. [Dostopano: 16. 12. 2020].

[27] Platform9. Difference between multi-cluster, multi-master, multi-tenant & federated kubernetes. Dosegljivo: `https://platform9.com/blog/difference-between-multi-cluster-multi-master-multi-tenant-federated-kubernetes/`. [Dostopano: 20. 11. 2020].

[28] ArgoCD skupnost. Argo cd - declarative gitops cd for kubernetes. Dosegljivo: `https://argoproj.github.io/argo-cd`. [Dostopano: 16. 12. 2020].

[29] ArgoCD skupnost. Declarative setup. Dosegljivo: `https://argoproj.github.io/argo-cd/operator-manual/declarative-setup`. [Dostopano: 16. 12. 2020].

[30] ArgoCD skupnost. Who uses argo cd? Dosegljivo: `https://github.com/argoproj/argo-cd/blob/master/USERS.md`. [Dostopano: 20. 1. 2021].

[31] Cilium skupnost. Introduction to cilium & hubble. Dosegljivo: `https://docs.cilium.io/en/latest/intro/`. [Dostopano: 3. 1. 2020].

[32] Cilium skupnost. Multi-cluster (cluster mesh). Dosegljivo: `https://docs.cilium.io/en/latest/concepts/clustermesh/`. [Dostopano: 3. 1. 2020].

[33] Cilium skupnost. Set up cluster mesh. Dosegljivo: https://docs.cilium.io/en/latest/gettingstarted/clustermesh/. [Dostopano: 3. 1. 2020].

[34] KubeEdge skupnost. Kubeedge. Dosegljivo: https://kubeedge.io. [Dostopano: 20. 1. 2021].

[35] Kubefed skupnost. Kubefed. Dosegljivo: https://hub.docker.com/r/hostops/kubefed, 2020. [Dostopano: 16. 12. 2020].

[36] KubeFed skupnost. kubefed: remove crossclusterservicediscovery feature. Dosegljivo: https://github.com/kubernetes-sigs/kubefed/issues/1283, 2020. [Dostopano: 16. 11. 2020].

[37] Kubernetes skupnost. Assigning pods to nodes. Dosegljivo: https://kubernetes.io/docs/concepts/scheduling-eviction/assign-pod-node/. [Dostopano: 20. 1. 2021].

[38] Kubernetes skupnost. Deployments. Dosegljivo: https://kubernetes.io/docs/concepts/workloads/controllers/deployment/. [Dostopano: 3. 1. 2021].

[39] Kubernetes skupnost. Federated cluster. Dosegljivo: https://v1-16.docs.kubernetes.io/docs/tasks/federation/administer-federation/cluster/. [Dostopano: 23. 11. 2020].

[40] Kubernetes skupnost. Kubefed repozitorij. Dosegljivo: https://github.com/kubernetes-sigs/kubefed. [Dostopano: 23. 11. 2020].

[41] Kubernetes skupnost. Kubefed user guide. Dosegljivo: https://github.com/kubernetes-sigs/kubefed/blob/master/docs/userguide.md. [Dostopano: 23. 11. 2020].

[42] Kubernetes skupnost. Network policies. Dosegljivo: https://kubernetes.io/docs/concepts/services-networking/network-policies/. [Dostopano: 20. 1. 2021].

[43] Kubernetes skupnost. Pods. Dosegljivo: https://kubernetes.io/docs/concepts/workloads/pods/. [Dostopano: 3. 1. 2021].

[44] Kubernetes skupnost. Service. Dosegljivo: https://kubernetes.io/docs/concepts/services-networking/service/. [Dostopano: 3. 1. 2021].

[45] Kubernetes skupnost. Set up cluster federation with kubefed. Dosegljivo: https://v1-16.docs.kubernetes.io/docs/tasks/federation/set-up-cluster-federation-kubefed/. [Dostopano: 23. 11. 2020].

[46] Kubernetes skupnost. Statefulsets. Dosegljivo: https://kubernetes.io/docs/concepts/workloads/controllers/statefulset/. [Dostopano: 3. 1. 2021].

[47] Kubernetes skupnost. What is kubernetes? Dosegljivo: https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/, 2020. [Dostopano: 16. 12. 2020].

[48] oprtokodna skupost Stephen Dolan. Github - stedolan/jq: Command-line json processor. Dosegljivo: https://github.com/stedolan/jq, 2020. [Dostopano: 20. 01. 2021].

[49] M. A. Tamiru, G. Pierre, J. Tordsson, and E. Elmroth. Instability in geo-distributed kubernetes federation: Causes and mitigation. In *2020 28th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 1–8, 2020.