

UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO
FAKULTETA ZA MATEMATIKO IN FIZIKO

Jakob Hostnik

Povezovanje klastrov 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

COPYRIGHT. Rezultati diplomske naloge so intelektualna lastnina avtorja in Fakultete za računalništvo in informatiko Univerze v Ljubljani. Za objavo in koriščenje rezultatov diplomske naloge je potrebno pisno privoljenje avtorja, Fakultete za računalništvo in informatiko ter mentorja.

Besedilo je oblikovano z urejevalnikom besedil \LaTeX .

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

TODO Besedilo teme diplomskega dela študent prepíše 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. Mojca 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, sestrám in prijateljem za podporo in spodbudo pri študiju.

Mami Lučki.

Kazalo

Povzetek

Abstract

1	Uvod	1
1.1	Motivacija	1
1.2	Cilj in vsebina naloge	2
2	Problem povezovanje klastrov	5
3	Kubernetes	9
3.1	Zgodovina	9
3.2	Osnovni pojmi	9
4	Povezovanje klastrov Kubernetes	13
4.1	ArgoCD in drugi GitOps	14
4.2	KubeFed	15
4.3	Cilium	17
5	Priprava sistema klastrov za testiranje	19
5.1	Raspberry PI 4	19
5.2	K3S in K3OS	19
5.3	Demonstracijska spletna aplikacija	21
5.4	Namestitev kubefed	22

6	Povezovanje med podatkovnimi centri	23
6.1	Problem velike latence	23
6.2	Povečanje dosegljivosti aplikacije	24
6.3	Povezovanje klastrov Kubernetes med podatkovnimi centri . .	24
6.4	Razporeditev uporabnikov po klastrih	24
6.5	Infrastruktura primera	25
6.6	Implementacija infrastrukture s kubefed	26
6.7	Sinhronizacija podatkov	27
7	Upravljanje izoliranih aplikacij	29
7.1	Zmanjševanje posledic vdorov in izpadov	29
7.2	Implementacija s Kubefed	30
8	Upravljanje klastrov na robu oblaka	31
8.1	Klaster na robu oblaka	31
8.2	Implementacija s kubefed	31
8.3	Sinhronizacija podatkov	32
9	Sklepne ugotovitve	33
	Literatura	35

Seznam uporabljenih kratic

kratica	angleško	slovensko
CRD	custom resource definition	definicija tipov po meri
DNS	domain name system	sistem za upravljanje domen
IP	internet protocol	internetni protokol
HA	high availability	visoka dosegljivost
GA	general availability	splošna dostopnost
VPN	virtual private network	virtualno privatno omrežje

Povzetek

Naslov: Povezovanje klastrov Kubernetes

Avtor: Jakob Hostnik

Več računalnikov povezujemo v skupine predvsem zaradi zagotavljanja večje zmogljivosti in stabilnosti. V zadnjih letih je bil na tem področju narejen zelo velik napredek in razvoj. Tako je nastal tudi sistem Kubernetes, ki je zaradi svoje popularnosti postal svetovni standard za upravljanje klastrov, orkestracijo kontejnerjev. A zelo pogosto en sam računalniški klaster ni dovolj. To se zgodi v primerih, ko imamo težave z dragim prenosom podatkov, preveliko latenco, do naših uporabnikov, želimo še bolj povečati stabilnost ali varnost našega sistema. Primerov uporabe je veliko in v diplomski nalogi si bomo pogledali nekaj najpogostejših. Pogledali si bomo, ozadje povezovanja klastrov postavljenih s sistemom Kubernetes in kakšne pristope lahko uporabimo. Lotili se bomo implementacije in reševanja problemov skozi nekaj pogostih problemov v resničnem svetu. Poseben poudarek pa bomo dali tudi sinhronizaciji podatkov, saj je to eden težjih delov pri upravljanju večih računalniških klastrov. Ugotovimo, da nam lahko sodobne metode povezovanja klastrov zelo olajšajo njihovo upravljanje in preprosto rešijo tudi težje probleme sinhronizacije podatkov.

Ključne besede: gruča, oblak, Kubernetes, računalniški klaster, povezovanje klastrov, mreža klastrov, KubeFed, Cilium, GitOps, ArgoCD.

Abstract

Title: Connecting Kubernetes clusters**Author:** Jakob Hostnik

TODO This sample document presents an approach to typesetting your BSc thesis using L^AT_EX. A proper abstract should contain around 100 words which makes this one way too short.

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

Poglavje 1

Uvod

1.1 Motivacija

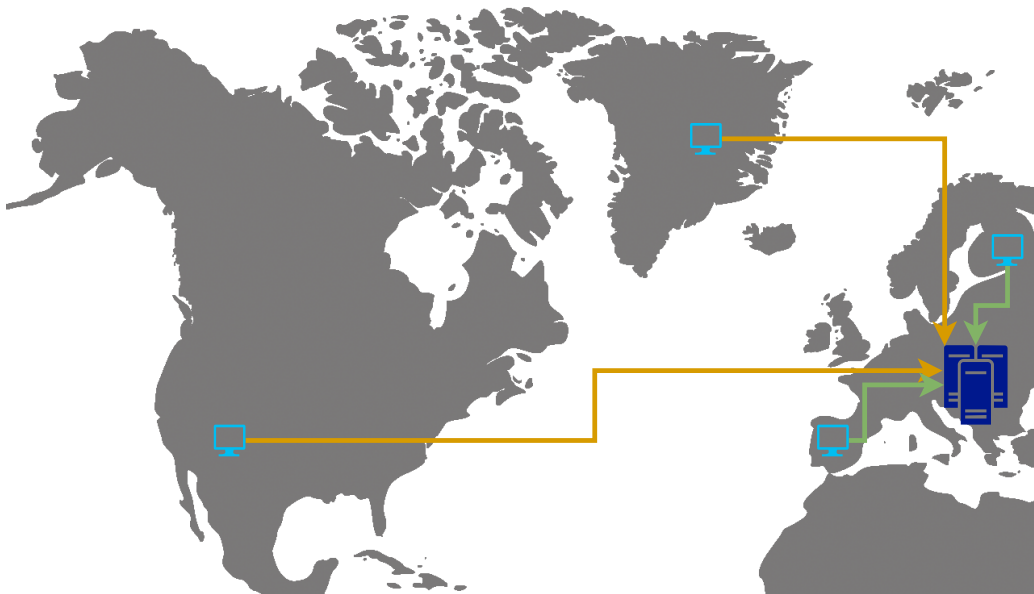
Glede orkestracije kontainerjev je bil v zadnjih nekaj letih narejen zelo velik preboj. V postopku tega preboja se je razvil tudi Kubernetes API, ki je na tem področju rešil marsikatero težavo. Kubernetes je univerzalni način, ki nam omogoča, da več računalnikov povežemo v klaster, ki deluje kot ena samostojna enota. To reši marsikatero težavo v računalništvu in vsaj teoretično je to dovolj, da imamo visoko dosegljivost (HA) naših storitev [11] in sinhrono delovanje večih računalnikov. A industrija je zelo hitro ugotovila, da obstaja kar nekaj primerov, ko ni dovolj, da ena skupina računalnikov deluje kot celota ampak bi želeli med seboj povezujemo tudi te skupine. Ta problem se je do sedaj reševalo na več različnih načinov. Je pa v zadnjem času tudi Kubernetes začel bolj aktivno problem reševati celostno. Začel je s propadlim projektom Federation 1, zdaj pa se zdi, kot da bo Federation 2 ali KubeFed uspešno prešel iz alfa verzije v beta. Skratka v računalništvu je na tem področju zelo veliko zanimanja in razvoja.

1.2 Cilj in vsebina naloge

V tem diplomskem delu si bomo pogledali nekaj o reševanju tega problema, Kubernetesu in reševanju resničnih problemov v industriji. Za vsakega od primerov si bomo pogledali kako se ta problem rešuje v Kubernetes okolju z uporabo orodja KubeFed in kako bi sinhronizirali tudi podatke.

1.2.1 Prevelika latenca

Ko spletne aplikacije postanejo bolj globalne zelo hitro opazimo, da uporabniki, ki so na drugi celini kot naši strežniki preživijo veliko več časa pred ikonami za nalaganje, saj podatki do njih potujejo dalj časa. Ta problem je rešljiv na način, da postavimo še en klaster bližje naših uporabnikov. Napriimer en klaster na celino. Tu pa zelo pogosto želimo, da se podatki sinhronizirajo. Zavedati se moramo, da strežniki v klastru zelo veliko komunicirajo zato morajo biti tudi fizično blizu skupaj. To je tudi ena glavnih omejitev, da ne moremo vseh klastrov povezati v en večji klaster.



Slika 1.1: Problem prevelike latence.

1.2.2 Viša dosegljivost

Vsaki nekaj let se zgodi da slišimo novico ko iz interneta izpade kakšen podatkovni center. To se lahko zgodi iz več razlogov, najpogostejša pa sta naravne nesreče in napake na sistemu. Če gre v takšnem primeru za večjega oblachnega ponudnika se to pozna tako, da je nezanemarljiv del interneta nedosegljiv. Izpadi posameznih klastrov pa se toliko pogosteje dogajajo, če gre za manjše ponudnike ali pa so naši strežniki v bolj nestabilnih okoljih. V splošnem se problem reši tako, da se namesti v vsak podatkovni center, skupino strežnikov en klaster.

1.2.3 Izolacija aplikacije

Ko govorimo o izolaciji aplikacije se ponavadi nanašamo na varnost pri vdoru, ali pa na večjo dosegljivost. Glede izolacije spletnih aplikacij smo z uporabo Kubernetesa naredili že kar nekaj korakov. Naprimer vsaka aplikacija lahko teče v svojem kontejnerju, tudi v imenskem prostoru. Lahko jo celo izoliramo samo na določera vozlišča. A vseeno se v Kubernetesu dogajajo problemi, ki naredijo cel klaster nedosegljiv. Kaj takšnega se najpogosteje zgodi med posodabljanjem celega klastra. Z varnostnega vidika pa ponavadi mislimo na dejstvo, da če nekomu uspe serija napadov in se uspešno polasti enega samega vozlišča si začne lastiti cel klaster. Torej ima dostop tudi do vseh drugih aplikacij. Če imamo vsako od naših aplikacij v svojem klastru pa se temu izognemo.

1.2.4 Drag prenos podatkov

Če se spustimo iz jedra računalniškega oblaka na njegov rob pa tam srečamo cel kup 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 naprimer mikro podatkovni centri in klastri, na majhnih računalnikih kot naprimer raspberry pi. Če naša aplikacija uporablja takšne klastre je potrebno tudi njihovo sinhrono delovanje. Takšni majhni klastri so

poleg že znanega problema prevelike razdalje ponavadi obsojeni, da s centralnimi strežniki komunicirajo samo minimalno, saj zelo pogosto za prenos podatkov uporabljajo draga mobilna omrežja.

1.2.5 Razdeljevanje dela po različnih lokacijah

Na robu oblaka pa se ponavadi srečamo ne samo z omejenim komuniciranjem s centralnim strežnikom ampak tudi zmanjšano dosegljivostjo in tudi zmožljivostjo naprav. Torej če ima en klaster manj dela kot drugi lahko delež tega prenese na druge klastre.

Poglavje 2

Problem povezovanje klastrov

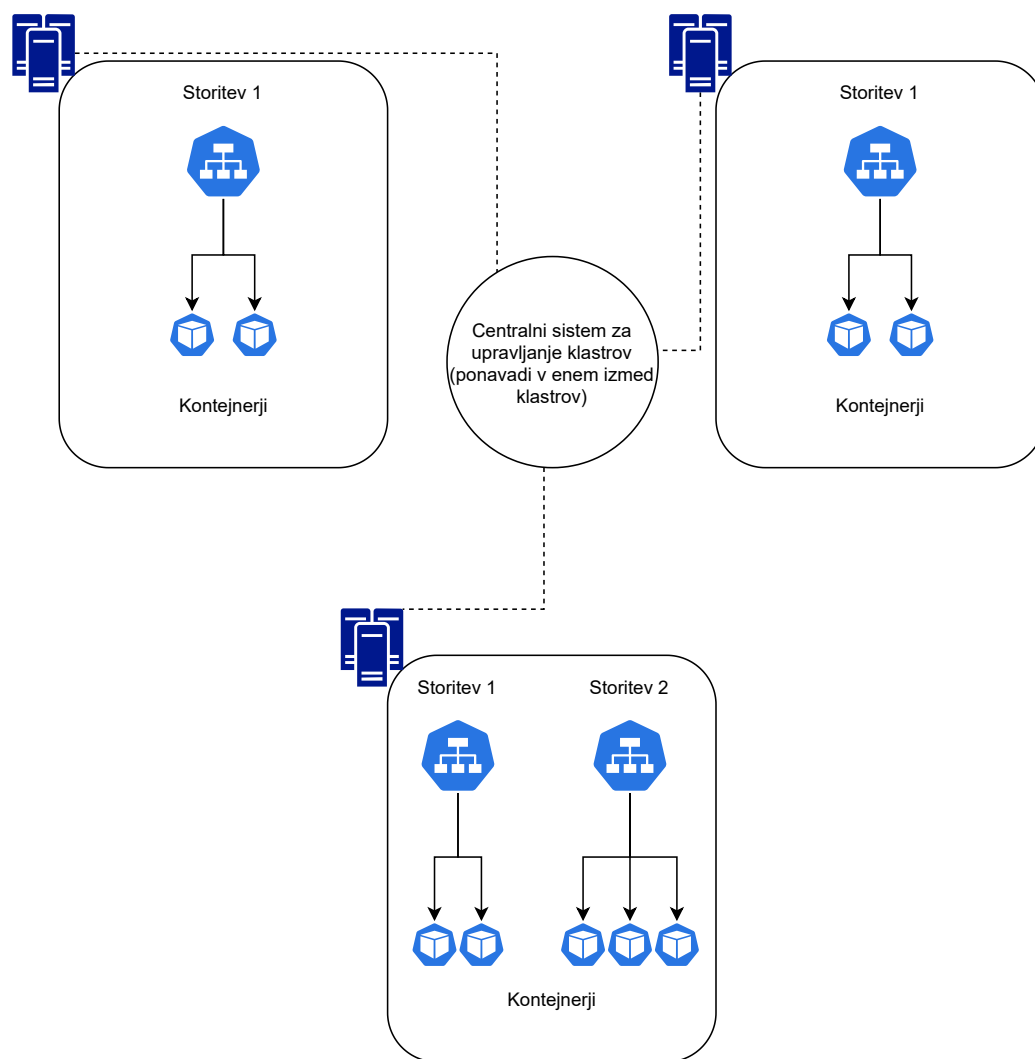
Računalniški klaster je skupina računalnikov, ki zaradi večje zanesljivosti in zmogljivosti skupaj opravlja določene storitve. Zaradi praktičnosti, pa so te storitve ponavadi napisane tako, da vsako storitev sestavlja več programov, ki tečejo v kontejnerjih.

Problem povezovanja večih računalniških klastrov je v svetu prisoten že kar nekaj časa. Ko postavimo klaster, želimo da več računalnikov deluje kot eno, 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 en tesno povezan klaster. V takšnih primerih ponavadi lahko računalnike povežemo v več tesno povezanih klastrov, te pa potem na različne načine povežemo šibkeje.

Ko govorimo o tesni povezanosti znotraj klastra ponavadi pričakujemo, da vsako vozlišče vidi vsako drugo, da vsak kontejner lahko komunicira z vsakim kontejnerjem, da so vozlišča v istem omrežju, da je povezava med vozlišči hitra, poceni in zanesljiva. Pričakujemo, da sistem, ki ga uporabljamo za klaster omogoča, razporejanje zaželenih storitev in kontejnerjev med vozlišči in v primeru izpada vozlišča to vozlišča odstrani iz sistema in storitve s tega vozlišča prerazporedi na preostala vozlišča.

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

je povezava med vozlišči počasna, nezanesljiva in draga. Ponavadi je vsak klaster v svojem omrežju in je nedosegljiv ostalim klastrom. Pričakujemo, da vsak klaster skrbi za svoja vozlišča in da ohranja svoje storitve in kntejnerje v delovanju. Od sistema za povezovanje klastrov pa si želimo, da nam omogoča centralni nadzor nad storitvami v klastrih, prerazporejanje teh storitev med klastri, dinamično odkrivanje drugih klastrov in njihovih storitev, izločanje nedosegljivih klastrov, povezljivost med vsemi vozlišči in kontejnerji, četudi so vozlišča v različnih omrežjih.



Slika 2.1: Primer povezanih večih klastrov.

Poglavje 3

Kubernetes

Kubernetes je REST API in trenutno obstaja že več kot 72 implementacij tega API-ja. Ko se bomo sklicevali na Kubernetes v tem dokumentu bomo imeli v mislih implementacijo Kubernetesa.

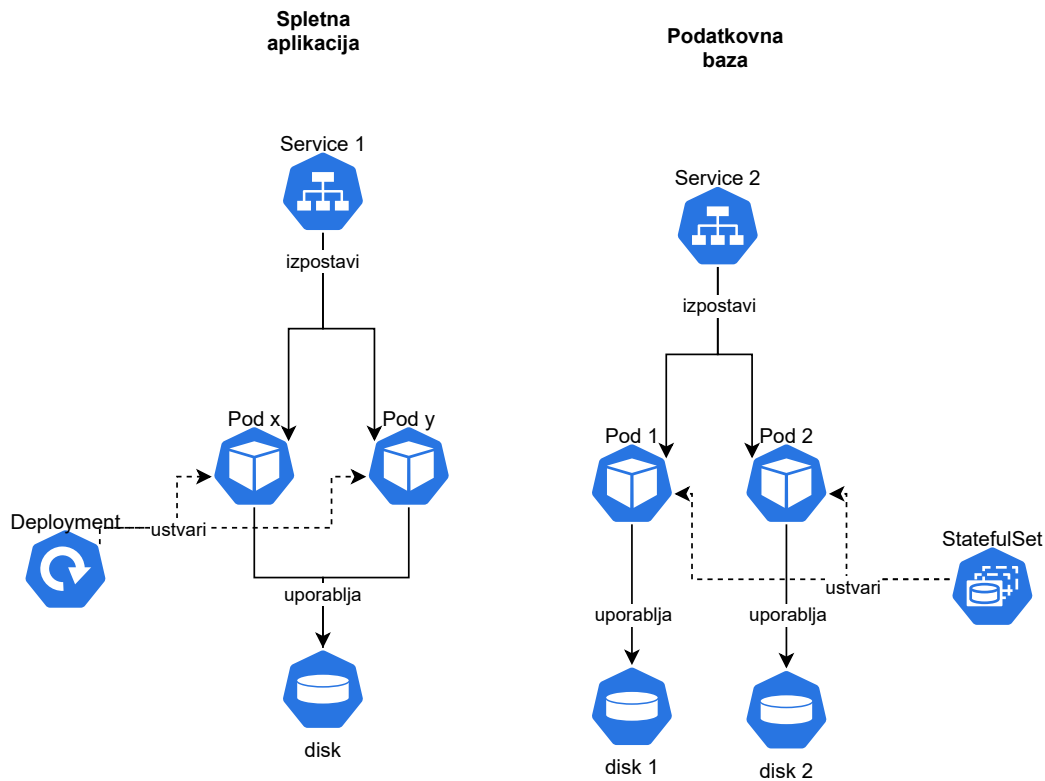
3.1 Zgodovina

Leta 2014 je Google objavil in odprl kodo od projekta Kubernetes [24]. Gre za program bil ustvarjen z namenom, da poenostavi upravljanje kontejnerjev in večjih računalniških klastrov 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 klastrov Borg. Kasneje leta 2013 je google predstavil sistem Omega. Leta 2014 pa je google objavil projekt Kubernetes, kot odprtokodno verzijo borga. Kasneje je upravljanje prevzela organizacija Cloud Native Computing Foundation.

3.2 Osnoovni pojmi

Kubernetes API nam omogoča, da v Kubernetes shranjujemo najrazličnejše objekte. Takšne, ki so v naprej definirani ali pa smo jih definirali sami (CRD). Najpogostejši v naprej definirani objekti, 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 [20]

Objekt pod ponavadi predstavlja neko instanco mikrostoritve. Gre za najmanjšo enoto v Kubernetesu, ki lahko teče v klastru. Je sestavljen 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 [21]

Objekt service ponavadi označuje vse pode ene mikrostoritve. Kubernetes iz objekta v internem DNS ustvari domeno za mikrostoritev in dinamično

razvršča promet med našimi podi. service uporabljamo tako, da namesto pošiljanja zahtevkov direktno na IP naslov Poda, delamo klice na ustvarjeno domeno. Takšen zahtevek potem dobi en izmed označenih podov v objektu service.

3.2.3 Deployment [18]

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 [23]

Objekt zelo podoben deploymentu, le da statefulset vsaki replikaciji poda dodeli unikatno številno. pod, ki se ustvari s to številko ohranja diske in IP naslov in domeno. 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

Povezovanje klastrov Kubernetes

Ko postavimo več različnih klastrov imamo vedno možnost, da upravljamo vsakega posebej [10]. A takšen pristop zelo kmalu odpove, če imamo takšnih klastrov res veliko. Ko govorimo o sistemu, ki ga uporabljamo za upravljanje večih klastrov, najpogosteje pričakujemo možnost prenašanje objektov med klastri. Tako lahko objekt definiramo samo enkrat in bo naš sistem ta objekt ustvaril v klastrih, kjer to želimo. Odvisno od naših potreb pa si lahko želimo tudi, da nam sistem omogoča dinamično odkrivanje servisov z enako definicijo v različnih klastrih, komunikacijo med servisi v različnih klastrih, dinamično odkrivanje podov med klastri in komunikacijo med podi v različnih klastrih. Te funkcionalnosti znotraj enega klastra nudi že Kubernetes sam. Je pa seveda odvisno od našega primera katere funkcionalnosti želimo uporabiti in kako kompleksno postavitev potrebujemo. V nadaljevanju si bomo pogledali različne sisteme za povezovanje klastrov Kubernetes, njihove glavne prednosti in značilnosti.

4.1 ArgoCD in drugi GitOps

4.1.1 Sinhronizacija objektov z uporabo GitOps sistemov

GitOps pristop pri postavljanju strukture aplikacij v klastrih Kubernetes se je izkazal za dober pristop za upravljanje klastrov. Osnovna ideja GitOpsa je to, da imamo našo strukturo aplikacij v klastru napisano v repozitoriju Git in potem je kontroler GitOps tisti, ki iz teh definici postavi strukturo klastra. Če uporabljamo kakšnega od sistemov GitOps lahko potem iz enakega repozitorija postavimo več klastrov. 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 klastrih ali pa odkrivanje storitev ali podov. V nadaljevanju si bomo izbrali sistem ArgoCD in si pogledali kako bi si postavili zgoraj opisano infrastrukturo.

4.1.2 Postavljanje sinhronizacija objektov z ArgoCD

ArgoCD podpira več različnih formatov konfiguracije klastra [12]. Najpreprostejše je, če uporabimo kar YAML format datoteke z definicijami objektov, ki jih želimo namestiti v vsak klaster. Za deployanje te konfiguracije na več kot en klaster imamo na voljo dva pristopa. Prvi način je, da v vsak klaster 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 klastre [13]. To pomeni, da moramo imeti samo v enem klastru nameščen ArgoCD kontroler.

Pogosto pa ne želimo, da imajo vsi klastri popolnoma enako infrastrukturo in želimo vsaj malo prilagoditi konfiguracijo posameznega klastra. V tem primeru bi uporabili format zapisa konfiguracije, ki podpira predloge. ArgoCD nam ponuja možnost, da ročno določimo spremenljivke predlogam. Tako lahko uporabimo naprimer predloge HELM in ArgoCD nam

bo omogočil, da vsakemu klastru izberemo svojo datoteko s spremenljivkami. Glede na preprostost delovanja takšnega sistema se moramo zavedati da od njega ne moremo pričakovati nikakeršnih naprednih funkcionalnosti kot sta dinamično odkirvanje storitev ali komunikacija podov med klastri. Takšen sistem nam omogoča samo sinhronizacijo infrastrukture.



Slika 4.1: Primer uporabe helm predloge v ArgoCD.

4.2 KubeFed

9. 1. 2018 je bil po propadlem projektu Kubernetes Federation V1 ustvarjen Kubernetes Federation V2 ali KubeFed [7]. Oba projekta sta želela poeno-

staviti upravljanje večih klastrov in razporejanje Kubernetes objektov. V projektu Federation V1 je bil ubran pristop, ki je skupino klastrov ali federacijo uporabniku predstavil kar kot nov klaster [22]. Uporabljal je svoj API in api kontroler, ki pa je bil združljiv s Kubernetesovim, kar pa je omogočalo tudi uporabo orodja kubectl [3]. Objekti, ki jih je federacija podpirala so bili kompatibilni s standardnimi Kubernetes objekti [19]. Takšne objekte je potem Federation V1 ustvaril tudi v ostalih klastrih. Zanimiv pristop, ki pa zaradi mnogih pomankljivosti in pomankanja možnosti naprednejših konfiguracij ni uspel pridobiti statusa GA.

Tako se je kasneje rodil projekt Federation V2 [7]. Glavna razlika s prvo verzijo z uporabniškega stališča je v tem, da za federacijo ne poizkuša imitirati Kubernetesovega API, 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 s komando `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 klastre in prepis lastnosti originalnega objekta za posamezne klastre. Takšen pristop je zelo široko zastavljen in omogoča tudi federacijo CRD objektov.

```
apiVersion: types.kubefed.io/v1beta1
kind: FederatedDeployment
spec:
  placement:
    clusterSelector:
      # izbira klastrov
      matchLabels: {}
      ...
  template:
```



```
# specifikacije deployment objekta
spec:
...
overrides:
  # prepis konfiguracije za posamezne klastre
  - clusterName: klaster1
    clusterOverrides:
      # nastavi polje replicas na vrednost 5
      - path: "/spec/replicas"
        value: 3
...
```

Federation V2 podpira poleg sinhronizacije infrastrukture tudi odkiravnje storitev v drugih klastrih prek DNS zapisov. Ampak se omenja možnost odstranitve te funkcionalnosti [17], 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 bo na voljo splošni javnosti(GA), če slučajno ne bo šel po stopinjah svojega predhodnika.

4.3 Cilium

Cilium je odprtokodni program, ki nam omogoča napredne varnostne in omrežne nastavitve v našem klastru [14]. Program deluje na tretji in četrti omrežni plasti, kjer nam zagotavlja osnovne principe varnosti in zaščite. Tu govorimo o zapiranju portov in omejevanju prometa. Cilium pa nam zagotavlja tudi naprednejšo varnost na sedmi omrežni plasti, saj nam omogoča na primer omejevanje in filtriranje HTTP zahtevkov in podobne varnostne funkcije na popularnih protokolih aplikacijskega nivoja [14].

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 klastrov Kubernetes. Tako nam kot

ključno prednost Cilium omogoča tudi komunikacijo med podi v različnih klastrih [15] in uporabo globalnih objektov service, ki so sposobni delati razporejanje prometa med različnimi klastri. Takšne objekte definiramo z anotacijo `io.cilium/global-service` [16]. Omogoča nam tudi omejevanje povezovanja med klastri z njihovim objektom `CiliumNetworkPolicy` [16]. Ko postavljamo mrežo klastrov, pa se moramo še vedno zavedati, da Cilium ne rešuje problema, če so naši klastri skriti v različnih privatnih omrežjih. Ključno pri uporabi Ciliuma za povezovanje klastrov je, da so vsa naša vozlišča dosegljiva med seboj. A četudi so naši klastri v med seboj nedosegljivih privatnih omrežjih, pa je problem z lahkoto rešljiv z uporabo sistema VPN, ki nam omogoča, da vsa vozlišča povežemo v eno virtualno omrežje [16].

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

Poglavje 5

Priprava sistema klastrov za testiranje

5.1 Raspberry PI 4

Za namene testiranja različnih načinov povezovanja klastrov Kubernetes moramo najprej postaviti nekaj klastrov Kubernetes. 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 je neglede na vse naš klaster in je delo zelo podobno, če uporabimo nekaj 1000 vozlišč v klastru v oblaku ali pa lokalni klaster 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 namestil Kubernetes klaster z enim samim vozliščem. Fizična postavitve klastrov je prikazana na sliki 5.1.

5.2 K3S in K3OS

Obstaja več implementacij Kubernetesa in mi bomo uporabili z viri varčno odprotokodno implementacijo K3S od podjetja Rancher [9]. Hkrati so v podjetju Rancher pripravili distribucijo operacijskega sistema Linux K3OS,



Slika 5.1: Postavitev raspberry klastrov.

ki jo lahko namestimo na katerikoli računalnik [8]. 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 [2]. 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 z Kubernetes klastrom in način v katerem želimo zagnati K3S na sistemu [8].

```
ssh_authorized_keys:
```

```
- ssh-rsa ...
```

```
hostname: klaster-1
k3os:
  ntp_servers:
  - ...
  password: ...
  token: ...
  dns_nameservers:
  - ...
  wifi:
  - name: ...
    passphrase: ...
  k3s_args:
  - server
```

V našem primeru smo vse K3S programe zagnali v strežniškem načinu in nobenega v načinu delovnega vozlišča, saj želimo, da vsak Raspberry PI predstavlja svoj klaster.

5.3 Demonstracijska spletna aplikacija

Za potrebe testiranja je bilo potrebno narediti 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 objavaljena v Git repozitoriju [5], je napisana v programskem jeziku Go. Iz kode je bil zgeneriran kontejner, ki je objavljen v javnem Docker repozitoriju [6]. Ob tem velja opozoriti, da Raspberry PI uporablja ARM arhitekturo procesorja, kar je zahtevalo posebno pozornost.

Aplikacija deluje preprosto. Na mrežnih vratih podanih z okoljsko spremenljivko 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:8080/users --data '{"name": "John", "lastname": "Doe"}'
# ukaz za prikaz vseh uporabnikov
curl localhost:8080/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 omoča napredne sinhronizacije tudi med različnimi strežniki in klastri. K sreči pa CrateDB implementira PostgreSQL vmesnik in nam kode za prehod med bazami ni potrebnost spreminjati.

5.4 Namestitev kubefed

Kot ena izmed ključnih komponent složnega delovanja večih klastrov je njihovo upravljanje. V te namene bomo uporabili program kubefed, ki ga moramo namestiti na enega izmed klastrov, 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 KubeFed pa moramo na svoj računalnik namestiti orodje kubefedcli. Z uporabo ukaza kubefedctl join povežemo vse tri klastre v kubefed sistem.

```
kubefedctl join klaster-1
kubefedctl join klaster-2
kubefedctl join klaster-3
```

S tem smo uspešno povezali več klastrov Kubernetes v sistem KubeFed.

Poglavje 6

Povezovanje med podatkovnimi centri

6.1 Problem velike latence

V industriji je zelo malo primerov spletna aplikacije, 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 računalništvu. Najbolj očiten primer je sinhronizacija podatkov. V našem primeru bomo uporabili podatkovno bazo CrateDB [1], novejšjo 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 base PostgreSQL.

6.2 Povečanje dosegljivosti aplikacije

Če je čim višja dosegljivost za našo aplikacijo kritičnega pomena in smo že poskrbeli za visoko dosegljivost (HA) aplikacije v našem klastru, še vedo lahko pride do situacije, ko iz omrežja izpade cel podatkovni center. Spomnimo, da je ena izmed zahtev, za učinkovito delovanje Kubernetesa, postavitev strežnikov blizu skupaj. V primeru naravnih nesreč ali hujših vremenskih pogojev pomeni, da je nedosegljiv cel podatkovni center in s tem klaster v njem. Če uporabljamo oblak, pa gremo lahko še korak dlje z zagotavljanjem dosegljivosti. Če nam ni dovolj niti, da uporabimo različne dosegljivostne cone in podatkovne centre oblačnih ponudnikov, lahko postavimo naše klastre 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.

6.3 Povezovanje klastrov Kubernetes med podatkovnimi centri

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

6.4 Razporeditev uporabnikov po klastrih

Ko imamo na vsakem klastrih javno izpostavljen service in urejene ingress zapise moramo še vedno uporabnike preusmeriti na njim najbližji klaster. Uporabnike lahko mi usmerimo avtomatsko z DNS zapisi, ki omogočajo usmerjanje na podlagi geolokacije. Lahko uporabimo in namestimo zunanjo DNS skozi Kubernetes ali pa kar ročno. V naših lokalnih testnih klastrih bomo ta korak preskočili in jih ne bomo usmerjali preko javnih dns strežnikov, saj v

The screenshot shows the 'Quick create record' interface in AWS Route 53. The form is for creating a new record named 'storitev.com'. The 'Routing policy' is set to 'Geolocation'. The 'Record type' is 'A - Routes traffic to an IPv4 address and so...'. The 'Value' field contains '192.0.2.235'. The 'TTL (seconds)' is set to '300'. The 'Location' is 'Europe'. The 'Health check' is optional and set to 'Choose health check'. The 'Record ID' is 'US West load balancer'. There are buttons for 'Cancel' and 'Create records'.

Slika 6.1: Ustvarjanje geolozijskega DNS zapisa v storitvi ROUTE53.

lokalnem okolju to ni smiselno.

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 katerem klastru ali strežniku želi igrati. V takšnih primerih se ponavadi tudi izognemo problemu sinhronizacije podatkov med strežniki, kar zelo poenostavi upravljanje naših klastrov.

6.5 Infrastruktura primera

V našem primeru spletne aplikacije bomo imeli v vsakem klastru en deployment z našo aplikacijo Stateful rest sample in en pripadajoči service, da aplikacijo izpostavimo izven klastra. Za podatkovno bazo pa bomo uporabili statefulset s CrateDB, diskom na lokalni SD kartici, internim servicom za prepoznavo ostalih instanc CrateDB v klastru in eksterni service, ki ga uporabimo za povezovanje na podatkovno bazo. Vsa konfiguracija je javno objavljena na Git repozitoriju[4]. Postavimo jo z ukazom.

```
kubectl apply -f diploma-demo-1
```



```
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 vse klastre in za predlogo kar podane objekte. Tako je za nas rezultat izvršenih ukazov kreiranje federated objektov in posledično kopiranje objektov v vse naše povezane klastre.

Po preizkusu delovanje z `curl` komando opazimo, da podatki med klastri še vedno niso sinhronizirani. Uporabniki, ki jih vnesemo v en klaster se še ne sinhronizirajo v ozadju. Na tej točki se ustavijo nekatere spletne aplikacije in prepustijo izbiro strežnika oziroma klastra kar uporabniku.

6.7 Sinhronizacija podatkov

Če želimo pred uporabnikom skriti, da uporabljamo več strežnikov je poleg `dns` strežnika, ki razporeja uporabnike ključno, da se podatki sinhronizirajo med seboj. Sicer v našem primeru res uporabljamo samo eno instanco `CrateDB` baze a vseeno smo na nivoju sinhronizacije znotraj klastra to stvar že uredili. Zopet se moramo spomniti, da so tudi klastri podatkovnih baz ponavadi narejeni tako, da najbolje delujejo če so vozlišča blizu skupaj. Zaradi tega mnoge baze, ki podpirajo sinhronizacijo podatkov znotraj klastra, podpirajo tudi sinhronizacijo med različnimi podatkovnimi centri.

6.7.1 Uporaba primerne podatkovne baze

Vedno je najlepše, če uporabimo podatkovno bazo, ki ima takšno sinhronizacijo s čim manj dela že podprto, a takih baz ni veliko na trgu. Konkretno CrateDB podpira sinhronizacijo tudi preko dosegljivostnih con. Vseeno pa je mišljeno, da vsa vozlišča povežemo v enak podatkovni klaster. Kar pomeni, da morajo vsa vozlišča imeti dostop do vseh. Najlažje je, če si lahko privoščimo in izpostavimo vsako vozlišča s svojim javnim IPjem in jih ročno povežemo v klaster. <https://crate.io/docs/crate/howtos/en/latest/clustering/multi-zone-setup.html> Nastavimo še nastavitve, ki jih baza podpira za zmanjšanje prometa in zagotavljanje željenje dosegljivosti med klasteri. Podobne načine sinhronizacije podpira tudi na primer podatkovna baza Cassandra. <https://docs.datastax.com/en/cassandra/2.2/cassandra/initialize/initMultipleDS.html>

6.7.2 Podatke sinhroniziramo sami

Sinhronizacija podatkovne baze tažak problem in če ne uporabimo primerne podatkovne baze ali pa želimo sinhronizirati samo določene stvari preko klastrov, bo za nas najverjetneje prišla v poštev sinhronizacija na roke. To pomeni, da bomo spisali novo mikrostoritev, ki bi v ozadju kopirala ključne podatke med podatkovnimi centri. Ker samo mi poznamo naš konkreten primer uprabe, je takšen pristop pogosto najbolj učinkovit.

V našem primeru bomo s preprosto skripto kopirali uporabnike iz ene aplikacije v drugo, kar z uporabo našega REST apija. To bomo storili v drugem ubuntu kontejnerju z uprabo komand curl za izvajanje rest api klicev in 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 192.168.10.111/users|jq --data '{"name": "John", "lastname": "Doe"}'
curl -s 192.168.10.112/users|jq
[{"Name": "John", "Lastname": "Doe"}]
```

Poglavje 7

Upravljanje izoliranih aplikacij

7.1 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 dosegljiv in sistema v katerega ne bo mogoče vdreti. To vsake toliko časa potrdijo največji igralci z izpadi ali vdori na njihovih storitvah. Vsake nekaj časa pa se vseeno pojavi podjetje, ki trdi da vdor k njim ni mogoč in so vsakič znova prepričani v nasprotno. Vseeno pa kljub temu, da so vdori in napake, zaradi katerih postanejo nedosegljivi naši klastri vedno lahko poizkusimo zmanjšati posledice ob morebitnem vdoru ali napadu.

7.1.1 Izpadi aplikacije

Kljub temu, da smo naše aplikacije namestili na različne klastre in je s tem aplikacija odporna na izpad enega klastra pa ob hujših nepravilnostih delovanja aplikacije in nastavitvah klastra lahko zaradi napake na eni aplikaciji izpadejo vsi klastri na katerih imamo aplikacijo nameščeno. Takšen primer bi bil, če ena aplikacija ali mikrostoritev zavzame vse vire na klastru 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 klaster, kjer napake drugih aplikacij ne bodo vplivale na naše delovanje. A vseeno se moramo zaveati, da je ta korak smislen šele ko smo opravili že vse predhodne preventivne ukrepe, kot so razdelitev aplikacije na mikrostoritve, kontejnerjizacija, izolacija na posamezno Kubernetes vozlišče, pravilna nastavitve omejitev avtomatskega povečevanja in še mnoge druge.

7.1.2 Vdori

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

7.2 Implementacija s Kubefed

Ključna lastnost federiranih objektov je možnost izbire klastov na katerih se bo določen objekt ustvaril. S tega stališča je naš primer zelo preprost. Samo določimo da se naša aplikacija izvaja na klastru 3 namesto na vseh. Tokrat za federacijo ne moremo uporabiti ukaza `kubefedctl federate`, ampak moramo kot za vse ostale Kubernetes objekte spisati konfiuracijo. Najprej bomo z ukazom `kubectrl tag` označili naš izoliran klaster (ali več njih). Potem pa bomo lastnosti vsakega `spec.placement.clusterSelector.matchLabels` federiranega objekta, ki ga želimo izolirati dodali označbo vseh izoliranih klastrov. V takšnih primerih se nam ni potrebno posebej ukvarjati s sinhronizacijo podatkov, saj ali smo vse podatke obdržali v istem klastru ali pa sinhroniziramo na enak način kot če sinhroniziramo med različnimi podatkovnimi centri.

Poglavje 8

Upravljanje klastrov na robu oblaka

8.1 Klaster na robu oblaka

Razlogov zakaj klastre postavljamo na rob oblaka oziroma fizično bližje končnemu uporabniku je več. Pogosto želimo na glavni strežnik pošiljati vseh podatkov ampak jih obdelovati že lokalno, da ne prenašamo po omrežju vseh podatkov. Smiselnost takšne postavitve pride še posebej do izraza, če podatke prenašamo bo dragem mobilnem omrežju. A možnih postavitev in razlogov zanje je res veliko. Naprimer lahko gre za zahteve strank, da se podatki obdelujejo lokalno, lahko gre za zakonske omejitve lahko želimo operacije izvajati na napravah, ki si jih ne lastimo direktno itd.

V našem primeru se bomo osredotočali na upravljanje takšnih klastrov.

8.2 Implementacija s kubefed

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

Nam pa kubefed omogoča še eno lepo možnost s svojo strukturo, saj je vsak od federiranih objektov v svojem imenskem prostoru. Tako lahko s svojim kontrollerjem in kubefed apijem implementiramo še dodatne funkcionalnosti kot so razporejanje obramenjenosti med lokalnimi strežniki in po potrebi povečujemo število instanc ali pa kar razporejamo opravila s Kubernetes Jobi.

Z zelo preprosto integracijo v Kubernetes nam kubefed api tu res omogoča zelo preprosto implementacijo katerekoli naše rešitve.

8.3 Sinhronizacija podatkov

asdf V primeru klastrov na robu oblaka se sinhronizacijo praviloma vedno implementira na novo, saj le mi vemo kakšen problem rešujemo in zakaj smo sploh postavljali klastre na robu oblaka.

Poglavje 9

Sklepne ugotovitve

Uporaba \LaTeX a in \BibTeX a je v okviru Diplomskega seminarja **obvezna**! Izbira \LaTeX ali ne \LaTeX pri pisanju dejanske diplomske naloge pa je prepuščena dogovoru med vami in vašim mentorjem. Res je, da so prvi koraki v \LaTeX u težavni. Ta dokument naj vam služi kot začetna opora pri hoji. Pri kakršnihkoli nadaljnih vprašanjih ali napakah pa svetujem uporabo Googla, saj je spletnih strani za pomoč pri odpravljanju težav pri uporabi \LaTeX a ogromno.

Literatura

- [1] Cratedb: the distributed sql database for iot and time-series data. Dosegljivo: <https://crate.io/>, 2020. [Dostopano: 16. 12. 2020].
- [2] odprtokodna skupnost Dennis Brentjes, Sjors Gielen. Picl k3os image generator. Dosegljivo: <https://github.com/sgielen/picl-k3os-image-generator>, 2020. [Dostopano: 16. 12. 2020].
- [3] Lukasz Guminski. Cluster federation in kubernetes 1.5. Dosegljivo: <https://kubernetes.io/blog/2016/12/cluster-federation-in-kubernetes-1-5/>. [Dostopano: 16. 12. 2020].
- [4] Jakob Hostnik. Connecting kubernetes clusters. Dosegljivo: <https://github.com/hostops/connecting-kubernetes-clusters>, 2020. [Dostopano: 16. 12. 2020].
- [5] Jakob Hostnik. Stateful rest sample. Dosegljivo: <https://github.com/hostops/stateful-rest-sample>, 2020. [Dostopano: 16. 12. 2020].
- [6] Jakob Hostnik. Stateful rest sample. Dosegljivo: <https://hub.docker.com/r/hostops/stateful-rest-sample>, 2020. [Dostopano: 16. 12. 2020].
- [7] Shashidhara T D Irfan Ur Rehman, Paul Morie. Kubernetes federation evolution. Dosegljivo: <https://kubernetes.io/blog/2018/12/12/kubernetes-federation-evolution/>, 2018. [Dostopano: 16. 12. 2020].

-
- [8] Rancher Labs. K3os. Dosegljivo: <https://github.com/rancher/k3os>, 2020. [Dostopano: 16. 12. 2020].
 - [9] Rancher Labs. K3s: Lightweight kubernetes. Dosegljivo: <https://k3s.io/>, 2020. [Dostopano: 16. 12. 2020].
 - [10] 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: 16. 12. 2020].
 - [11] Gigi Sayfan. *Mastering Kubernetes*. Packt Publishing Ltd., 2017.
 - [12] ArgoCD skupnost. Argo cd - declarative gitops cd for kubernetes. Dosegljivo: <https://argoproj.github.io/argo-cd>. [Dostopano: 16. 12. 2020].
 - [13] ArgoCD skupnost. Declarative setup. Dosegljivo: <https://argoproj.github.io/argo-cd/operator-manual/declarative-setup>. [Dostopano: 16. 12. 2020].
 - [14] Cilium skupnost. Introduction to cilium & hubble. Dosegljivo: <https://docs.cilium.io/en/latest/intro/>. [Dostopano: 16. 12. 2020].
 - [15] Cilium skupnost. Multi-cluster (cluster mesh). Dosegljivo: <https://docs.cilium.io/en/latest/concepts/clustermesh/>. [Dostopano: 16. 12. 2020].
 - [16] Cilium skupnost. Set up cluster mesh. Dosegljivo: <https://docs.cilium.io/en/latest/gettingstarted/clustermesh/>. [Dostopano: 16. 12. 2020].
 - [17] KubeFed skupnost. kubefed: remove crossclusterservicediscovery feature. Dosegljivo: <https://github.com/kubernetes-sigs/kubefed/issues/1283>, 2020. [Dostopano: 16. 12. 2020].

-
- [18] Kubernetes skupnost. Deployments. Dosegljivo: <https://kubernetes.io/docs/concepts/workloads/controllers/deployment/>. [Dostopano: 16. 12. 2020].
- [19] Kubernetes skupnost. Federated cluster. Dosegljivo: <https://v1-16.docs.kubernetes.io/docs/tasks/federation/administer-federation/cluster/>. [Dostopano: 16. 12. 2020].
- [20] Kubernetes skupnost. Pods. Dosegljivo: <https://kubernetes.io/docs/concepts/workloads/pods/>. [Dostopano: 16. 12. 2020].
- [21] Kubernetes skupnost. Service. Dosegljivo: <https://kubernetes.io/docs/concepts/services-networking/service/>. [Dostopano: 16. 12. 2020].
- [22] Kubernetes skupnost. Set up cluster federation with kubefed. Dosegljivo: <https://v1-16.docs.kubernetes.io/docs/tasks/federation/set-up-cluster-federation-kubefed/>. [Dostopano: 16. 12. 2020].
- [23] Kubernetes skupnost. Statefulsets. Dosegljivo: <https://kubernetes.io/docs/concepts/workloads/controllers/statefulset/>. [Dostopano: 16. 12. 2020].
- [24] Kubernetes skupnost. What is kubernetes? Dosegljivo: <https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/>, 2020. [Dostopano: 16. 12. 2020].