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

Jakob Hostnik

Povezovanje Kubernetes klastrov

diplomsko delo

interdisciplinarni univerzitetni
študijski program prve stopnje
računalništvo in matematika

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

Ljubljana, 2018

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

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

Na tem mestu zapišite, komu se zahvaljujete za izdelavo diplomske naloge. Pazite, da ne boste koga pozabili. Utegnil vam bo zameriti. Temu se da izogniti tako, da celotno zahvalo izpustite.

Mami Lučki.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
CA	classification accuracy	klasifikacijska točnost
DBMS	database management system	sistem za upravljanje podatkovnih baz
SVM	support vector machine	metoda podpornih vektorjev

Povzetek

Naslov: Povezovanje Kubernetes klastrov **Avtor:** Jakob Hostnik V vzorcu je predstavljen postopek priprave diplomskega dela z uporabo okolja LaTeX. Vaš povzetek mora sicer vsebovati približno 100 besed, ta tukaj je odločno prekratek. Dober povzetek vključuje: (1) kratek opis obravnavanega problema, (2) kratek opis vašega pristopa za reševanje tega problema in (3) (najbolj uspešen) rezultat ali prispevek magistrske naloge. **Ključne besede:** cluster, cloud, kubernetes, kubefed.

Abstract

Title: Connecting Kubernetes clusters**Author:** Jakob HostnikThis sample

document presents an approach to typesetting your BSc thesis using LaTeX. A proper abstract should contain around 100 words which makes this one way too short. **Keywords:** gruča, oblak, kubernetes, kubefed.

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 kuberentes 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 naših storitev in sinhrono delovanje večih računalnikov. A industrija je zelo hitro ugotovila, da to ne bo dovolj, saj 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. Najprej s propadlim projektom kubefed 1 zdaj pa se zdi, kot da bo kubefed 2 uspešno prešel iz alfa verzije v beta. Skratka v računalništvi je na tem področju zelo veliko zanimanja in razvoja.

1.2 Cilj in vsebina naloge

V tem diplomskem delu bomo na problem gledali, kot da rešujemo realne probleme v industriji. Pri reševanju pa se bomo posebej ozirani tudi na aplikacije, ki morajo sinhronizirati podatke. Pri vsakem problemu bomo ugotavljali kakšni so bili standardni pristopi brez orekstracije in kubernetesa in kako se ta problem rešuje v kubernetes okolju z uporabo kubefed.

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 uporanbikov. Naprimer 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.

1.2.2 Viša dosegljivost

Vsakih nekaj let se zgodi da slišimo novico ko iz interneta izpade kakšen podatkovni center. To se lahko zgodi iz več razlogov, najpogostejša pa sta naravna nesreča - vremenski pogoji in napaka na sistemu. Če gre v takšnem primeru za večjega oblačnega 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 aplikaije, 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 potrbno tudi njihovo sinhrono delovanje. Takšni majhni klastri so poleg ze znanega problema prevelike razdalje ponavadi obsojeni, da s centralinimi strežniki komunicirajo samo minimalno, saj zelo pogosto za prenost podatkov uporabljajo draga mobilna omrežja.

1.2.5 Razdeljevanje dela po različnih lokacijah

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

Poglavje 2

Koncepti

2.1 Kubernetes

Leta 2014 je Google objavil in odprl kodo od projekta Kubernetes [**?**]. 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 Cloud Native Computing Fundation.

2.2 Osnovni pojmi v kubernetesu

Kubernetes predstavi koncept poda, ki

Poglavje 3

Priprava

3.1 Raspberry PI 4

Za namene testiranja različnih načinov povezovanja kubernetes klastrov moramo najprej postaviti nekaj kubernetes klastrov. 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.

3.2 K3S in K3OS

Obstaja več implementacij kubernetesa in mi bomo uproabili z viri varčno odprotokodno implementacijo K3S od podjetja Rancher. Hkrati so v podjetju Rancher pripravili že pripravljeno distribucijo operacijskega sistema linux K3OS, ki jo lahko namestimo na katerikoli računalnik. Majhna težava se pojavi, ker še ni pripravljene uradne verzije operacijskega sistema za ploščice Raspberry PI. A k sreči se je v ta namen začel odprtokodni projekt PiCl k3os

image generator, ki nam iz slik operacijskih sistemov K3OS in Raspberry OS in konfiguracijskih datorek zgradi novo sliko operaijskega sistema za naš Raspberry PI. Konfiguracijske datoteke, ki jih moramo priložiti so standardne YAML datoteke, ki jih podpira K3OS. Vanje zapišemo nastavitve kot so ssh javni ključi za dostop, podatki od WiFi omrežja na katerega se povezujemo, geslo, žeton za povezavo z kubernetes klastrom in način v katerem želimo zagnati K3S na sistemu. V našem primeru smo vse K3S programe zagnali v strežniškem načinu in nobenega v načinu delovnega vozlišča, saj želimo, da vsak Raspberry PI predstavlja svoj klaster.

3.3 Demonstracijska spletna aplikacija

Za potrebe testiranja sem moral naredili novo testno aplikacijo. Ker se to delo želi posebej osredotočiti na realne probleme v industriji, sem nujno potreboval preprosto aplikacijo, ki premore tudi shranjevanje podatkov v podatkovno bazo. Aplikacija dostopna v javnem Docker repozitoriju je narejena v programskem jeziku Go, saj z njim lahko zgradimo res majhno Docker sliko, kar pa je pri testiranju pomembno, da ne izgubimo preveč časa na čakanju. Aplikacija deluje preprosto. Na podatnih mrežnih vratiih izpostavi vmesnik REST z dvema preprostima 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. 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 api in nam kode za prehod med bazami ni potrebon spreminjati.

3.4 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. Namestimo ga v razdelek kube-federation-system. Ker je izdelek še v razvoju in še ni prišel iz alfa faze nimajo objavljene verzije za arm procesorje. Zato sem si moral sam zgraditi svojo Docker sliko in sem jo objavil na hub.docker.com. Lahko pa sem uporabil originalni Helm zapis strukture in sem samo spremenil Docker sliko iz originalne na mojo. Za delo z kubefed namestitvijo v klastru pa sem moral na svoj računalnik namestiti orodje za komandno vrstico kubefedcli. Z uporabo ukaza kubefedctl join povežemo vse tri klastre v kubefed sistem. S tem smo uspešmo povezali več kubernetes klastrov.

3.5 Postavitev lokalnih klastrov

TODO: Ipiji so ipd... TODO: SHEMA lokalna INFRASTRUKRUa

Poglavje 4

Povezovanje med podatkovnimi centri

4.1 Problem velike latence

Najbolj standarden industrijski primer aplikacije je spletna aplikacija, ki oranjati stranje. Ko neko podjetje lastnik aplikacije poseže po globalnem trgu zelo hitro ugotovijo, da stranke, ki niso blizu podatkovnega centra precej dlje čakajo pred ekrani, da se naloži njihova spletna aplikacija in njihovi podatki. Takšen problem se v splošnem rešuje tako, da našo aplikacijo postavimo še na dodaten strežnik bližje uporabniku. Rešitev se sliši preprosta, a vseeno se tu srečamo z zelo zahtevnimi problemi v računalništvo. Najbolj očiten primer je sinhronizacija podatkov. V našem primeru bomo uporabili dokaj novo alternativo standardnim sql podatkovnim bazam CrateDB, ki ima veliko boljšo podporo za sinhornizacijo podatkov med vozlišči, hkrati pa nam kompatibilnost z apijem podatkovne baze postgres omogoča, da naših aplikacij ni potrebno konceptualno spreminjati.

4.2 Povečanje dosegljivosti aplikacije

Če je čim višja dosegljivost za našo aplikacijo kritičnega pomena in smo že poskrbeli za visoko dosegljivost aplikaije v našem kubernetes klastru še vedo lahko pride do situacije, ko iz omrežja izpade cel podatkovni center. Spomnimo, da je ena izmed zahtev, da kubernetes dela učinkovito to, da so strežniki blizu skupaj. In v primeru naravnih nesreč ali hujših vremenskih pogojev pomeni, da je nedosegljiv cel podatkovi center. Če uporabljamo oblak in gremo še korak dlje z zagotavljanjem dosegljivosti pa nam niti ni dovolj, da uporabimo različne podatkovne centre, ki jih ponujajo oblačni ponudniki ampak ne zaupamo niti temu, da bo ves čas dosegljiv ponudnik pa lahko postavimo naše klastre pri več različnih ponudnikih. Tu kubernetes pride zelo do izraza, saj kljub nekaj ne enakostim skozi implementacije ohranja enak api in je zato takšna postavitev precej lažja, kot bi bila brez uporabe kubernetesa.

4.3 Povezovanje kubernetes klastrov med podatkovnimi centri

Rešitev za oba problema je identična. Postaviti moramo kubernetes 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.

4.4 Razporeditev uporabnikov po klastrih

Ko imamo na vsakem klastrih javno izpostavljen Service preko z uporabo NodePort ali loadbalancerja in urejene ingress zapise moramo še vedno uporabnike preusmeriti na njim najbljižji klaster. Uporabnike lahko mi usmerimo avtomatsko z DNS zapisi, ki omogočajo usmerjanje na podlai geolo-

kacije. Lahko uporabimo in namestimo zunanju dns skozi kubernetes ali pa kar ročno. Slika: Ustvarjanje geolakacijskega A recorda na ROUTE53 V naših lokalnih testnih klastrih bomo ta korak preskočili in jih ne bomo usmerjali preko javnih dns strežnikov, saj v lokalnem okolju to ni smiselno. Naslednja možnost pa je rešitev, ki se jo zelo pogosto poslužujejo spletne računalniške igre, da so naši strežniki popolnoma ločeni in se vsak uporabnik sam odloči na katerem klastru želi igrati. V takšnih primerih se ponavadi tudi izognemmo problemu sinhronizacije podatkov med strežniki, kar zelo poenostavi upravljanje našega programa.

4.5 Infrastruktura primera

V našem primeru spletne aplikacije bomo imeli v vsakem klastru en Deployment z rašim rest apijem - stateful rest sample in en pripadajoči service tipa LoadBalancer, da ga izpostavimo izven klastra. Za podatkovno bazo pa bomo uporabili StatefulSet s CrateDB, PVC diskom na lokalni sd kartici internim servicom za prepoznavo ostalih instanc CrateDB v klastru in service, ki izpostavi podatkovno bazo izven klastra za sinhronizacijo. Slika: infrastruktura aplikacije Na strani https TODO je dostopna konfiguracija za kubernetes infrastrukturo aplikacije. Postavimo jo z ukazom.

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

Takoj preverimo, ce aplikacija deluje in lahko podatke zapisujemo v bazo, tako da prek demo aplikacije poizkusimo dodati uporabnika in izpisati vse uporabnike. To naredimo z curl komando.

```
curl -X POST TODO:8080/users --data '{"name": "John", "lastname": "Doe"}'
curl localhost:8080/users
```

4.6 Implementacija infrastrukture 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 tipe service, deployment in statefulset. Vklopimo jih z naslenjim ukazom.

```
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 statefulset crate
kubefedctl federate service crate-internal TODO
```

Po preizkusu delovanje z curl komando opazimo, da podatki niso sinhornizirani. Uporabniki, ki jih vnesemo v en klaster se še ne sinhronizirajo v ozadnju. Na tej točki se ustavij nekatere spletne igre in prepustijo izbiro strežnika oziroma klastra kar uporabniku.

```
curl -s -X POST 192.168.10.111/users|jq --data '{"name": "John", "lastname": "Doe"}
curl -s 192.168.10.112/users|jq
[]
```

4.7 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.

4.7.1 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 vozlišla 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 ki jih baza podpira za zmanjšanje prometa in zagotavljanje željenje dosegljivosti med klastri. Podobne načine sinhorinzaije podpira tudi naprimer podatkovna baza Cassandra. https://docs.datastax.com/e oss/2.2/cassandra/initialize/initMultipleDS.html

4.7.2 Podatke sinhroniziramo sami

Sinhronizacija podatkovne baze tažak problem in če ne uporabimo primerne podatkone 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 uproabo 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":
curl -s 192.168.10.112/users|jq
[{"Name":"John","Lastname":"Doe"}]
```

Poglavje 5

Upravljanje izoliranih aplikacij

Poglavje 6

Upravljanje klastrov na robu oblaka

Poglavje 7

Sklepne ugotovitve

Uporaba LaTeXa in BibTeXa 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 LaTeXu 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 LaTeXa ogromno.

Literatura

[1] Michael Riis Andersen, Thomas Jensen, Pavel Lisouski, Anders Krogh Mortensen, Mikkel Kragh Hansen, Torben Gregersen, and Peter Ahrendt. Kinect depth sensor evaluation for computer vision applications. Technical report, Department of Engineering, Aarhus University, 2012.

[2] Andreja Balon. Vizualizacija. Diplomska naloga, Fakulteta za elektrotehniko in računalništvo, Univerza v Ljubljani, 1990.

[3] Donald knuth. Dosegljivo: `https://sl.wikipedia.org/wiki/Donald_Knuth`. [Dostopano: 1. 10. 2016].

[4] Donald E Knuth and Peter B Bendix. Simple word problems in universal algebras. In Jörg H. Siekmann and Graham Wrightson, editors, *Automation of Reasoning: Classical papers on computational logic 1957–1966*, pages 342–376. Springer, 1983.

[5] Leslie Lamport. *LaTEX: A Document Preparation System*. Addison-Wesley, 1986.

[6] Tobias Oetiker, Hubert Partl, Irene Hyna, and Elisabeth Schlegl. Ne najkrajši uvod v LaTeX2ϵ. Dosegljivo: `http://www-lp.fmf.uni-lj.si/plestenjak/vaje/latex/lshort.pdf`, 2006. [Dostopano: 1. 10. 2016].

[7] Oren Patashnik. BibTeXing. Dosegljivo: `http://bibtexml.sourceforge.net/btxdoc.pdf`, 1988. [Dostopano 5. 6. 2016].

[8] PDF/A. Dosegljivo: http://en.wikipedia.org/wiki/PDF/A, 2005. [Dostopano: 5. 6. 2016].

[9] Peter Peer and Borut Batagelj. Art—a perfect testbed for computer vision related research. In *Recent Advances in Multimedia Signal Processing and Communications*, pages 611–629. Springer, 2009.

[10] Franc Solina. 15 seconds of fame. *Leonardo*, 37(2):105–110, 2004.

[11] Franc Solina. Light fountain—an interactive art installation. Dosegljivo: https://youtu.be/CS6x-QwJywg, 2015. [Dostopano: 9. 10. 2015].

[12] Matjaž Gams (ured.). DIS slovarček, slovar računalniških izrazov, verzija 2.1.70. Dosegljivo: http://dis-slovarcek.ijs.si. [Dostopano: 1. 10. 2016].