Primer Optimizacije stroškov v Kubernetes Cluster-ju na Amazon AWS

Milan Nikolić
Fakulteta za matematiko, naravoslovje in informacijske tehnologije
Univerza na Primorskem)
Koper, Slovenija
milan.nikolic.f@gmail.com

***Abstract*—TODO: Write propper abstract**
***Index Terms*—Kubernetes, HPA, VPA, optimization, cost, monitoring, open-source, networking**

I. Uvod

Kubernetes je odprtokodni orkestrator za nameščanje aplikacij v vsebnikih (ang. containers). Prvotno ga je razvil Google, navdih pa so mu dale desetletne izkušnje z uvajanjem skalabilnih in zanesljivih sistemov v vsebnikih prek aplikacijsko usmerjenih API-jev. [1]

A. Kubernetes

Kubernetes je v svojem bistvu zmogljiv sistem za upravljanje aplikacij v cloud-native okolju. Ima dva glavna namena: deluje kot cluster za izvajanje aplikacij in kot orkestrator za upravljanje mikrostoritev. V cluster Kubernetes so aplikacije porazdeljene po več vozliščih (ang. nodes), celoten sistem pa upravlja centralizirana nadzorna raven (ang. control plane). Nadzorna ravnina opravlja naloge, kot so načrtovanje delovnih obremenitev, spremljanje zdravja sistema in skaliranje aplikacij, medtem ko vozlišča skrbijo za dejansko izvajanje teh delovnih obremenitev.

Kubernetes lahko v marsičem razumemo kot možgane in mišice sodobne infrastrukture. Nadzorna raven ali možgani" zagotavljajo učinkovito izvajanje ključnih funkcij, kot sta samodejno skaliranje in posodobitve brez zastojev, medtem ko vozlišča, mišice", izvajajo vsakodnevne operacije izvajanja programske kode. Z avtomatizacijo ključnih vidikov upravljanja aplikacij Kubernetes omogoča večjo prilagodljivost in učinkovitost, zlasti v dinamičnih okoljih v oblaku.

Kubernetes deluje tudi kot orkestrator, ki skrbi za učinkovito namestitev in upravljanje mikrostoritev. Različne posamezne storitve, kot so spletni strežniki, avtentikacijski moduli in komponente za ohranjanje podatkov, organizira v celovito aplikacijo. Ta orkestracija omogoča, da se Kubernetes dinamično odziva na spremembe in dogodke ter samodejno prilagaja dodeljevanje resource-ov (ang. resource allocation) in konfiguracijo za ohranjanje nemotenega delovanja, ko se sistem razvija v realnem času.

Zagon aplikacij v cluster-ju Kubernetes poteka po doslednem vzorcu: razvijalci napišejo mikrostoritve, jih zapakirajo v container-e in jih namestijo v cluster. Kubernetes zagotavlja, da aplikacija deluje, kot je bilo pričakovano, kjer so opredeljena želena stanja, Kubernetes pa prevzame odgovornost za vzdrževanje teh stanj. Če se obnašanje aplikacije oddalji od deklariranega, Kubernetes poskuša to popraviti.

Čeprav Kubernetes ponuja impresivno funkcionalnost za orkestriranje aplikacij, lahko njegova kompleksnost povzroči izzive pri upravljanju stroškov. Ko organizacije širijo svoje cluster-je Kubernetes na platformah, kot je AWS, lahko prevelike rezervacije, neučinkovito dodeljevanje resource-ov in nenadzorovano širjenje povečajo stroške. Ta članek obravnava te izzive s prikazom praktičnih tehnik optimizacije stroškov z uporabo orodij, kot so samodejni skalirniki, Kubecost, kube-resource-report, vtičniki VPC CNI in spot instance, vse v okolju Kubernetes na AWS.

II. Izzivi pri optimizaciji stroškov v Kubernetes-u na platformi AWS

Kubernetes ponuja zmogljive orkestracijske zmogljivosti, vendar lahko uporaba Kubernetesa v okoljih v oblaku, zlasti v AWS, povzroči hitro naraščajoče stroške, če se ne upravlja učinkovito. Razumevanje ključnih dejavnikov, ki povzročajo stroške, je ključnega pomena za razvoj in izvajanje strategij optimizacije. V tem razdelku so opredeljeni glavni dejavniki, ki prispevajo k stroškom pri uvajanju Kubernetesa na AWS, in obravnavani možni pristopi za zmanjšanje stroškov.

A. Cloud-Native Stroškovni Nosilci

Preučimo glavne elemente, ki vplivajo na stroške sistema Kubernetes, vključno z računskimi viri, shranjevanjem, omrežnimi stroški in licenciranje in podporo. [3]

1) Compute Resources: Eden glavnih dejavnikov stroškov v Kubernetes je poraba compute resource-ov, vključno s procesorjem in pomnilnikom. Ti viri so dodeljeni kontejnerskim aplikacijam, ki se izvajajo v več vozliščih. Posebni stroški, povezani z compute resource-i, se razlikujejo glede na izbrane vrste instanc in način njihovega upravljanja.

- **Vrste Instanc**: AWS ponuja različne vrste instanc, optimiziranih za različne delovne obremenitve. Splošno namenske instance so pogosto primerne za uravnotežene delovne obremenitve, medtem ko so računsko optimizirane instance namenjene nalogam, ki zahtevajo veliko procesorja. Izbira ustrezne vrste instance za vsako delovno obremenitev lahko prinese znatne prihranke pri stroških.

- **Spot Instance**: Uporabnikom omogočajo, da se potegujejo za proste zmogljivosti v oblaku, so lahko bistveno cenejše od instanc na zahtevo. Ti so še posebej uporabni za nekritične, na napake odporne delovne obremenitve, kot sta paketna obdelava in obsežna analiza podatkov, pri katerih je mogoče prihraniti do 90 % stroškov.
- **Dinamično Skaliranje**: Kubernetes omogoča samodejno skaliranje resource-ov glede na povpraševanje v realnem času z Horizontal Pod Autoscaler (HPA) in Vertical Pod Autoscaler (VPA). Z dinamičnim skaliranjem aplikacij v času konic in zmanjšanjem dodeljevanja resource-ov, ko se povpraševanje zmanjša, lahko organizacije optimizirajo stroške.

2) Shranjevanje (ang. Storage): Tudi shranjevanje je lahko pomemben dejavnik pri stroških sistema Kubernetes, zlasti ko gre za trajne volumne in shranjevanje podatkovnih baz.

- **Persistent Volumes**: Z njimi zagotovimo, da so podatki na voljo tudi ob okvari ali ponovnem zagonu pod-a. Vendar so stroški persistent volum-a zelo odvisni od vrste uporabljenega pomnilnika. Solid-state diski (SSD) na primer zagotavljajo večjo zmogljivost, vendar so dražji od običajnih trdih diskov (HDD). Izbira ustreznega razreda shranjevanja za vsak primer uporabe pomaga optimizirati stroške.
- **Podatkovne Baze**: Upravljane podatkobne baze pogosto zagotavljajo stroškovno učinkovitejšo rešitev v primerjavi s samostojnimi podatkovnimi bazami v cluster-ju Kubernetes. Poleg tega lahko strategije za optimizacijo shranjevanja, kot sta stiskanje podatkov in brisanje zastarelih podatkov, še dodatno zmanjšajo stroške shranjevanja.

3) Stroški Omrežja: Omrežni stroški se lahko hitro povečajo, zlasti pri aplikacijah, ki vključujejo veliko podatkovnega prometa ali delujejo v več regijah.

- **Prenos Podatkov**: Prenos podatkov znotraj cluster-ja in med njimi ali med cluster in zunanjimi storitvami je povezan z dodatnimi stroški. Zmanjšanje nepotrebnih prenosov podatkov in optimizacija pretoka podatkov po gruči lahko pomagata zmanjšati te stroške.
- **Network Policies**: Z izvajanjem network policy-ev za omejevanje prometa med pod-i le na potrebne komunikacije se lahko zmanjšajo stroški omrežja.

Za učinkovito upravljanje stroškov Kubernetesa v sistemu AWS morajo organizacije sprejeti več najboljših praks, med drugim:

- **Pravilna velikost resource-ov**: Redno pregledovanje in prilagajanje dodeljenih sredstev, da se zagotovi, da delovne obremenitve tečejo na primerkih ustrezne velikosti, s čimer se izognemo premajhni in preveliki rezervaciji.
- **Orodja za spremljanje in optimizacijo**: Rešitve, kot sta Kubecost in AWS Cost Explorer, omogočajo vpogled v izkoriščenost resource-ov in razčlenitev stroškov, kar ekipam omogoča prepoznavanje področij za izboljšave in nenehno optimizacijo porabe.
- **Avtomatizacija**: Z orodji Kubernetes-a, kot je Cluster Autoscaler, je mogoče avtomatizirati skaliranje node-ov glede na potrebe delovnih obremenitev, zagotoviti učinkovito dodeljevanje resource-ov in preprečiti, da bi nedelujoči node-i povzročali nepotrebne stroške.

III. Tehnike za optimizacijo stroškov

A. Horizontal Pod Autoscaler

HPA je ključna funkcija v Kubernetes-u, ki pomaga upravljati skaliranje aplikacij s prilagajanjem števila pod-ov glede na spreminjajoče resource potrebe. Deluje tako, da spremlja izkoriščenost resource-ov, kot sta procesor ali pomnilnik, in dinamično povečuje ali zmanjšuje število pod-ov na podlagi opazovanih metrik. Ta avtomatiziran pristop k skaliranju zagotavlja učinkovito uporabo virov, hkrati pa ohranja zmogljivost aplikacij.

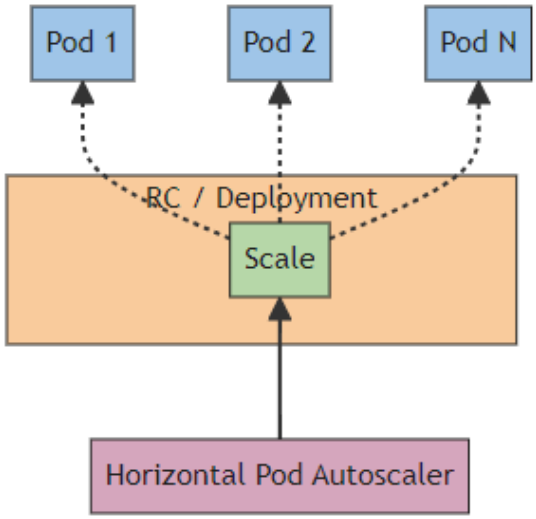

Fig. 1. Horizontal Pod Autoscaler [4]

HPA deluje prek kontrolne zanke v kontrolni ravnini sistema in redno poizveduje po metrikah, da oceni porabo virov. HPA se zanaša na strežnik metrike (metric server), ki zbira podatke v realnem času iz pod-ov v cluster-ju. Krmilnik HPA primerja trenutno izkoriščenost resource-ov s ciljno izkoriščenostjo, določeno v konfiguraciji. Na podlagi te primerjave krmilnik sprejme odločitev o skaliranju - doda več pod-ov, če je izkoriščenost nad ciljno vrednostjo, ali zmanjša število pod-ov, če je izkoriščenost pod pragom.

V tipičnih konfiguracijah krmilnik HPA uporablja metrike, kot sta izkoriščenost procesorja ali pomnilnika, vendar ga je mogoče razširiti tudi na lastne ali zunanje metrike, kar omogoča večjo prilagodljivost pri odločitvah o skaliranju.

Pri konfiguriranju HPA je treba določiti:

- **Ciljna izkoriščenost resource-ov**: Opredelitev želenega praga uporabe procesorja ali pomnilnika.
- **Najmanjše in največje število pod-ov**: Določanje meja za skaliranje, ki zagotavlja, da se aplikacija skalira v določenem območju.
- **Metrike za spremljanje**: CPU, pomnilnik ali metrike po meri, odvisno od zahtev aplikacije.

HPA se lahko na primer konfigurira tako, da vzdržuje ciljno 50-odstotno izkoriščenost procesorja in za dosego tega cilja poveča ali zmanjša število strokov v okviru določenih omejitev.

```
kind: HorizontalPodAutoscaler
metadata:
 name: my-app-hpa
spec:
 scaleTargetRef:
 apiVersion: apps/v1
 kind: Deployment
 name: my-app
 minReplicas: 1
 maxReplicas: 10
 metrics:
 - type: Resource
 resource:
 name: cpu
 target:
 type: Utilization
 averageUtilization: 50
```

B. Vertical Pod Autoscaler

Za razliko od HPA, ki aplikacije skalira z dodajanjem ali odstranjevanjem pod-ov, VPA spreminja zahteve po resource-ih posameznih pod-ov, da se zagotovi, da niso niti preveč niti premalo oskrbljeni. Ta zmožnost pomaga izboljšati splošno izkoriščenost resource-ov v cluster-ju, zato je bistvena tehnika za optimizacijo stroškov v Kubernetes.

VPA deluje na podlagi več ključnih elementov:

- **Zbiranje metrik**: Podobno kot HPA se VPA pri ocenjevanju uporabe resource-ov zanaša na metrike iz cluster-ja. Nenehno spremlja porabo procesorja in pomnilnika v pod-ih ter na podlagi teh podatkov priporoča optimalne zahteve za resource-e.
- **Priporočilni mehanizem**: VPA ima priporočilni mehanizem, ki ustvarja predloge za dodelitev resource-ov na podlagi pretekle uporabe pod-ov. Ta priporočila so shranjena v objektu VPA in jih je mogoče uporabiti samodejno ali ročno.
- **Proces samodejnega skaliranja**: Ko je VPA nastavljen na način samodejno“, samodejno posodablja zahteve po resource-ih pod-ov v realnem času. Če pod porablja več resource-ov, kot jih zahteva, VPA poveča njegovo dodelitev, če pa pod zahteva preveč resource-ov, zmanjša obseg, da sprosti neuporabljene resource-e.

VPA je konfiguriran z objektom CRD (Custom Resource Definition), ki določa:

- **Izbira podov**: Določa, kateri pod-i naj bodo vertikalno samodejno skalirani.
- **Resource Policy**: Določa, kako VPA izračunava zahteve po resource-ih.
- **Politika posodabljanja**: Določa, ali se posodobitve izvajajo samodejno ali ročno.
-

VPA vključuje tudi nadzornik sprejema VPA, ki prestreže zahteve za ustvarjanje pod-ov in prilagodi njihove zahteve po resource-ih na podlagi priporočil, ki jih zagotovi VPA. To zagotavlja, da se novi pod-i zaženejo z ustreznimi dodeljenimi viri.

```
apiVersion: autoscaling.k8s.io/v1
kind: VerticalPodAutoscaler
metadata:
 name: my-app-vpa
spec:
 targetRef:
 apiVersion: apps/v1
 kind: Deployment
 name: my-app
 updatePolicy:
 updateMode: Aut
```

IV. Orodja za spremljanje in preglednost stroškov

Za učinkovito upravljanje stroškov v Kubernetes cluster-ju so poleg strategij optimizacije potrebne tudi možnosti za spremljanje uporabe resource-ov in preglednost nad tem, kje nastajajo stroški. Brez preglednosti nad porabo resource-ov in morebitnimi neučinkovitostmi organizacije tvegajo, da bodo pretirano zagotavljale resource-e, jih premalo izkoriščale in na koncu povečale operativne stroške. V tem poglavju so predstavljena različna orodja, namenjena zagotavljanju vpogleda v uporabo resource-ov in pripisovanje stroškov, kar ekipam omogoča sprejemanje premišljenih odločitev o optimizaciji cluster-ja.

A. AWS Cost Explorer

AWS Cost Explorer je enostaven za uporabo vmesnik, ki uporabnikom omogoča vizualizacijo, razumevanje in upravljanje stroškov in uporabe AWS skozi čas. [5] S funkcijami, kot so izbira časa, filtriranje in združevanje, se lahko uporabniki poglobijo v določene vire, da ugotovijo stroškovne dejavnike, odkrijejo nepravilnosti in preučijo vzorce porabe na podrobni ravni.

Za organizacije, ki uporabljajo delovne obremenitve (workloads) na AWS, je Cost Explorer dragocen vir za sledenje in upravljanje stroškov. Zagotavlja podrobne razčlenitve stroškov, vključno z možnostmi za prikaz stroškov na ravni ur in virov, kar uporabnikom pomaga razumeti, katera sredstva povzročajo večino stroškov. Ta raven vpogleda je lahko še posebej uporabna pri upravljanju obsežnih okolij v oblaku z različno uporabo virov v več storitvah.

Vendar ima AWS Cost Explorer omejitve pri spremljanju Kubernetes cluster-jev. Cost Explorer sicer lahko zagotovi vpogled v stroške na ravni vozlišč, vendar ne upošteva celotnega spektra stroškov, povezanih s Kubernetes-om. Stroški, povezani z upravljanjem cluster-ja, operacijami nadzorne ravnine, uporabo diskov, izravnavo obremenitve in omrežjem, niso neposredno zajeti na način, ki bi odražal kompleksnost okolja. Poleg tega sledenja stroškov na ravni posameznega

pod-a ali imenskega prostora (namespace) – kar je pogosta zahteva v Kubernetes-u - ni mogoče preprosto doseči z možnostmi poročanja na visoki ravni Cost Explorer-ju.

Zaradi teh omejitev Cost Explorer ni zelo uporaben pri spremljanju cluster-ja, zato orodja, ki so posebej zasnovana za spremljanje stroškov Kubernetes-a, kot sta Kubecost ali kube-resource-report, ponujajo bolj prilagojen vpogled v uporabo resource-ov, stroške na ravni pod-ov in morebitne prihranke v cluster-ju.

B. Kube-Resource-Report

Kube-Resource-Report je orodje, namenjeno zagotavljanju preglednosti uporabe resource-ov in pomoči organizacijam pri optimizaciji zahtevkov za resource-e za zmanjšanje neučinkovitosti, znane tudi pod imenom slack“. Slack predstavlja razliko med resource-ov, zahtevanimi za aplikacijo, in dejansko porabljenimi resource-i, kar povzroča neizkoriščene in zapravljene zmogljivosti, ki še vedno povzročajo stroške. Orodje identificira in izračuna nezadostno izkoriščenost procesorja in pomnilnika v cluster-ju, skupinah in aplikacijah ter predstavi potencialne prihranke stroškov v dolarskih vrednostih.

Orodje zbira in združuje podatke iz različnih cluster-jev, ocenjuje stroške node-ov (za okolja AWS in GCP), zbira podatke o uporabi resource-ov na ravni pod-ov in celo vključuje podatke iz VPA, da zagotovi natančnejša priporočila glede resource-ov. Z analizo teh podatkov Kube-Resource-Report izpostavi področja, na katerih je mogoče prilagoditi zahteve po virih, da se bolj uskladijo z dejansko uporabo, s čimer se zmanjšajo zaostanki in optimizirajo izdatki v oblaku.

Spodaj, slike 2 in 3, sta pregled stroška cluster-ja na nivoju node-a in cluster-ja, pred optimizacijo:

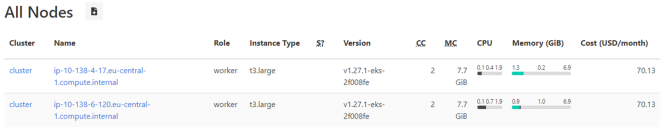

Fig. 2. Kube-resource-report: Node-i pred optimizacijo

Fig. 3. Kube-resource-report: Cluster pred optimizacijo

Vozlišča v cluster-ju so tipa t3.large in to je default tip, ki je priporočen tudi v dokumentaciji AWS, vendar v tem primeru ne uporabljamo vseh resource-ov. Od možnih 7 GB pomnilnika uporabljamo 1,3, od dveh procesorjev pa le 0,1. Očitno je, da imamo prevelik tip instance za naše potrebe.

V naslednjem koraku vidimo, da če zmanjšamo velikost instance na t3.small, zmanjšamo tudi stroške za 75%.

Nato si oglejmo posamezne deployment-e in njihove namespace-e.

Fig. 4. Kube-resource-report: Node-i pred optimizacijo

Fig. 5. Kube-resource-report: Cluster pred optimizacijo

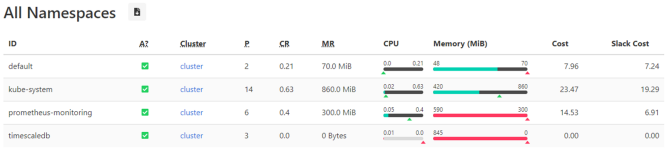

Fig. 6. Kube-resource-report: Namespace-i pred optimizacijo

Iz zgornje slike je razvidno, da nekateri deployment-i nimajo dovolj resource-ov (rdeče označene), druge pa imajo preveč resource-ov in imajo slack stroške.

To lahko popravimo z uporabo že omejenih tehnik - HPA in VPA.

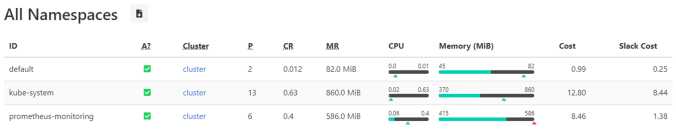

Fig. 7. Kube-resource-report: Namespace-i, optimizirani

V primerjavi z izpisom iz slike 6, prihranki so očitni.

Z uporabo Kube-Resource-Report lahko organizacije pridobijo dragocen vpogled v to, kako se njihovi resourc-i uporabljajo in kje so priložnosti za prihranek stroškov. Ekipam pomaga optimizirati njihove cluster-je z zmanjšanjem prostih resource-ov in zagotavljanjem, da so dodelitve resource-ov ustrezno usklajene z dejanskimi zahtevami aplikacij, kar neposredno prispeva k učinkovitejšemu in stroškovno učinkovitejšemu delovanju Kubernetesa.

C. Kubecost

Kubecost je rešitev za spremljanje in zmanjšanje stroškov v Kubernetes cluster-ju.

Uporabnikom omogoča, da si z namestitvijo, ki traja le nekaj minut, ogledajo naslednje podatke:

- Razporejanje stroškov v realnem času po vseh ključnih konceptih Kuberntes-a, npr. poraba imenskega prostora (namespace-a), deployment-a, service-a, pod-ov, itd..
- Razporeditev stroškov z nastavljivimi oznakami za merjenje porabe po lastnikih, skupinah, oddelkih, izdelkih itd.
- Dinamično določanje cen, omogočeno z integracijo z AWS

- Metrike stroškov za CPU, GPU, pomnilnik in shranjevanje
- Pregled stroškov oblaka zunaj cluster-ja povezani z lastnikom, npr. S3 bucket-i in instance RDS, dodeljene podom/namestitvi.
- Izvoz podatkov za nadaljnjo analizo.

Kubecost ponuja podrobnejši pregled stroškov, povezanih s Kubernetes cluster-jem, kot poročilo kube-resource-report. Na spodnjih slikah lahko vidimo, da nam Kubecost omogoča filtriranje in združevanje pogleda stroškov na podlagi vseh low-level komponent Kubernetes cluster-ja. Takšna granularnost je ključnega pomena pri velikih in zapletenih cluster-jih. Uporabniku omogoča, da ne poskrbi le za upravljanje stroškov, temveč tudi za to, da imajo vse komponente in deployment-i dovolj dodeljenih resource-ov za pravilno delovanje.

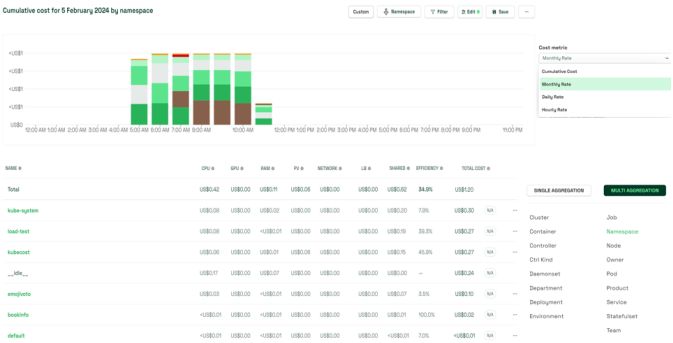

Fig. 8. Kubecost: Nadzorna Plošča

Uporabnik ima lahko vnaprej konfiguriranih in shranjenih več nadzornih plošč, da ima hiter pregled nad pomembnimi stroški in resource-i.

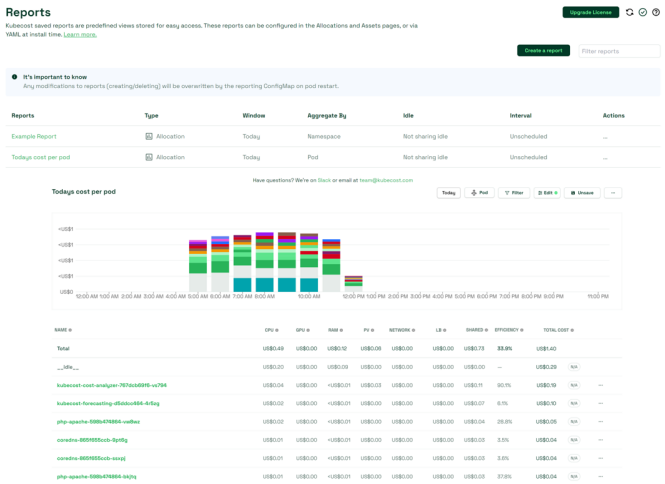

Fig. 9. Kubecost: Pročila

Poleg vnaprej pripravljenih nadzornih plošč Kubecost omogoča tudi samodejno izdelavo poročil. Poročila so soobli-kovana s strani uporabnika in prikazujejo podrobno analizo stroškov. Prav tako jih je mogoče načrtovati, da se generirajo v uporabniško določenem intervalu, in jih poslati odgovorni osebi ali ekipi.

Poleg tega, da Kubecost zagotavlja izjemno natančnost pri poročanju o stroških, analizira tudi cluster in zagotavlja priporočila za optimizacijo cluster-ja za vsako komponento Kubernetes.

Na spodnji, sliki 10, je prikazan seznam vseh priporočil, ki jih lahko pripravi Kubecost, in koliko denarja lahko prihranimo z vsako spremembo.

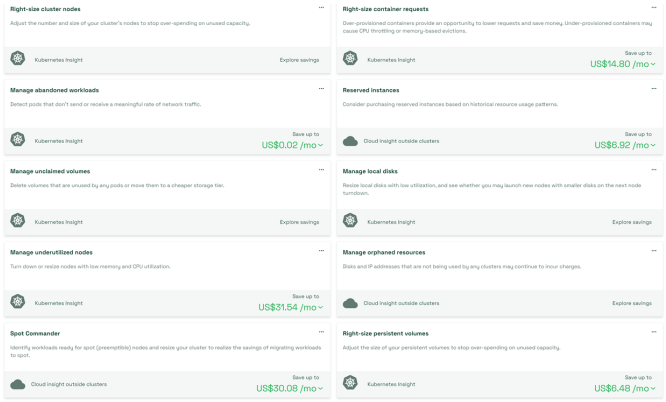

Fig. 10. Kubecost: Priporočila

Poglabljanje v eno od priporočil, v tem primeru za persistant volume-e. Vidimo lahko, da je prikazano, katere točno volume-e moramo konfigurirati, kakšna je njihova trenutna največja kapaciteta, kakšno zmogljivost priporoča, kaj imamo trenutno konfigurirano, približni stroški po konfiguraciji, trenutni stroški in približni prihranki.

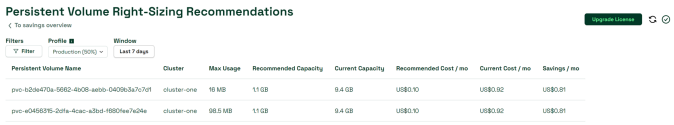

Fig. 11. Kubecost: Priporočilo za Persistent Volume

Kubecostova priporočila so vezana na ponudnike storitev v oblaku. Ker je naš cluster nameščen na AWS, lahko spodaj vidimo, da so priporočila za velikost cluster-ja podana z uporabo vrst instanc, ki so na voljo na AWS.

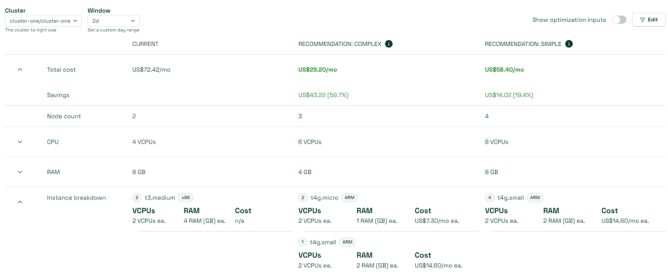

Fig. 12. Kubecost: Priporočilo za velikost in tip instanc

To kaže tudi na podrobnosti, s katerimi Kubecost podaja priporočila. Dve priporočili, eno, ki ga je preprosto izvesti,

drugo, ki je bolj zapleteno. Za vsako je podan podroben seznam vrst instanc, števila vozlišč, CPU, RAM in prihrankov, ki jih lahko dosežemo.

Na splošno je Kubecost vsestransko in zelo uporabno orodje. Za vsako veliko gručo je bistvenega pomena, da spremlja ne le stroške, temveč tudi dodeljevanje virov.

V. Spot Instanca

Spot instance so edinstvena ponudba ponudnikov storitev v oblaku, kot so AWS, Azure in Google Cloud, ki vam omogočajo, da ponudite neuporabljene računalniške zmogljivosti po znatno nižjih cenah. [3]

Za razliko od standardnih instanc na zahtevo so spot instance na voljo po bistveno nižjih cenah, saj uporabljajo neizkoriščene zmogljivosti oblaka. Vendar pa lahko ponudnik oblaka te instance prekine z majhnim opozorilom, zato so idealne za delovne obremenitve, ki so odporne proti napakam, brez stanja ali so sposobne prenašati prekinitve, kot so paketna opravila (batch jobs), analize podatkov in naloge strojnega učenja.

Prednosti uporabe spot instanc

- Spot instance so do 90 % cenejše od instanc na zahtevo, zato so zelo privlačna možnost za organizacije, ki se zavedajo stroškov.
- Zaradi nizkih stroškov spot instance omogočajo veliko skalabilnost. Organizacije si lahko privoščijo uporabo več resource-ov v obdobjih največjega povpraševanja ali pri obsežnih izračunih, ki bi bili pri uporabi instanc na zahtevo (on-demand) predragi. To je še posebej uporabno za podatkovno intenzivne aplikacije, strojno učenje in obdelavo velikih količin podatkov.

Kubecost na edinstven način prikazuje tudi priporočila za spot instance. Določi delovne obremenitve, ki so primerne, in jih oceni glede na njihovo pripravnost.

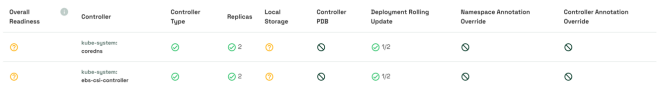

Fig. 13. Kubecost: Priporočilo za spot instance

VI. Implementacija

Obe orodji sta precej preprosti za namestitev, zahtevata minimalno ali nikakršno konfiguracijo in sta zelo dobro dokumentirani. Spodaj sta dva odlomka kode za instalacijo orodja z uporabo Helm Package Manager-ja, za nadaljnja navodila se obrnite na dokumentacijo.

A. Kube-resource-report

Za bolj podrono dokumentacijo github strana: https://codeberg.org/hjacobs/kube-resource-report

```
$ git clone https://codeberg.org/hjacobs
  /kube-resource-report
$ cd kube-resource-report
$ helm install --name kube-resource-report
  ./unsupported/chart/kube-resource-report
$ helm status kube-resource-report
```

Zgornje 4 komande se izvajajo v terminal-u, in preuzamejo že pripravljen helm chart kube-resource-report, ter prestavi terminal v kube-resource-report folder, instalira z uporabo helm-a in na konsu izpiše status.

B. Kubecost

Za bolj podrono dokumentacijo Kubecost strana: https://docs.kubecost.com/install-and-configure/install

```
$ helm upgrade --install kubecost \
  --repo https://kubecost.github.io/
        cost-analyzer/ cost-analyzer \
  --namespace kubecost --create-namespace
```

Zgornje komanda se izvaja v terminal-u, in instalira Kubecost z uporabo helm-a.

VII. Zaključek

V tem raziskovalnem seminarju smo raziskali različne metode in orodja za optimizacijo stroškov v Kubernetes cluster-ju, ki gostujejo v oblaku AWS. Preučili smo ključne tehnike samodejnega skaliranja, kot sta Horizontal Pod Autoscaler (HPA) in Vertical Pod Autoscaler (VPA), ki imata ključno vlogo pri dinamičnem prilagajanju resource-ov, da bi zadostili povpraševanju in hkrati zmanjšali stroške.

Poleg tega orodja, kot so AWS Cost Explorer, Kube-resource-report in Kubecost, ponujajo dragocen vpogled v uporabo resource-ov in pomagajo ugotoviti neučinkovitost. AWS Cost Explorer omogoča širok pregled porabe v oblaku, vendar nima dovolj podrobnih podatkov, ki so potrebni za delovne obremenitve Kubernetes-a. To vrzel zapolnjujejo orodja, namenjena Kubernetes-u, kot sta Kube-resource-report in Kubecost, ki ponujata podroben pregled uporabe vseh low-level komponent, kar omogoča natančnejšo optimizacijo stroškov.

Če primerjamo Kube-resource-report in Kubecost, Kube-resource-report ne ponuja granularnosti kot Kubecost, vendar ima prednosti v svoji preprostosti. Poleg tega lahko Kubecost ponudi vpoglede in priporočila, ki jih Kube-resource-report ne more. Za začasne, testne ali majhne cluster-je Kube-resource-report zadostuje, za velike produkcijske cluster-je pa je Kubecost bistveno orodje in je veliko zmogljivejši.

References

[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,” ACM Queue 14 (2016): 70–93.

[2] Nigel Poulton, and Pushkar Joglekar, “The Kubernetes Book,“ Lean Pub 2017 - 2020

[3] Anirudh Mustyala1, and Sumanth Tatineni, “Cost Optimization Strategies for Kubernetes Deployments in Cloud Environments“, ESP Journal of Engineering & Technology Advancements ISSN: 2583-2646 / Volume 1 Issue 1, September, 2021 / Page No: 34-46

[4] Kubernetes Documentation, “Horizontal Pod Autoscaling“, found at https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/

[5] AWS Documentation, “AWS Cost Explorer“, found at https://aws.amazon.com/aws-cost-management/aws-cost-explorer/