KARELIA-AMMATTIKORKEAKOULU
Tietojenkäsittelyn koulutusohjelma

Petri Karvinen

SKAALAUTUVAT JA JAETTAVAT TIEDOSTOJÄRJESTELMÄT GOOGLE CLOUD PLATFORM:SSA

Opinnäytetyö
Lokakuu 2019

Karelia AMMATTIKORKEAKOULU	**OPINNÄYTETYÖ** **Lokakuu 2019** **Tietojenkäsittelyn koulutus** Tikkarinne 9 80200 JOENSUU +350 13 260 600 (vaihde)

Tekijä(t)
Petri Karvinen

Nimeke
Skaalautuvat ja jaettavat tiedostojärjestelmät Google Cloud Platform:ssa

Toimeksiantaja
Pilvia Oy

Opinnäytetyön tavoitteena oli tarkastella erilaisia jaettavia levyjärjestelmiä, saada nämä tarkastelussa olevat levyjärjestelmät toimimaan Google Cloud Platform -alustalla sekä vertailla niiden suorituskykyä. Pääasiallisena tarkastelun kohteena oli levyjärjestelmien käyttöönotto. Levyjärjestelmien suorituskyvyn testaaminen sekä levyjärjestelmiin liittyvä teoriapuoli pidettiin suhteellisen kevyenä.

Vertailtavat levyjärjestelmät olivat Ceph, GlusterFS ja NFS, joita ajettiin Google Cloud Platformissa virtuaalipalvelimilla. Ceph ja GlusterFS julkaistiin Kubernetes-klusteriin ja NFS yksittäiselle palvelimelle. Luodut levyjärjestelmät liitettiin ulkoiseen, GKE:n hallinnoimaan Kubernetes-klusteriin ja levyjärjestelmien suorituskykyä vertailtiin Linux:n ‘dd’ -komennolla.

Opinnäytetyössä tarkastellut levyjärjestelmät saatiin otettua onnistuneesti käyttöön ja niille suoritettiin yksinkertaiset kirjoitusnopeustestit. Testien perusteella GlusterFS ja Ceph toimivat skaalautuvissa järjestelmissä NFS:ää paremmin, joskin näiden järjestelmien käyttöönotto ja hallinta oli huomattavasti NFS:ää monimutkaisempaa.

Kieli Suomi	Sivuja 37 Liitteet 2 Liitesivumäärä 3

Asiasanat
Opinnäytetyö, tiedostojärjestelmät, Google Cloud, Ceph, NFS, GlusterFS

Karelia
AMMATTIKORKEAKOULU

THESIS
October 2019
Degree Programme in business IT

Tikkarinne 9
FI 80200 JOENSUU
FINLAND
+350 13 260 600 (switchboard)

Author(s)
Petri Karvinen

Title
Scalable and shareable file systems at Google Cloud Platform

Commissioned by
Pilvia Oy

The goal of this thesis was to configure a variety of shareable file systems on Google Cloud Platform and to compare their features and performance. The main focus was on the technical implementation of these file systems, including instructions for setting up both the environment and the file systems themselves. Performance testing and file system theory were kept light due to the already expansive scope of the thesis.

The three file systems compared were Ceph, GlusterFS and NFS, which were run on virtual machines at Google Cloud Platform. Ceph and GlusterFS were built on top of a Kubernetes-cluster, while NFS remained a standalone server. The created file systems were mounted into an external, GKE-managed Kubernetes cluster, and their performance was benchmarked using Linux's 'dd' -command.

All three file systems were successfully deployed and their performance was tested via rudimentary writing speed tests. GlusterFS and Ceph were benchmarked to be faster than NFS, while their setup and maintenance was found to be significantly more difficult.

Language
Finnish

Pages 37
Appendices 2
Pages of Appendices 3

Keywords
Thesis, File systems, Google Cloud, Ceph, NFS, GlusterFS

Sisältö

1 Johdanto

Opinnäytetyön tarkoituksena oli vertailla skaalautuvia ja jaettavia tiedostojärjestelmiä, ja sitä kuinka hyvin ne sopivat sovelluskäyttöön Google Compute Enginessä.

Jaettava tiedostojärjestelmä tarkoittaa jonkinlaista teknologiakokonaisuutta, jonka avulla fyysinen tai virtualisoitu levy voidaan jakaa useisiin eri palvelimiin luku- sekä kirjoituskäyttöön. Skaalautuvuudella tässä yhteydessä tarkoitetaan taas levytilan dynaamista kasvattamista ilman palvelukatkoksia.

Levytilan jakaminen useisiin palvelimiin on nykypäivänä lisääntyvässä määrin tärkeää johtuen hajautettujen järjestelmien kasvaneesta suosiosta.

2 Käytetyt termit ja pohjatieto

Tässä osiossa käydään läpi opinnäytetyössä ilmaantuvia termejä ja asioita, jotka voivat vaatia lisäselvitystä. Termit käydään läpi ainoastaan opinnäytetyöhön liittyvien asioiden kannalta.

Klusteri:
Klusteri (eng. Cluster) tarkoittaa useista palvelimista muodostettua kokonaisuutta. Klusteria hallinnoidaan jonkinlaisella palvelimiin asennetulla ohjelmistolla, jonka tarjoamien viestikanavien ja hallintatyökalujen avulla useat koneet pystyvät toimimaan jonkin tarpeen täyttävänä, yhtenä kokonaisuutena (Wikipedia 2019).

Solmu:
Solmu (eng. Node) on yksi klusteriin kuuluvista palvelimista.

Kontti:

Kontti (eng. Container) on klusterin sisälle virtualisoitu, eristetty ympäristö. Konteilla ei peruskäytössä ole suoraa pääsyä sitä suorittavan solmu-koneen ytimeen, vaan konttien suoritus tapahtuu kokonaisuudessaan käyttäjä-avaruudessa. Konttien tarkoituksena on paketoida koodi helppokäyttöiseksi ja siirreltäväksi paketiksi jonka suoritus onnistuu alustasta riippumatta (Docker 2019).

GCP:

GCP on lyhenne sanasta Google Cloud Platform. GCP on Googlen tarjoama pilvialusta ohjelmisto- ja arkkitehtuurikehitykseen.

GKE:

GKE on lyhenne sanasta Google Kubernetes Engine. GKE on GCP:n tarjoama, valmis hallintaympäristö Kubernetes-klusterille. GKE automatisoi monia toimenpiteitä Kubernetesin hallinnan suhteen, muun muassa uusien solmukoneiden luomisen ja päivittämisen (Google 2019).

Kubernetes:

Kubernetes on konttien suorittamiseen kehitetty järjestelmä, joka mahdollistaa applikaatioiden suorittamisen hajautetussa ympäristössä.

3 Kubernetes:n asentaminen

Osa tässä opinnäytetyössä tarkasteltavista levyjärjestelmistä hyödynsi Kubernetes-klusteria omiin toiminnallisuuksiinsa. GCP:stä löytyi valmis integraatio Kubernetesille, nimeltään GKE. Levyjärjestelmiä käytettäessä oli Kubernetes kuitenkin parempi asentaa GKE:n ulkopuolelle, koska GKE on suunniteltu enemmän hiukan dynaamiseksi ja eläväksi kokonaisuudeksi, johon lisätään ja josta poistetaan palvelimia vapaasti (Google 2019).

Levyjärjestelmät vaativat vakaan ja tukevan pohjan sekä levyillä olevan että palvelimelta löytyvän tiedon suhteen. Tämänlainen toiminnallisuus oli huomattavasti turvallisempaa toteuttaa, kun Kubernetes oli asennettu yksittäisistä virtuaalipalvelimista muodostettuun kokonaisuuteen eikä GKE:n hallinnan alle.

Kubernetesin käyttöönotto aloitettiin luomalla GCP:hen sille alustaksi palvelimet.

gcloud beta compute --project=$PROJECT_ID instances create datacluster-master datacluster1 datacluster2 --zone=$ZONE --machine-type=n1-standard-2 --image=ubuntu-1604-xenial-v20181114 --image-project=ubuntu-os-cloud --boot-disk-size=10G --boot-disk-type=pd-ssd --metadata-from-file startup-script=startup.sh

Ylläoleva gcloud-komentorivityökalun komento loi GCP:hen kolme n1-standard-2 luokan konetta käyttämällä pohjakuvana Ubuntu Linuxia. Komennossa käytetty käynnistyskoodinpätkä startup.sh löytyy tämän opinnäytetyön liitteestä numero 1. Koodinpätkä asensi jokaiseen luotuun palvelimeen tarvittavat työkalut Kubernetesin käyttöönottoa varten.

Komennon lisäksi datacluster1 ja datacluster2 koneisiin liitettiin myös 500gb levyt. Tämä pystyttiin suorittamaan joko komentoriviltä, tai GCP:n käyttöliittymästä käsin.

Koneiden luomisen ja alustamisen jälkeen Kubernetes täytyi vielä ottaa käyttöön. Prosessin sai aluilleen ottamalla SSH-yhteys datacluster-master -koneeseen komennolla:

gcloud compute ssh datacluster-master --project $PROJECT_ID --zone $ZONE

Koneen sisällä suoritettiin opinnäytetyön liitteestä 2 löytyvä scripta, joka latasi Kubernetesin master-koneeseen tarvittavat työkalut ja alusti klusterin käyttöön. Scripta palautti komentoriville myös komennon, jolla datacluster1 ja datacluster2 koneet saatiin liitettyä tähän Kubernetes-klusteriin.

Scriptan palauttama klusteriin liittymiskomento oli tämän mallinen:

kubeadm join $IP:6443 --token $TOKEN --discovery-token-ca-cert-hash $HASH

Komento tuli ajaa sekä datacluster1 että datacluster2 koneissa. Tämän jälkeen Kubernetes ympäristö oli käyttökelpoinen ja sen tilannetta pystyttiin tarkastelemaan master-koneessa suorittamalla kubectl-komentoja. Koko klusterin tilanne tässä vaiheessa näytti tältä:

```
root@datacluster-master:~# kubectl get node
NAME                STATUS   ROLES    AGE     VERSION
datacluster-master   Ready    master   2m24s   v1.13.3
datacluster1         Ready    <none>   114s    v1.13.3
datacluster2         Ready    <none>   105s    v1.13.3
root@datacluster-master:~#
```

Kuva 1. Klusterin solmukoneiden tilanne heti käyttöönoton jälkeen.

4 Tarkasteltavat tiedostojärjestelmät

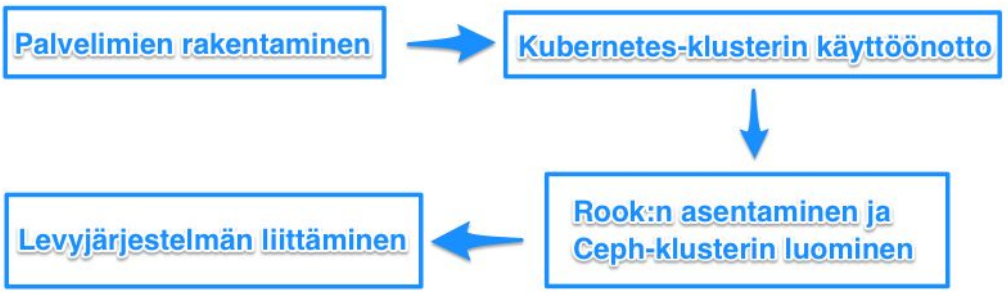
Tässä osiossa esitellään opinnäytetyön aikana kokeillut tiedostojärjestelmät jotka liitettiin ulkoiseen Kubernetes-klusteriin.

Huomionarvoisena asiana on, että vaikka tiedostojärjestelmät asennettiin toimimaan niitä hyödyntävän klusterin ulkopuolisena kokonaisuutena, tiedostoklusterit / tiedostopalvelimet jakoivat silti verkkoyhteydet tuon klusterin kanssa. Tästä syystä käytössä on sekä GCP:n tarjoamia nimipalveluita palvelinosoitteiden selvittämiseen, sekä sisäisiä IP-avaruuksia yhteyden muodostamiseen.

4.1 Ceph

Ceph on vuodesta 2006 asti kehitteillä ollut tallennustila-alusta, joka tarjoaa lohko-, objekti- sekä tiedostojärjestelmäkohtaista tallennusteknologiaa. Tämän opinnäytetyön osalta järjestelmän tarkastelu keskittyi ainoastaan tiedostojärjestelmäkohtaiseen tallennustilaan sen jaettavuuden ja monipuolisuuden takia (Ceph 2019b).

Ceph tiedostojärjestelmän käyttöönottoon Google Compute Enginessä löytyi useita vaihtoehtoja. Tässä opinnäytetyössä vaihtoehdoista tarkasteltiin Rook-nimistä sovelluskehystä.



Kuva 2. Ceph-levyjärjestelmän käyttöönotto.

4.1.1 Rook:n käyttöönotto

Rook on Kubernetesille kehitetty sovelluskehys, joka tarjoaa julkaisuominaisuudet monille eri tallennusjärjestelmille. Tässä opinnäytetyössä tutkittiin ainoastaan Rookin käyttöä Ceph:n suhteen.

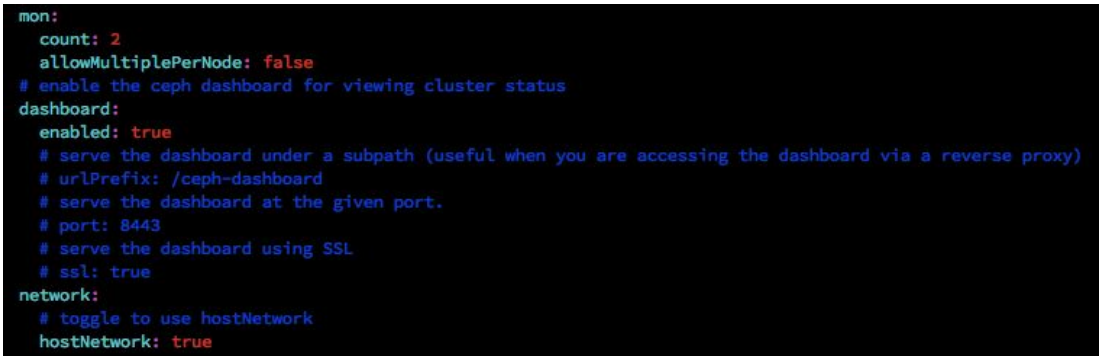
Rookin avulla Ceph on mahdollista julkaista suoraan käytössä olevaan klusteriin omaksi kokonaisuudekseen tai vaihtoehtoisesti omaksi erilliseksi klusterikseen, johon toinen klusteri ottaa yhteyden. Lähtökohtaisesti konfiguraatio on molemmille tekniikoille varsin samanlainen ja ero näiden tekniikoiden välille tulee lähinnä levyilmentymän liittämistekniikasta kontteihin.

Vaikka Rookin asentaminen suoraan olemassaolevaan, GKE:n hallinnoimaan klusteriin on mahdollista, kyseinen toteutustapa on hiukan riskialtis koska GKE:n hallinnoima Kubernetes-klusteri on hallintaominaisuuksiltaan rajoitetumpi, varsinkin palvelimiin liitettävien levyjen suhteen. Tästä syystä Rookille luotiin GCP:hen oma, GKE:n ulkopuolinen klusteri, jonne manuaalisesti asennettiin Kubernetes. Ohjeet Kubernetesin manuaaliasennukseen ovat esiteltynä tämän opinnäytetyön kohdassa 3.

Kubernetes-klusterin käyttöönoton jälkeen Rook saatiin asennettua klusteriin hakemalla ensimmäiseksi kopio Rook:n Git-repositoriosta

(https://github.com/rook/rook). Komennon “kubectl apply -f $FILENAME” avulla klusteriin saatiin asennettua kansiosta ./cluster/examples/kubernetes/ceph löytyvien tiedostojen common.yaml, operator.yaml, cluster.yaml sekä filesystem.yaml sisällöt (Rook 2019a).

Ennen cluster.yaml-tiedoston asentamista tuli tiedostosta muokata Ceph:n asetuksia erilaiseksi, koska halusimme liittää klusterin levyksi itse klusterin ulkopuolelta. Ceph:n monitorien määrää jouduttiin myös pudottamaan, koska käytössä oli ainoastaan kaksi tiedostopalvelinta. Huomionarvoisena asiana on, että tuotantokäytössä parillinen monitorien määrä ei ole Ceph:n kanssa suositeltu asia, koska monitorit pitävät huolen Ceph-klusterin tilan terveydestä enemmistöperiaatteella: Mikäli käytössä on vain kaksi monitoria, toisen näistä pettäessä koko klusterin tila on epäterve (Ceph 2019a). Testitilanteessa, kuten tässä opinnäytetyössä, tämä ei kuitenkaan ole kriittinen asia.



Kuva 3. Cluster.yaml-tiedoston muutokset.

Ennen filesystem.yaml-tiedoston asentamista, tiedostosta muokattiin replikointiasetuksia pienemmäksi, koska ylläpitämässämme tiedostoklusterissa oli tällä hetkellä vain kaksi solmukonetta joista datalevyt löytyivät.

```yaml
apiVersion: ceph.rook.io/v1
kind: CephFilesystem
metadata:
  name: myfs
  namespace: rook-ceph
spec:
  # The metadata pool spec. Must use replication.
  metadataPool:
    replicated:
      size: 2
  # The list of data pool specs. Can use replication or erasure coding.
  dataPools:
    - failureDomain: host
      replicated:
        size: 2
```

Kuva 4. Filesystem.yaml-tiedostoon tehtävät muutokset.

Tässä opinnäytetyössä Rook:n Git-repositoriosta hyödynnettiin versiota 6469b06f0a058603d32c4a0007f0ff88de5c4d90.

4.1.2 Ceph tiedostojärjestelmän liittäminen

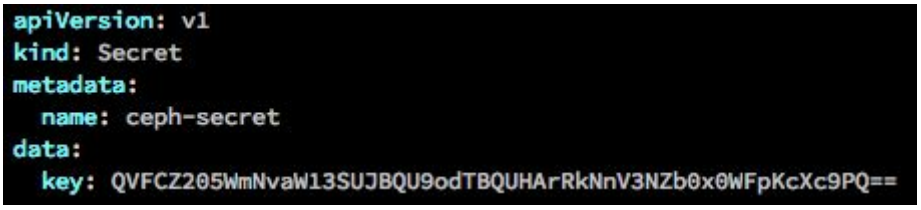
Ceph:n liittämiseen ulkoiseen Kubernetes-klusteriin tarvittiin Ceph-klusterista asiakasavain sekä Ceph:n monitorikonttien IP-osoitteet. Ceph-klusterin avaimen sai helpoiten Ceph:n hallinnan kautta. Rook:ia käytettäessä hallintaa varten tarvittiin erillinen työkalukontti, joka löytyi sen Git-repositoriosta kansiosta ./cluster/examples/kubernetes/ceph, tiedostosta toolbox.yaml. Työkalukontin sai julkaistua ajamalla komento “kubectl apply -f toolbox.yaml” datacluster-master koneessa.

Kontin käynnistyttyä se löytyi rook-ceph nimiavaruudesta. Kontin tilaa pystyi tarkastelemaan komennolla “kubectl get pods -n rook-ceph”, jolla kaikki Rook:n luomat, Ceph:n hallintaan tarkoitetut kontit näkyvät. Työkalu kontti löytyi nimellä rook-ceph-tools-$ID, jossa $ID oli satunnaisgeneroituja merkkejä. Kontin sisälle pääsi komennolla “kubectl -n rook-ceph exec -it rook-ceph-tools-$ID” ja Ceph-klusterin avaimen sai helpoiten suorittamalla komento:

kubectl -n rook-ceph exec rook-ceph-tools-$ID ceph auth export client.admin

Yllä olevalla komennolla Ceph:n asiakasavain tulostui komentoriville muodossa “key = AQBgm9ZcoimwIBAAOhu0PPp+FCgWsYoLtXZJqw==”.

Ulkoiseen klusteriin (joka myöskin pyöri Kubernetes-pohjalla) tuli luoda tämän avaimen perusteella Kubernetes-salausavain, jonne avain talletettiin base64-enkoodattuna. Enkoodauksen sai suoritettua komentoriviltä base64 komentoa hyväksikäyttämällä.



Kuva 5. Ceph-salausavain ulkoiselle klusterille.

Kuvassa 5 näkyvä salausavain saatiin julkaistua klusteriin kubectl:ää hyväksikäyttäen komennolla:

kubectl create -f $FILE.

Liittämistä varten tarvittiin vielä Ceph-monitorien IP-osoitteet. Osoitteet saatiin Cephiä hallinnoivasta Kubernetes-klusterista suorittamalla datacluster-master -koneessa komento:

kubectl get pods -n rook-ceph -o wide | grep ceph-mon | awk '{printf "%s\n",$6}'

Tässä tapauksessa komentoriville tulostui kaksi IP-osoitetta, koska määritimme Ceph-klusterille kaksi monitoria.

Kun salausavain oli luotu ja IP-osoitteet selvitetty, ulkopuoliseen klusteriin voitiin julkaista kontti jonne Ceph-levyjärjestelmä asetettiin näkyviin.

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  labels:
    app: ceph-test
  name: ceph-test
spec:
  selector:
    matchLabels:
      app: ceph-test
  template:
    metadata:
      labels:
        app: ceph-test
    spec:
      containers:
      - args:
        - infinity
        command:
        - sleep
        image: debian
        name: app
        volumeMounts:
        - mountPath: /cephmount
          name: ceph
      volumes:
      - cephfs:
          monitors:
          - 10.132.15.221:6789
          - 10.132.15.222:6789
          secretRef:
            name: ceph-secret
          user: admin
        name: ceph
```

Kuva 6. Ceph tiedostojärjestelmää hyödyntävä Kubernetes-deployment.

Kuvassa 6 kohtaan volumes määriteltiin tyypiksi cephfs, jonne Ceph:n monitorien IP:t listattiin. Salausavaimeksi määritettiin aiemmin ceph-secret -nimellä luotu Ceph:n asiakasavain. Levyjärjestelmä asetettiin näkyviin testikontin sijaintiin /cephmount.

4.1.3 Tilan ja suorituskyvyn lisääminen

Lisää tilaa Rookin hallinnoimaan Ceph-klusteriin saatiin lisäämällä kokoonpanoon uusi osd (Object Storage Daemon). Opinnäytetyössä tähän asti toteutettu Ceph-klusteri koostui kahdesta osd:stä: Klusterissa on kaksi datapalvelinta joista molemmista löytyy yksi datalevy. Jokainen klusteriin lisätty levy ilmentyy Ceph:n puolelle uutena osd:nä.

Uuden osd:n sai klusteriin lisättyä joko kiinnittämällä olemassaolevaan koneeseen uuden levyn tai lisäämällä klusteriin kokonaan uusi solmukone. Uuden koneen lisääminen tapahtui kuten kahden ensimmäisenkin lisääminen opinnäytetyön kohdassa 3.

gcloud beta compute --project=$PROJECT_ID instances create datacluster3 --zone=$ZONE --machine-type=n1-standard-2 --image=ubuntu-1604-xenial-v20181114 --image-project=ubuntu-os-cloud --boot-disk-size=10G --boot-disk-type=pd-ssd --metadata-from-file startup-script=startup.sh

Koneen valmistuttua koneeseen lisättiin jälleen uusi levy. Tämän jälkeen kone piti vielä liittää klusteriin kiinni, joka tapahtui ajamalla datacluster-master koneessa komento:

kubeadm token create --print-join-command

Yllä oleva komento tulosti komentoriville samanlaisen komennon kuin opinnäytetyön kohdan 3 klusterin alustuskomentokin. Komennon suorittamalla datacluster3 koneessa uusi kone saatiin liitettyä klusteriin kiinni. Jotta Rook havaitsi klusteriin liittyneen uuden datalähteen, klusterissa pyörivä “rook-operator” -niminen podi tuli vielä käynnistää uusiksi. Helpoiten tämä onnistui lopettamalla vanha operaattori komennolla:

kubectl delete pod rook-ceph-operator-$ID -n rook-ceph

Tämä käynnisti uuden operaattorin ja hetken kuluttua Rook huomasi uuden koneen sekä koneesta löytyvän uuden levyn ja alusti klusteriin uuden osd:n.

Mikäli halutaan kasvattaa ainoastaan klusterin levykokoa ilman suorituskyvyn lisäämistä, voidaan jo olemassa olevaan solmukoneeseen lisätä uusi levy. Levyn lisäämisen ja rook-operator -podin uudelleenkäynnistämisen jälkeen Rook alustaa klusteriin uuden osd:n samalla tavalla kuin kokonaan uuden koneen lisäämisen jälkeen.

4.1.4 Huomionarvoista asiaa Rook Ceph:stä

Opinnäytetyön edistyessä Rook:n kokoonpanoon tuli melko huomattavia muutoksia. Ensimmäinen versio opinnäytetyöstä hyödynsi Rook:n Git-repositorion versiota ea03048ef16e7487964d570792a62a4873ed73ce.

Tämän version kanssa Ceph-klusterin julkaisu ja hallinnointi toimi täysin identtisesti verrattuna opinnäytetyössä nyt listattuihin ohjeisiin, mutta Rook:n sisäisten päivitysten takia tämä versio ei suoraan enää toiminut näillä ohjeilla. Olisi ollut mahdollista päivittää opinnäytetyössä listatut ohjeet vastaamaan nykypäivää, mutta tämä olisi vaatinut julkaisuun hiukan lisää omia konfiguraatioita. Tästä syystä helpoin tapa oli vaihtaa Rook:sta uudempaan versioon. Päivittäminen on myös lähtökohtaisesti parempi tapa hallinnoida ja ylläpitää järjestelmiä.

Tässä uudemmassa Rook:n julkaisussa on yksi todella mielenkiintoinen muutos verrattuna versioon, joka opinnäytetyön alussa oli tarjolla: uudessa julkaisussa on mahdollista käyttää Kubernetesiin luotuja PVC:itä (Persistent Volume Claim) Rook:n hallinnoiman Ceph-klusterin tilanlähteenä. Käytännössä tämä tarkoittaa sitä, että sen sijaan että klusterin muodostaviin virtuaalikoneisiin liitetään

suoraan itse levyjä kiinni, annetaan Kubernetesin ja Kubernetesiä pyörittävän alustan (tässä tapauksessa GCP) hoitaa tämä operaatio (Rook 2019b).

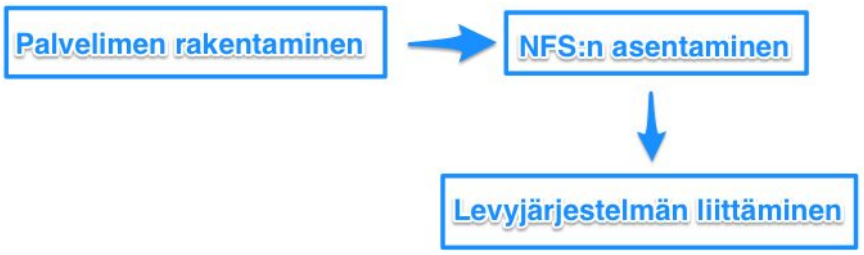
Suurin etu tästä on se, että tällä uudella versiolla on hyvinkin mahdollista julkaista Ceph-klusteri suoraan GKE:n hallinnoimaan Kubernetes-klusteriin, koska PVC:illä hallitut levyt eivät ole samalla tavalla riippuvaisia solmukoneesta ja datan menettäminen ei näin ole enää samalla tavalla helposti mahdollista.

GKE:henkin on tullut muutoksia opinnäytetyön aloituksen jälkeen. Alussa GKE:n hallinnoimiin koneisiin oli vielä mahdollista liittää levyt suoraan GCP:n hallintapaneelista. Tämä ominaisuus oli Ceph-klusterin kannalta jo lähtökohtaisesti huono, ja tämän kirjoituksen aikaan, GKE ei enää tue levyjen suoraa liittämistä GKE:n hallinnoimiin solmukoneisiin. Tästä syystä, Rook:n uuden version tarjoama mahdollisuus PVC:n käyttöön datalähteenä on vieläkin kiinnostavampi, mikäli Ceph-klusterin tarjoamaa dataa halutaan hyödyntää samassa Kubernetes-klusterissa.

4.2 NFS

NFS (Network File System) on tiedostonjakoprotokolla, jonka avulla palvelimelta voi jakaa kansioita lähiverkon kautta. Tarkasteltavista tekniikoista tämä on vanhin, jo vuotena 1984 kehitetty.

Toisin kuin muut vaihtoehdot, NFS ei suoraan tarjoa mahdollisuutta kasvattaa levytilaa tai prosessointitehoa jouhevasti, vaan kyseiset operaatiot vaativat normaalitilanteissa palvelukatkoksen. GCP:tä käytettäessä, levytilan dynaaminen kasvattaminen on mahdollista, koska Google virtualisoi palvelimiin liitettävät levyt.



Kuva 7. NFS levyjärjestelmän käyttöönotto.

4.2.1 Asentaminen ja käyttöönotto

Tekniikan asentaminen Linux-pohjaisille palvelimilla oli erittäin yksinkertaista, koska NFS:ää varten Linuxin paketinhallinnasta löytyivät valmiit työkalut.

```
Warning: Permanently added 'compute.8799383308653026415' (ECDSA) to the list of known hosts.
Linux nfs-server 4.9.0-8-amd64 #1 SMP Debian 4.9.130-2 (2018-10-27) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
petrikarvinen@nfs-server:~$ sudo apt-get update && sudo apt-get install nfs-kernel-server -y
```

Kuva 8. NFS:n asentaminen Debian Linux -palvelimelle.

Asennuksen jälkeen, NFS Kernel Server käynnistyi automaattisesti palveluksi Linux palvelimelle. NFS:n tilaa pystyttiin seuraamaan esimerkiksi systemctl:n avulla, “systemctl status nfs-kernel-server” -komennolla.

Ennen jakoasetusten määrittämistä, palvelimeen lisättiin levy, joka löytyi palvelimelta sijainnista /dev/sdb, ja se kiinnitettiin palvelimen sijaintiin /files. Levyn sai GCP:ssä lisättyä palvelimeen suoraan käyttöliittymästä, jonka jälkeen levy tuli palvelimelta käsin SSH-yhteyden kautta alustaa komennolla:

mkfs.ext4 /dev/sdb

Levyn lisäyksen ja alustuksen jälkeen levy saatiin liitettyä kohteeseen /files editoimalla palvelimen /etc/fstab tiedostoa sisältämään rivi:

/dev/sdb /files ext4 defaults 0 0

ja ajamalla tämän jälkeen komento:

mount /dev/sdb

Jotta NFS jakaisi palvelimelta jotain, täytyi sille määritellä palvelimelta löytyvään /etc/exports -tiedostoon jakoasetukset.

```
# /etc/exports: the access control list for filesystems which may be exported
#               to NFS clients.  See exports(5).
#
# Example for NFSv2 and NFSv3:
# /srv/homes       hostname1(rw,sync,no_subtree_check) hostname2(ro,sync,no_subtree_check)
#
# Example for NFSv4:
# /srv/nfs4        gss/krb5i(rw,sync,fsid=0,crossmnt,no_subtree_check)
# /srv/nfs4/homes  gss/krb5i(rw,sync,no_subtree_check)
#
/files 10.0.0.0/8(rw,no_subtree_check,no_root_squash,fsid=100)
```

Kuva 9. NFS:n jakoasetukset.

Kuvassa 9 esitellyissä jakoasetuksissa, järjestelmän sijainti /files jaettiin koko palvelimen lähiverkolle luku- sekä kirjoitusoikeuksin. Jakoasetusten määrittelyn jälkeen, NFS tuli käynnistää uudelleen, jotta asetukset tulivat voimaan. Tämä saatiin suoritettua komennolla “systemctl restart nfs-kernel-server”.

4.2.2 Tiedostojärjestelmän liittäminen konttiin

Kubernetesistä löytyi suora liitin NFS:lle.

nfs-deployment.yaml ×

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  labels:
    app: testdeployment
  name: testdeployment
spec:
  selector:
    matchLabels:
      app: testdeployment
  template:
    metadata:
      labels:
        app: testdeployment
    spec:
      containers:
      - image: debian
        command:
        - sleep
        args:
        - infinity
        name: testcontainer
        volumeMounts:
        - mountPath: /nfsmount
          name: nfs
      volumes:
      - name: nfs
        nfs:
          path: /files
          server: nfs-server
```

Kuva 10. Kubernetesiin julkaistava “deployment” joka käynnistää Kubernetesiin kontin, minne liitetään nfs-server -palvelimelta jaettu kansiosijainti /files.

Kuvassa 10 on esitelty perusmallinen Kubernetesiin ajettava julkaisu. Kuvasta on karsittu pois kaikki ylimääräiset määrittelyt, jotka eivät itse levyn käsittelyyn tai kontin käynnistämiseen liity. Levy liitettiin Debian-pohjaisen levykuvan suoritukseen sijaintiin /nfsmount.

4.2.3 Tilan ja suorituskyvyn lisääminen

NFS:n suhteen tilan ja suorituskyvyn lisäämisoperaatiot ovat GCP:ssä huomattavasti yksiselitteisempiä ja rajoittuneempia verrattuna muihin levyjärjestelmiin. Koska NFS on yksittäisestä palvelimesta verkon kautta

ulkopuolelle jaettu levy, koneen prosessointitehon kasvattaminen ilman katkoksia ei ole tällä tekniikalla mahdollista: Mikäli palvelin vaatii lisää tehoa on palvelin sammutettava konetyypin vaihtoa varten.

Itse levytilan lisääminen on kuitenkin GCP:ssä mahdollista katkoitta. Itse kiinnitetyn levyn kokoa voi kasvattaa GCP:n käyttöliittymästä valitsemalla uusi koko. Jotta laajennettu levytila näkyi NFS-palvelimella oli koneessa ajettava komento

resize2fs /dev/sdX

Komennon sdX viittaa laajennetun levyn sijaintiin koneessa. Tässä opinnäytetyön tapauksessa sijainti oli /dev/sdb.

Laajennuskomennon suorituksen jälkeen laajennettu levytila oli käytettävissä NFS:n kautta.

4.3 GlusterFS

GlusterFS on avoimella lähdekoodilla kirjoitettu skaalautuva verkkolevyjärjestelmä, joka on suunniteltu dataintensiivisiin operaatioihin.

Kuva 11. GlusterFS -levyjärjestelmän käyttöönotto.

4.3.1 Asentaminen ja käyttöönotto

Tässä opinnäytetyössä GlusterFS otettiin käyttöön hyödyntämällä GitHubista löytyvää gluster-kubernetes repositoriota (Github 2019). Repositorion avulla GlusterFS saatiin julkaistua helposti olemassa olevaan Kubernetes-klusteriin.

Ennen käyttöönottoa, GCP:hen tuli luoda erillinen Kubernetes-klusteri. Klusterin luominen on esitelty tämän opinnäytetyön osiossa 3. Osiosta 3 poiketen, koneiden niminä tässä käyttöönottoesimerkissä käytettiin nimiä gluster-master, gluster1 ja gluster2.

Klusterin luonnin jälkeen gluster1 ja gluster2 koneisiin tarvittiin vielä GlusterFS:ää varten muutamia muutoksia. Koneisiin tuli asentaa glusterfs-client tiedostojärjestelmäoperaatioita varten, sekä aktivoida muutama kernel-moduuli. Operaatiot saatiin suoritettua ottamalla SSH-yhteys gluster1 ja gluster2 koneisiin ja suorittamalla seuraava scripta:

```
# INIT SCRIPT
modprobe dm_mirror
modprobe dm_thin_pool
modprobe dm_snapshot

apt-get install -y software-properties-common
add-apt-repository ppa:gluster/glusterfs-3.12
apt update
apt install -y glusterfs-client
```

Tämän jälkeen hallintaoperaatioita saatiin suoritettua gluster-master koneesta, SSH-yhteyden kautta. Aiemmin mainitusta gluster-kubernetes repositoriosta otettiin klooni /usr/share kansioon:

```
cd /usr/share
```

git clone https://github.com/gluster/gluster-kubernetes

Tässä työssä käytettiin gluster-kubernetes Git-repositorion versiota 7246eb4053c8c5336e4da68d86b76124d435eb3e.

Gluster-kubernetes:iin vaadittiin muutamia muutoksia ennen julkaisua johtuen lähinnä siitä, että tässä työssä käytetty tiedostoklusteri koostui ainoastaan kahdesta datapalvelimesta: Lähtökohtaisesti, kuten Ceph-klusterinkin kanssa, tiedostoklustereille suositeltu minimikoko on kolme.

Kloonatun repositorion kansiosta “deploy” löytyvään ajo scriptaan “gk-deploy” tuli tehdä muutos riville 924 johtuen Kubernetesiin tulleesta muutoksesta, jossa --show-all parametri poistettiin käytöstä. Riviltä tuli poistaa äsken mainittu --show-all merkintä, jonka jälkeen rivi näytti kokonaisuudessaan tältä:

heketi_pod=$(${CLI} get pod --no-headers --selector="heketi" | awk '{print $1}')

Repositorion ./deploy/kube-templates/gluster-daemonset.yaml tiedostosta poistettiin myös käytöstä monitorointi lohkovolyymeihin sekä tarpeettomana että siksi, että kyseinen monitori ei toimi halutulla tavalla. Monitori poistettiin käytöstä asettamalla kontille ympäristömuuttujan GLUSTER_BLOCKD_STATUS_PROBE_ENABLE arvoksi “0”.

```
    spec:
      nodeSelector:
        storagenode: glusterfs
      hostNetwork: true
      containers:
      - image: gluster/gluster-centos:latest
        imagePullPolicy: IfNotPresent
        name: glusterfs
        env:
        # alternative for /dev volumeMount to enable access to *all* devices
        - name: HOST_DEV_DIR
          value: "/mnt/host-dev"
        # set GLUSTER_BLOCKD_STATUS_PROBE_ENABLE to "1" so the
        # readiness/liveness probe validate gluster-blockd as well
        - name: GLUSTER_BLOCKD_STATUS_PROBE_ENABLE
          value: "0"
        - name: GB_GLFS_LRU_COUNT
          value: "15"
        - name: TCMU_LOGDIR
          value: "/var/log/glusterfs/gluster-block"
```

Kuva 12. Blockd -monitoroinnin poistaminen.

Viimeinen operaatio ennen GlusterFS julkaisemista oli sen käyttöön annettavien solmukoneiden määrittely topology.json-tiedostoon. Repositorion kansiosta “deploy” löytyi tiedosto nimeltään topology.json.example jonka pohjalta pystyttiin rakentamaan oma topology.json-tiedosto jossa GlusterFS:lle annettiin kuvaus suoritusympäristöstä.

Ajamalla komento “kubectl get node -o wide” saatiin tulostettua ruudulle Kubernetes-klusterin muodostavien solmukoneiden sisäiset IP-osoitteet alla olevan kuvan näköisesti.

```
root@gluster-master:~# k get no -o wide
NAME             STATUS   ROLES    AGE   VERSION   INTERNAL-IP    EXTERNAL-IP   OS-IMAGE             KERNEL-VERSION    CONTAINER-RUNTIME
gluster-master   Ready    master   49m   v1.15.3   10.132.0.19    <none>        Ubuntu 16.04.5 LTS   4.15.0-1024-gcp   docker://18.6.0
gluster1         Ready    <none>   49m   v1.15.3   10.132.0.14    <none>        Ubuntu 16.04.5 LTS   4.15.0-1024-gcp   docker://18.6.0
gluster2         Ready    <none>   49m   v1.15.3   10.132.0.13    <none>        Ubuntu 16.04.5 LTS   4.15.0-1024-gcp   docker://18.6.0
```

Kuva 13. Kubernetes-ympäristön solmukoneiden sisäiset IP:t.

Komennolla saadut osoitteet, sekä solmukoneiden nimet määriteltiin topology.json-nimiseen tiedostoon, joka tallennettiin kansioon “deploy”.

```
root@gluster-master:/tmp/gluster-kubernetes/deploy# cat topology.json
{
  "clusters": [
    {
      "nodes": [
        {
          "node": {
            "hostnames": {
              "manage": [
                "gluster1"
              ],
              "storage": [
                "10.132.0.14"
              ]
            },
            "zone": 1
          },
          "devices": [
            "/dev/sdb"
          ]
        },
        {
          "node": {
            "hostnames": {
              "manage": [
                "gluster2"
              ],
              "storage": [
                "10.132.0.13"
              ]
            },
            "zone": 1
          },
          "devices": [
            "/dev/sdb"
          ]
        }
      ]
    }
  ]
}
```

Kuva 14. topology.json-tiedoston sisältö.

Tiedostoon määriteltiin myös palvelimilta löytyvät tallennuslaitteet, tässä tapauksessa osion 3 mukaisesti Kubernetes-klusteriin lisätyt 500gb:n datalevyt, jotka palvelimien sisäisesti löytyivät sijainnista /dev/sdb.

Yllä esiteltyjen muutosten jälkeen GlusterFS-klusteri oli viimein julkaisukunnossa. Julkaisu saatiin suoritettua ajamalla gluster-master -koneessa kansiossa /usr/share/gluster-kubernetes/deploy komento:

./gk-deploy -gy --admin-key $ADMIN-KEY --user-key $USER-KEY --single-node

Komennolle tuli määritellä admin- ja käyttäjäavaimet Heketin käyttöön. Heketi on GlusterFS:ään tarjolla oleva REST-pohjainen hallintatyökalu, jonka avulla GlusterFS:ään saa luotua uusia loogisia levyjä ja hallittua näitä. --single-node -parametri taasen rajoitti Heketin tietokantapodien määrää: tässä opinnäytetyössä yksi riitti varsin mainiosti ja klusterin normaalia pienemmästä koosta johtuen julkaisu epäonnistui mikäli tämä parametri puuttui.

Julkaisukomennon suoritettua, GlusterFS:n tilanne näytti tältä:

```
root@gluster-master:/tmp/gluster-kubernetes/deploy# k get po
NAME                      READY   STATUS    RESTARTS   AGE
glusterfs-5brk2           1/1     Running   0          9m57s
glusterfs-8nr5h           1/1     Running   0          9m57s
heketi-75885fc97b-5wdqk   1/1     Running   0          90s
```

Kuva 15. GlusterFS podien tilanne julkaisun jälkeen.

Klusterissa näkyi 2 glusterfs-podia jotka pyörivät gluster1 ja gluster2 solmukoneissa. Tämän lisäksi klusterissa näkyi heketi-niminen hallintapodi jonka avulla GlusterFS:ää voidaan hallita.

GlusterFS:ään tuli vielä luoda looginen levy jotta sieltä saatiin Kubernetesin käyttöön levytilaa. Tämä saatiin tehtyä ajamalla Heketi hallintapodille tämän tyylinen komento:

kubectl exec heketi-75885fc97b-5wdqk -- heketi-cli --user admin --secret $ADMIN-KEY volume create --name=testvolume --size=70 --replica=2

Kubectl exec -komennolle hallintapodin koko nimen sai kuvan 15 mukaisella komennolla. Tässä tapauksessa tämä nimi oli “heketi-75885fc97b-5wdqk”. Komento suoritti hallintakontin sisällä “heketi-cli” -komennon, jolle annettiin

salasanaksi GlusterFS:n julkaisun yhteydessä valittu $ADMIN-KEY. Heketi-cli komento loi GlusterFS:ään loogisen levyn, jonka koko oli 70 gigatavua, ja joka replikoitiin kahden eri solmukoneen välille. Oletuksena replikointi tapahtuisi kolmeen osaan, mutta kuten useassa muussakin kohdassa tämän työn osalta, klusterin pienemmästä koosta johtuen replikointia tuli pienentää.

4.3.2 Tiedostojärjestelmän liittäminen konttiin

GlusterFS:n liittäminen ulkoiseen klusteriin tapahtui ottamalla yhteys Heketin tarjoamiin osoitteisiin. Tästä syystä ulkoiseen klusteriin täytyi luoda dataklusterista löytyvät Kubernetes-endpoint ja Kubernetes-service -rakenteet (Kubernetes 2019).

Gluster-master -koneen sisältä nämä rakenteet saatiin tulostettua ajamalla komento:

kubectl get $TYPE heketi-storage-endpoints -o yaml

Komennossa $TYPE oli joko “endpoints” tai “service”. Komentoriville tulostui näiden avulla dataklusterista löytyvien julkaisuiden versiot. Näitä julkaisuita kuitenkin jouduttiin hiukan editoimaan ennen niiden kopioimista ulkoiseen klusteriin ja tulostuksista tuli editoida pois kaikki julkaisun “staattisuuteen” liittyvät rivit. Esimerkiksi servicestä löytyvä “clusterIP”, jonka Kubernetes antaa service:lle luonnin yhteydessä automaattisesti, tuli poistaa koska klustereiden jakamasta verkosta johtuen kahdella service:llä ei voi olla samaa IP:tä, eikä ulkoiseen klusteriin saa edes luotua service:ä dataklusterin varaamalle IP-alueelle.

```
# Endpoints
apiVersion: v1
kind: Endpoints
metadata:
  name: heketi-storage-endpoints
  namespace: default
subsets:
- addresses:
  - ip: 10.132.0.13
  - ip: 10.132.0.14
  ports:
  - port: 1
    protocol: TCP
---
# Service
apiVersion: v1
kind: Service
metadata:
  name: heketi-storage-endpoints
  namespace: default
spec:
  ports:
  - port: 1
    protocol: TCP
    targetPort: 1
  sessionAffinity: None
  type: ClusterIP
```

Kuva 16. Ulkoiseen klusteriin julkaistavat kopiot Heketin luomista Kubernetes-endpoints ja Kubernetes-service -rakenteista.

Luomalla kuvan 16 kaltaisen tiedoston dataklusterista haetuista rakenteista nämä rakenteet voitiin julkaista ulkoiseen klusteriin komennolla:

kubectl create -f $FILE

Tämän jälkeen GlusterFS:n tarjoama levytila voitiin ottaa käyttöön Kubernetesissä pyörivään podiin GlusterFS-liittimellä.

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: glustertest
spec:
  selector:
    matchLabels:
      app: glustertest
  template:
    metadata:
      labels:
        app: glustertest
    spec:
      containers:
      - args:
        - infinity
        command:
        - sleep
        image: debian
        name: app
        volumeMounts:
        - mountPath: /data
          name: data
      volumes:
      - glusterfs:
          endpoints: heketi-storage-endpoints
          path: testvolume
        name: data
```

Kuva 17. GlusterFS:n liittäminen ulkoiseen klusteriin.

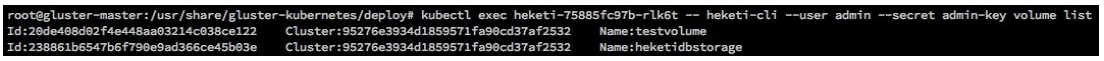
Huomionarvoista kuvassa 17 on GlusterFS-liittimelle määritelty “path”, jonka täytyi vastata GlusterFS:stä löytyvää loogista levyä, joka tässä esimerkissä luotiin nimellä “testvolume”.

4.3.3 Tilan ja suorituskyvyn lisääminen

GlusterFS:n sisältämien loogisten levyjen tilaa saatiin kasvatettua Heketi -hallintapodista käsin. Klusterista tällä hetkellä löytyvät levyt saatiin tulostettua komentoriville gluster-master koneesta suorittamalla Heketi -hallintapodissa komento:

heketi-cli --user admin --secret $ADMIN-KEY volume list

Tässä tapauksessa näytölle tulostui kahden eri loogisen levyn lista.



Kuva 18. GlusterFS:n sisältämät loogiset levyt.

Kuvassa näkyy sekä luomamme “testvolume” että Heketi:n GlusterFS:ään itselleen luoma “heketidbstorage”. Tarkemman tilanteen levystä, tässä tapauksessa meitä kiinnostavasta “testvolume”:sta, sai suorittamalla komennon:

heketi-cli --user admin --secret $ADMIN-KEY volume info $ID

Kyseiseen komentoon laitettava $ID saatiin edellisen listauskomennon avulla. Tässä tapauksessa ID oli 20de408d02f4e448aa03214c038ce122. Komento tulosti ruudulle tietoa levyn koosta, replikointiasetuksista, käytetystä tilasta ja muusta.

Kasvattaaksemme tämän loogisen levyn kokoa, ajettiin komento:

heketi-cli --user admin --secret admin-key volume expand --volume $ID --expand-size 70

Komento lisäsi tässä tapauksessa levylle 70 gigatavua lisää tilaa GlusterFS:lle käytettävissä olevasta määrästä. Kun katsoimme levyn tilanteen uusiksi aiemmin mainitulla info-komennolla, näimme nyt, että loogisen levyn koko oli 140 gigatavua yhteensä.

```
[root@heketi-75885fc97b-rlk6t /]# heketi-cli --user admin --secret admin-key volume info 20de408d02f4e448aa03214c038ce122
Name: testvolume
Size: 140
Volume Id: 20de408d02f4e448aa03214c038ce122
Cluster Id: 95276e3934d1859571fa90cd37af2532
Mount: 10.132.0.26:testvolume
Mount Options: backup-volfile-servers=10.132.0.3
Block: false
Free Size: 0
Reserved Size: 0
Block Hosting Restriction: (none)
Block Volumes: []
Durability Type: replicate
Distribute Count: 2
Replica Count: 2
```

Kuva 19. Yhden GlusterFS:n loogisen levyn tilanne koon kasvattamisen jälkeen.

Loogisten levyjen koon kasvattaminen on tietenkin mahdollista ainoastaan, mikäli GlusterFS:lle tarjolla olevissa levyissä on tilaa. GlusterFS-klusterissa tällä hetkellä olevien levyjen tilanteen pystyi tarkistamaan Heketi -hallintapodista käsin, selvittämällä heketi-cli -komennoilla GlusterFS -klusterin ID:n, tämän klusterin solmukoneiden ID:t ja sitten tarkastelemalla niiden sisältämien levyjen tilannetta. Vaihtoehtoisesti, voitiin tarkastella Heketi:lle sisäänajettua topologiaa, suorittamalla Heketi-podissa komento:

heketi-cli --user admin --secret $ADMIN-KEY topology info

Komento tulosti ruudulle GlusterFS-klusterista erinäistä tietoa. Tilankäytön kannalta oleellisimmat asiat löytyivät infon alalaidasta, kuten kuvasta 20 näkyy.

```
Nodes:

    Node Id: 0eba389b236a126510de87cd6770eef6
    State: online
    Cluster Id: 95276e3934d1859571fa90cd37af2532
    Zone: 1
    Management Hostnames: gluster2
    Storage Hostnames: 10.132.0.26
    Devices:
            Id:15b28e4bdda1d198056b6a145afd3162   State:online    Size (GiB):499     Used (GiB):140     Free (GiB):359
                    Known Paths: /dev/sdb

                    Bricks:
                            Id:9a785ae98df556f4b9ab4c739f3a1b19   Size (GiB):70      Path: /var/lib/heketi/mounts/vg_15b28e4bdda1d198056b6a145afd3162/brick_9a785ae98df556f4b9ab4c739f3a1b19/brick
                            Id:a2e7a8d132d70279141b786b888643fa   Size (GiB):70      Path: /var/lib/heketi/mounts/vg_15b28e4bdda1d198056b6a145afd3162/brick_a2e7a8d132d70279141b786b888643fa/brick

    Node Id: 45f0635583297216ca312f0ca3ff64da
    State: online
    Cluster Id: 95276e3934d1859571fa90cd37af2532
    Zone: 1
    Management Hostnames: gluster1
    Storage Hostnames: 10.132.0.3
    Devices:
            Id:a714589f9c7cea07ab68986163881c00   State:online    Size (GiB):499     Used (GiB):142     Free (GiB):357
                    Known Paths: /dev/sdb

                    Bricks:
                            Id:8ba035b9857293d5ac3a168d03878963   Size (GiB):70      Path: /var/lib/heketi/mounts/vg_a714589f9c7cea07ab68986163881c00/brick_8ba035b9857293d5ac3a168d03878963/brick
                            Id:b8f46a9985c4a3915de54dc525c39ca6   Size (GiB):2       Path: /var/lib/heketi/mounts/vg_a714589f9c7cea07ab68986163881c00/brick_b8f46a9985c4a3915de54dc525c39ca6/brick
                            Id:f77cbdcdf504168e0c395941fbe8712e   Size (GiB):70      Path: /var/lib/heketi/mounts/vg_a714589f9c7cea07ab68986163881c00/brick_f77cbdcdf504168e0c395941fbe8712e/brick
```

Kuva 20. GlusterFS-klusterin sisältämät solmukoneet sekä näiden solmukoneiden käytössä olevien levyjen tilanne.

Jotta GlusterFS-klusteriin saatiin liitettyä uusia levyjä, klusteriin täytyi joko lisätä uusi solmukone tai olemassaoleviin solmukoneisiin tuli lisätä uusi levy.

Koneiden lisääminen tapahtui samalla tavalla kuin opinnäytetyön kohdassa 4.1.3 Ceph-klusteriin lisätyn koneen osalta: GCP:hen luotiin uusi kone johon laitettiin kiinni levy ja joka liitettiin klusteriin kiinni ajamalla koneen sisällä “kubeadm join” komento.

Kun uusi kone oli lisätty Kubernetes-klusteriin, se täytyi vielä lisätä GlusterFS:n kokoonpanoon. Ensiksi uudelle solmukoneelle täytyi asettaa gluster-kubernetes:in vaatima merkintätieto. Tämän sai suoritettua ajamalla gluster-master koneessa komento:

kubectl label node $ID storagenode=glusterfs

Komennon $ID oli uuden koneen nimi Kubernetes-klusterissa. Tämän $ID:n pystyi tarkistamaan komennolla “kubectl get node”. Tässä opinnäytetyössä noudatetun kaavan mukaan, koneen nimi oli “gluster3”.

GlusterFS:n kokoonpano oli esitelty Heketille sisäänladatussa topology.json-tiedostossa, jonka klusteria luodessamme loimme järjestelmään. Uudesta koneesta tuli topology.json-tiedostoon lisätä samat tiedot kuin klusteria luodessakin, eli koneen sisältämät levyt, koneen nimi sekä koneen osoite. Uusi topology.json-tiedosto ladattiin GlusterFS:n käyttöön Heketi-hallintapodista käsin komennolla:

heketi-cli --user admin --secret $ADMIN-KEY topology load --json topology.json

Kun uusi topology.json-tiedosto oli ladattu sisään, GlusterFS:lle näkyvä tila oli päivitetty ja uusi levy oli käytettävissä.

Uuden koneen lisäämisen sijaan oli myös mahdollista lisätä olemassa oleviin solmukoneisiin uusia levyjä. Tämä tapahtui lisäämällä solmukoneeseen uusi levy ja tämän jälkeen päivittämällä topology.json-tiedostoa tämän solmu koneen osalta, lisäämällä koneesta löytyvien levyjen osioon uusi “/dev/sdX” -merkintä osioon “devices”, esimerkiksi:

```
"devices": [
  "/dev/sdb",
  “/dev/sdc”
]
```

Määritetyn levyn sijainti tuli vastata solmukoneeseen lisätyn levyn sijaintia.

4.3.4 Huomionarvoista asiaa GlusterFS:stä

Opinnäytetyössä hyödynnetty Github-repositorio gluster-kubernetes ei ole enää saanut päivityksiä useaan kuukauteen. Tämä mahdollisesti viittaa siihen, että kyseistä projektia ei enää kehitetä / ylläpidetä, joten tämän järjestelmän käyttöönottoon käyttäen tätä repositoriota on suhtauduttava varauksella.

Kubernetesiin tulevat muutokset voivat vaikuttaa järjestelmän toimivuuteen, ja mikäli klusteria halutaan päivittää, tämä saattaa vaatia käyttäjältä oma-aloitteisia toimenpiteitä ja erikoisosaamista. Jo opinnäytetyön kirjoittamisen aikana tulleet muutokset vaikuttivat käyttöönottoon sen verran, että jouduin tekemään muutoksia julkaisutiedostoihin ja käyttöönotto-ohjeisiin.

5 Suorituskykyvertailut

Tässä kohdassa kaikkien esiteltyjen tiedostojärjestelmätekniikoiden suorituskykyä vertailtiin. Tiedostojärjestelmien testausta varten GCP:hen pystytettiin Kubernetes-pohjainen testiympäristö vakioidulla kalustolla. Testiympäristöön laitettiin pyörimään kontti / kontteja, joihin vertailtavat tiedostojärjestelmät liitettiin.

Ellei toisin mainita, kaikki testiympäristössä käytetyt koneet olivat Googlen n1-standard-2 -kokoluokan palvelimia, joka tarkoittaa 2:llä prosessorilla varustettuja koneita. Kaikki testipalvelimet käyttivät Linux-pohjaista käyttöjärjestelmää.

Huomionarvoisena asiana testaukseen liittyen oli se, että Googlen verkkoyhteyksien nopeus on suhteellinen testikoneiden kokoon. Pitämällä testikoneet pääsääntöisesti samankokoisina pyrittiin poistamaan verkkoyhteyksien nopeuden vaikutusta testituloksiin, mutta tilanteesta riippuen tämäntyylinen testausmetodi saattaa vääristää tuloksia, koska verkkoyhteyden nopeuden vaikutus eri tiedostojärjestelmäteknologioihin on erilainen.

5.1 Yksittäiset kirjoitustestit

Kirjoitusnopeuden testaukseen käytettiin Linux:sta löytyvää dd-komentoa. Linux:n dd-komento on tiedostojen lukemiseen, kirjoittamiseen ja kopioimiseen kehitetty järjestelmätyökalu. Työkalulla on mahdollista lukea bittivirtaa jostain lähteestä tietyin parametrein ja kirjoittaa luetut bitit jonnekin (GeeksForGeeks 2019).

Tässä kohdassa opinnäytetyötä dd-komentoa käytettiin rakennettujen tiedostojärjestelmien nopeuden testaamiseen yksittäisten kirjoitusoperaatioiden

näkökulmasta. Komennolla luettiin nolla dataa /dev/zero lähteestä ja kirjoitettiin tämä data levylle: mitä korkeampi kirjoitusnopeus oli, sen tehokkaammin levyjärjestelmä toimii yksittäisen kirjoitusprosessin näkökulmasta.

Testeissä käytetty ympäristö josta käsin tiedostopalvelimeen / tiedostoklusteriin otettiin yhteys oli GKE:ssä sijaitsevassa klusterissa oleva n1-standard-1 kokoluokan palvelin, eli tämän kirjoituksen aikaan 1:n prosessoriytimen virtualisoitu kone 3.75:n gigatavun muistilla.

Yksittäisille testeille testikomentona käytettiin:

dd if=/dev/zero of=./testfile bs=1G count=1

Komento kirjoitti yhden gigatavun kokoisen tiedoston nimeltään “testfile” levylle lukien sen Linux järjestelmän nollasijainnista /dev/null.

Testit toistettiin jokaiselle levyjärjestelmälle 10 kertaa peräkkäin yksinkertaisella for-loopilla ja tuloksista otettiin keskiarvo. Testit suoritettiin sekä tuhoamalla testien välissä kirjoitettu tiedosto, että jättämällä tiedosto paikalleen jolloin levyjärjestelmästä riippuen voitiin hyödyntää välimuistia. Ylikirjoitustesti ajettiin kahteen kertaan, kerran aiemmin mainitulla komennolla ja uudestaan vielä lisäämällä komennolle oflag=direct -parametri. Kyseinen parametri ei verkon kautta toimiville levyjärjestelmille ole kovinkaan kuvaava, koska parametrilla pyritään ohittamaan tietoliikenneyhteyksien vaikutus levyn nopeuteen. Koska tarkasteltavana oli juuri verkkoyhteyksien kautta toimivia levyjärjestelmiä, jotka ensiksi kirjoittavat tiedon välimuistiin ja lopuksi levylle, itse levyjärjestelmän suorituskykyä tämä tieto ei kuvaa kovinkaan tarkasti.

Yksittäisille testeille tulokseksi saatiin alla oleva taulukko.

Taulukko 1. Levyjärjestelmien nopeudet yksittäisten kirjoitus operaatioiden osalta Linux:n dd-komennon avulla.

Levyjärjestelmä	Uusi tiedosto 10x	Ylikirjoitus 10x	Ylikirjoitus 10x oflag=direct
CEPH	110.0 MB/s	593.6 MB/s	68.6 MB/s
NFS	110.1 MB/s	109.2 MB/s	100.9 MB/s
GlusterFS	144.2 MB/s	141.3 MB/s	53.7 MB/s

Testeistä nähtiin, että yksittäisten kirjoitusoperaatioiden suhteen GlusterFS ja Ceph pärjäsivät kumpikin paremmin kuin NFS. GlusterFS tarjosi suhteellisen tasapainoisen nopeuden riippumatta kuorman laadusta. Ceph taas hyödynsi olemassaolevaa dataa välimuistina erittäin tehokkaasti. Tositilanteessa data ei olisi yhtä tasapaksua kuin palvelimelta luettava puhdas nolladata, joten Ceph:n kuusinkertaiseen ylikirjoitusnopeuteen on suhtauduttava varauksella: tosielämäntilanteissa palvelimelle kirjoitettava ja siellä muokattava data olisi huomattavasti monivivahteisempaa. Testistä kuitenkin näkyi Ceph:n tehokkuus tällä saralla ja varsinkin tiedostojen muokkausoperaatiot olivat tällä tiedostojärjestelmällä tehokkaita.

Kuten aiemmin mainittiin, oflag=direct testit eivät suoraan kuvanneet itse jaetun tiedostojärjestelmän tehokkuutta tässä tapauksessa, koska nämä testit valmistuivat vasta, kun data oli välimuistista kirjoitettu itse levylle. Varsinkin Ceph:n ja GlusterFS:n suhteen datan tallennus välimuistiin oli riittävä indikaattori kirjoitusoperaation onnistumisesta, koska tällöin data oli jo tiedostojärjestelmästä ulos saatavilla, vaikka ei vielä minnekään fyysiselle levylle olisikaan kokonaisuudessa talletettuna.

5.2 Samanaikaiset kirjoitustestit

Horisontaalisen skaalautuvuuden testausta varten tiedostopalvelimeen / tiedostoklusteriin muodostettiin yhteys ulkopuolelta kolmesta eri solmukoneesta. Solmukoneisiin julkaistiin Kubernetes-podi, jossa suoritettiin opinnäytetyön kohdan 5.1 kaltainen dd-komento sillä erolla, että jokainen podi kirjoitti erilliseen tiedostoon: Saman tiedoston samanaikainen kirjoittaminen dd:llä aiheutti ongelmia sekä itse kirjoitusoperaatioon että levyjärjestelmän suorituskykyyn.

Testiympäristönä käytettiin GKE:ssa pyörivää n1-standard-2 kokoluokan klusteria, tässä tapauksessa kolmella koneella varustettuna. Klusteriin julkaistiin kolme podia, jotka levitettiin klusterin eri solmukoneisiin. Podit laitettiin suorittamaan kirjoitusoperaationsa samanaikaisesti, 10 kertaa toistuvasti, ja kaikista kirjoitusoperaatioista otettiin keskiarvo.

Kirjoitusoperaatioiden tulokset ovat listattuina alapuolella olevassa taulukossa.

Taulukko 2. Samanaikaisten kirjoitusoperaatioiden suorituskyky eri levyjärjestelmillä Linux:n dd-komentoa käyttäen.

Levyjärjestelmä	3x samanaikainen uusi tiedosto 10x	3x samanaikainen ylikirjoitus 10x
CEPH	46.5 MB/s	456.4 MB/s
NFS	43.0 MB/s	43.3 MB/s
GlusterFS	74.8 MB/s	72.5 MB/s

Horisontaalisesti skaalatun kuorman tulokset vastasivat yksittäisten kirjoitusoperaatioiden tuloksia. NFS ja Ceph vastasivat läheisesti toisiaan nopeuden puolesta, mikäli kyseessä oli täysin uusi tiedosto. Mikäli Ceph pystyi hyödyntämään olemassaolevaa dataa operaation aikana, kirjoitusoperaatiot nopeutuivat huomattavasti. GlusterFS taasen tarjosi vertailukumppaneitaan paremman suorituskyvyn uusille tiedostoille, joskin jäi selkeästi Ceph:n jälkeen

välimuistin hyödyntämisessä. GlusterFS:n horisontaalinen skaalautuvuus vaikutti myös hiukan muita tiedostojärjestelmiä tehokkaammalta.

5.3 Havaintoja nopeustesteistä

Levyjärjestelmien suorituskykytestaus jäi todella pintaa raapaisevaksi ja testaukseen jäi useita puutteita. Googlen verkkoyhteyksien nopeus on huomattavana tekijänä kaikkien kolmen levyjärjestelmän osalla ja varsinkin Ceph ja GlusterFS hyötyivät paljon nopeista verkkoyhteyksistä. Useiden eri konetyyppien vertailu tiedostojärjestelmien nopeuksiin olisi myös hyödyllistä.

Testeihin jäi myös mahdollista virheellisyyttä johtuen Linux:n levy-cachetusta (Webnersolutions 2019): Toistuvat testit samannimiseen tiedostoon saattoivat vääristyä itse levyjärjestelmästä riippumattomista syistä ja testejä ajettaessa olikin havaittavissa jonkinlaisia eroja riippuen siitä, monesko testin suorituskerta oli kyseessä. Mikäli peräkkäisiä kirjoitusnopeustestejä ajettiin eri nimisiin tiedostoihin kaikkien levyjärjestelmien osalta havaittiin testien edistyessä huomattava pudotus suoritusnopeudessa.

Linux:n levy-cachetusta ei suoraan saa poistettua käytöstä mitenkään, mutta testien paikkansapitävyyttä olisi voinut parantaa tyhjentämällä tämän cachen testien välissä.

Testausta olisi myös voinut laajentaa lisäämällä tähän lukunopeustestit. Nopeat lukunopeustestit suoritettiinkin, mutta aiemmin mainitusta Linux:n levy-cachetuksesta johtuen ensimmäisen lukukerran jälkeen testin tulokset tulivat lähes puhtaasti cachesta, joka vääristi tulosta huomattavasti ja lopulta nopeuden ainoaksi rajoittavaksi tekijäksi jäi verkkoyhteyden nopeus.

Testausta varten olisi myös voitu kokeilla vaihtoehtoisia metodeja Linux:n dd-komennon lisäksi, esimerkiksi GlusterFS:n dokumentaatiossa mainittuja iozone ja smallfile työkaluja (GlusterFS 2019).

Kaiken kaikkiaan levyjärjestelmien testaus olisi voinut huomattavasti kattavampaa ja aihealueeseen tulisi perehtyä huomattavasti tarkemmin näiden testien suoritusta varten, kuin mitä tässä opinnäytetyössä tehtiin.

6 Yhteenveto

Työssä tarkasteltiin kolmea eri tiedostojärjestelmää joilla tarjota luku- sekä kirjoitusoikeudet yhteiseen dataan. Opinnäytetyö perehtyi suurelta osin tiedostojärjestelmien käyttöönottoon Google Cloud Platform:ssa ja pääasiallisena tarkoituksena oli kartoittaa levyjärjestelmien toimivuus kyseisellä alustalla.

Järjestelmistä yksinkertaisin oli NFS. Kyseinen tekniikka ei ole samalla tavalla skaalautuva kuin kaksi muuta vertailun kohteena olevaa tiedostojärjestelmää, koska kysymys on ainoastaan yhdestä palvelimesta, jolta lähiverkon kautta jaetaan ulkopuolelle pääsy tiedostoihin. Tekniikan käyttöönotto ja itse levytilan helppo lisääminen ovat järjestelmän parhaat puolet. Prosessointitehon lisääminen tauotta ei ole mahdollista ja mikäli järjestelmä kasvaisi huomattavaan kokoluokkaan yhden palvelimen ratkaisu suuren datamäärän käsittelyyn olisi riskialtista.

Ceph ja GlusterFS vaativat valmiiden ohjeiden lisäksi huomattavan määrän osaamista. Vaikka tekniikoiden käyttöönottoon hyödynnettiin suoraan Kubernetesille kehitettyjä sovelluskehyksiä, molemmat tekniikat vaativat silti huomattavan tietämyksen itse Ceph:n ja GlusterFS:n toimivuudesta. Kummallekaan järjestelmälle ei vielä löydy täydellistä implementaatiota

Kubernetes-klusteriin vaan toimivuuden takaamiseksi käyttäjältä tulee löytyä osaamista järjestelmien vikatilanteiden korjaamisesta käsipelillä.

Suoritustehoiltaan sekä GlusterFS että Ceph pärjäsivät NFS:ää paremmin ja levytilan sekä prosessointitehon lisääminen tauoitta oli mahdollista, vaikkakin selkeästi vaivalloisempaa kuin NFS:n tapauksessa järjestelmien monimutkaisuudesta johtuen.

Opinnäytetyössä GlusterFS:än julkaisuun käytetty gluster-kubernetes -repositorio on mahdollisesti hyljätty, joten tätä en voi varauksetta suositella. Rook:n avulla julkaistuun Ceph-klusteriin on viime kuukausina tullut mielenkiintoisia muutoksia joita tässä opinnäytetyössä ei suoraan tarkasteltu, kuten PVC-pohjainen levytilan varaus suoraan GCP:stä. Tämä tekniikka mahdollistaa huomattavasti helpomman implementaation Ceph-klusterin pystyttämiseksi suoraan Google Kubernetes Engineen.

Jaettavien levyjärjestelmien osalta kehitys on vieläkin sekä todella nopeaa että keskeneräistä. Tulevina vuosina tällä saralla on luultavasti nähtävissä useita muutoksia ja levyjärjestelmät jotka saadaan tuotantoympäristöön helposti julkaistua skaalautuvaksi kokonaisuudeksi tulevat varmasti kasvattamaan suosiotaan.

Lähteet

Ceph. 2019a. Adding/Removing Monitors. https://docs.ceph.com/docs/master/rados/operations/add-or-rm-mons/. 20.09.2019.

Ceph. 2019b. Intro to Ceph. http://docs.ceph.com/docs/master/start/intro/. 10.5.2019.

Docker. 2019. What is a Container. https://www.docker.com/resources/what-container. 10.5.2019.

GeeksForGeeks. 2019. ‘dd’ command in Linux. https://www.geeksforgeeks.org/dd-command-linux/. 16.09.2019.

Github. 2019. Gluster-kubernetes. https://github.com/gluster/gluster-kubernetes. 20.09.2019.

GlusterFS. 2019. Gluster performance testing. https://docs.gluster.org/en/latest/Administrator%20Guide/Performance%20Testing/. 20.09.2019.

Google. 2019. GKE Overview. https://cloud.google.com/kubernetes-engine/docs/concepts/kubernetes-engine-overview. 8.5.2019.

Kubernetes. 2019. GlusterFS Readme.md. https://github.com/kubernetes/examples/tree/master/volumes/glusterfs. 11.09.2019.

Rook. 2019a. Ceph Storage Quickstart. https://rook.io/docs/rook/v1.0/ceph-quickstart.html. 10.5.2019.

Rook. 2019b. Rook StorageClassDeviceSets. https://github.com/rook/rook/blob/master/design/storage-class-device-set.md. 18.09.2019.

Webnersolutions. 2019. Understanding Disk Cache in Linux. https://blog.webnersolutions.com/understanding-disk-cache-in-linux. 20.09.2019.

Wikipedia. 2019. Computer cluster. https://en.wikipedia.org/wiki/Computer_cluster. 10.5.2019.

```
#! /bin/bash

###
# DOCKER
###

# Install Docker from Ubuntu's repositories:
apt-get update
apt-get install -y docker.io

# or install Docker CE 18.06 from Docker's repositories for Ubuntu or Debian:

## Install prerequisites.
apt-get update && apt-get install apt-transport-https ca-certificates curl software-properties-common -y

## Download GPG key.
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | apt-key add -

## Add docker apt repository.
add-apt-repository \
  "deb [arch=amd64] https://download.docker.com/linux/ubuntu \
  $(lsb_release -cs) \
  stable"

## Install docker.
apt-get update && apt-get install docker-ce=18.06.0~ce~3-0~ubuntu -y

# Setup daemon.
cat > /etc/docker/daemon.json <<EOF
{
 "exec-opts": ["native.cgroupdriver=systemd"],
 "log-driver": "json-file",
 "log-opts": {
   "max-size": "100m"
 },
 "storage-driver": "overlay2"
}
EOF

mkdir -p /etc/systemd/system/docker.service.d
```

```
# Restart docker.
systemctl daemon-reload
systemctl restart docker

###
# KUBERNETES
###

apt-get update && apt-get install -y apt-transport-https curl
curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | apt-key add -
cat <<EOF >/etc/apt/sources.list.d/kubernetes.list
deb https://apt.kubernetes.io/ kubernetes-xenial main
EOF
apt-get update
apt-get install -y kubelet kubeadm kubectl
apt-mark hold kubelet kubeadm kubectl
```

```
# Init cluster
sudo kubeadm init --pod-network-cidr=10.244.0.0/16

#set up environment
mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config

# install canal network
kubectl apply -f https://docs.projectcalico.org/v3.3/getting-started/kubernetes/installation/hosted/canal/rbac.yaml
kubectl apply -f https://docs.projectcalico.org/v3.3/getting-started/kubernetes/installation/hosted/canal/canal.yaml
```