LATVIJAS UNIVERSITĀTE
DATORIKAS FAKULTĀTE

OS KONTEINERU PIELIETOJAMĪBA MŪSDIENU INFRASTRUKTŪRAS MĀKOŅOS

BAKALAURA DARBS

Autors: Liene Rieksta
Studenta apliecības Nr.:lr11118
Darba vadītājs: lektora p.i. Dr.sc.comp. Leo Trukšāns

RĪGA 2015

ANOTĀCIJA

Darbā ir veikts pētījums par Linux konteineru, īpašu uzsvaru liekot uz Docker, pielietojamību kā virtualizācijas platformu infrastruktūras servisu mākonī. Darbs izstrādāts ar mērķi izprast uz Linux konteineriem balstītu mākoņu realizācijas tehnoloģijas un to attīstības tendences un izmantot iegūtās zināšanas reāla datu centra infrastruktūras un servisu pakalpojumu nodrošināšanai. Pētījuma gaitā tika uzstādīts Docker virtualizācijas dzinis uz OpenStack platformas, kā arī tika uzstādīti un izpētīti vairāki citi projekti, kuri ir cieši saistīti ar Docker konteineru pārvaldības rīku un ir potenciāli izmantojami kā mākoņskaitļošanas platformu komponentes.

Atslēgvārdi: Virtualizācija, konteiners, mākoņskaitļošana, Docker

ABSTRACT

Usage of OS containers in modern infrastructure clouds

This document consists of a research about Linux container – especially Docker, usability as a virtualization platform in an infrastructure cloud. The main purpose of the research is to understand technologies that underlay Linux container powered clouds and overall cloud computing technology evolution trends and use the gained knowledge to provide an infrastructure service in an existing data center. During the research, Docker virtualization driver was installed on an OpenStack platform. Also several other products, that are tightly connected with Docker container management tool and are usable as cloud computing platform components, were installed and tested.

Keywords: Virtualization, container, cloud computing, Docker

APZĪMĒJUMI

- OS – Operētājsistēma. Programmatūra, kas atbild par datora aparatūras resursu pārvaldību un nodrošina lieotāja lietotņu darbību.
- IP adrese - Skaitliska adrese, kas viennozīmīgi identificē katru datoru tīklā Internet un kas izveidota kā četru ar punktiem atdalītu skaitļu virkne, piem.: 192.100.81.101. [1]
- VM – virtuālā mašīna.Virtualizācijas tehnoloģijas produkts. Uz virtuālās mašīnas iespējams uzstādīt pilnu operētājsistēmu un darbināt lietojumprogrammas tāpat kā uz fiziska datora.
- Chroot – mehānisms, kur no procesa skatpunkta tiek nomainīta saknes direktorija tādā veidā izolējot šo procesu no pārējā direktoriju koka.
- Ezjail – rīks FreeBSD Jail operētājsistēmas līmeņa virtualizācijas procesa pārvaldībai.
- pkgng – FreeBSD pakotņu pārvaldības rīks.
- Hipervizors – virtualizācijas platforma, programmatūra, kas nodrošina virtuālo mašīnu pārvaldību.
- Docker – uz Linux konteineriem balstīta OS līmeņa virtualizācijas programmatūra.
- OpenStack – populāra atvērtā pirmkoda mākoņskaitļošanas platforma.
- CoreOS – Uz Linux kodola balstīta kompakta klasteru operētājsistēma ar iebūvētiem klastera resursu pārvaldības rīkiem un kuras lietotņu pārvaldības mehānisms balstās uz Docker.
- Kubernetes – Docker konteineru klasteru pārvaldības rīks, kas izmanto etcd servisu klastera mezglu pārvadībai.
- Borg – Google izstrādāta un lietota klasteru pārvaldības sistēma.
- etcd – klasteru vadības serviss, kurš koordinē klastera veiksmīgai darbībai nepieciešamos procesus - nodrošina mezglu atpazīšanu, Master mezgla ievēlēšanas mehānismu, etc.
- IaaS – Infrastructure as a service. Mākoņskaitļošanas pakalpojums, kurā klientam tiek piedāvāta virtuālas infrastruktūras īre – virtuālās mašīnas.
- PaaS – Plaform as a service. Mākoņskaitļošanas pakalpojums, kurā klientam tiek piedāvāta platforma klienta lietotnes izstrādei un darbināšanai.

- SaaS – Software as a Service. Mākoņskaitļošanas pakalpojums, kur klientam tiek piedāvāts izmantot konkrētu lietotni.
- Kontrolgrupa (cgroup) – Linux kodola funkcija, kas uzrauga un ierobežo procesiem vai procesu grupām datora resursu pieejamību.
- Vārdtelpa (namespace) – Linux kodola mehānisms, kas izolē procesus vai procesu grupas no pārējās sistēmas.

SATURA RĀDĪTĀJS

IEVADS

Šajā darbā tiek pētīti operētājsistēmas līmeņa virtualizācijas risinājumi ar mērķi optimizēt eksistējošu datu centra infrastruktūru. Darbā tiek aplūkoti dažādi mākoņu, virtualizācijas un konteineru risinājumi un produkti, apzinātas datu centra infrastruktūras problēmas un trūkumi, kā arī meklēti veidi, kā tos risināt ņemot vērā patreizējās IT pasaules attīstības tendences un pieejamos risinājumus. Darba laikā tika uzinstalēti un testēti vairāki produkti, kas darbojas kā komponentes mākoņplatformu būvēšanai un tiek izvērtēta to pielietojamība infrastruktūras servisa sniegšanā un patreizējās datu centra arhitektūras optimizēšanā.

Datu centrs savai organizācijai nodrošina plašu servisu klāstu, tai skaitā izvieto mājaslapas, datu bāzes, kā arī specifiskas lietotnes organizācijas iekšējo procesu darbības nodrošināšanai. Tāpat datu centrs arī piedāvā infrastruktūru (virtuālās mašīnas) dažādu servisu izvietošanai. Izvēloties pareizus infrastruktūras pārvaldības risinājumus ir iespējams uzlabot pakalpojuma efektivitāti un minimizēt resursu virstēiņu, tādā veidā samazinot uzturēšanas izmaksas.

Darba izstrādes gaitā tiek apzināti potenciālie risinājumi, kuri būtu pielietojami infrastruktūras optimizēšanai. Ņemot vērā tehnoloģijas strauji augošo popularitāti pētījuma uzsvars tiek likts uz Linux konteineru pielietojamību infrastruktūras, kā servisa pakalpojuma sniegšanā.

Informācija par potenciālajiem risinājumiem tiek gūta no Interneta un praktiskiem eksperimentiem. Tiek lasīta produktu dokumentācija, kā ari produktu lietotāju atsauksmes par tā stabilitāti, kļūdām, risinājumiem un aprisinājumiem. Visi zvēlētie produkti, izņemot OpenStack, tika uzstādtīti un pārbaudīti uz tādām pašām iekārtām, uz kādām darbojas produkcijas vide. Tika novērtēti potenciālie ieguvumi un zaudējumi, kas varētu rasties ieviešot izvērtējamo produktu datu centra infrastruktūrā.

Darbā vispirms tiek aprakstīta esošā infrastruktūra. Tad tā tiek analizēta un tiek uzskaitīti esošā risinājuma trūkumi un priekšrocības. Tiek apzināts, kādu funkcionalitāti ir vēlams saglabāt, no kā būtu jāatbrīvojas un kādas jaunas funkcijas būtu nepieciešams ieviest, lai uzlabotu pakalpojuma kvalitāti. Pēc tam tiek aplūkoti mākoņskaitļošanas un virtualizācijas risinājumi, ar kuriem iespējams sasniegt uzstādītos mērķus. Tiek aprakstīts katra produkta

izvērtēšanai izveidotais infrastruktūras modelis un tā izveides process, kā arī tiek veika šī modeļa novērtēšana ņemot vērā tā pielietojamību datu centra darbības uzlabošanai. Tiek apzināti ieguvumi gan problēmas, kas var rasties ieviešot jauno risinājumu. Visbeidzot tiek nodefinēts attīstības virziens, ņemot vērā organizācijas uzņēmējdarbības mērķus, kā arī straujo attīstību mākoņdatņošanas un virtualizācijas jomā.

1 ESOŠĀS INFRASTRUKTŪRAS APRAKSTS

1.1 Pakalpojumi un klienti

Datu centra klientu loku sastāda organizācijas struktūrvienības un darbinieki, kuri izmanto datu centra resursus, lai nodrošinātu pakalpojumu sniegšanu tālāk organizācijas klientiem. Neliela daļu no datu centra klientiem strādā ārpus organizācijas lokālā tīkla. Rezultātā pakalpojuma pieejamība jānodrošina gan no organizācijas tīkla gan arī no Interneta.

Piedāvāto pakalpojumu sarakstā ir gan atsevišķi servisi, piemēram mājaslapu mitināšana, datņu izvietošana, datu bāzes, etc., gan arī infrastruktūra – virtuāls serveris klienta lietotnes izvietošanai.

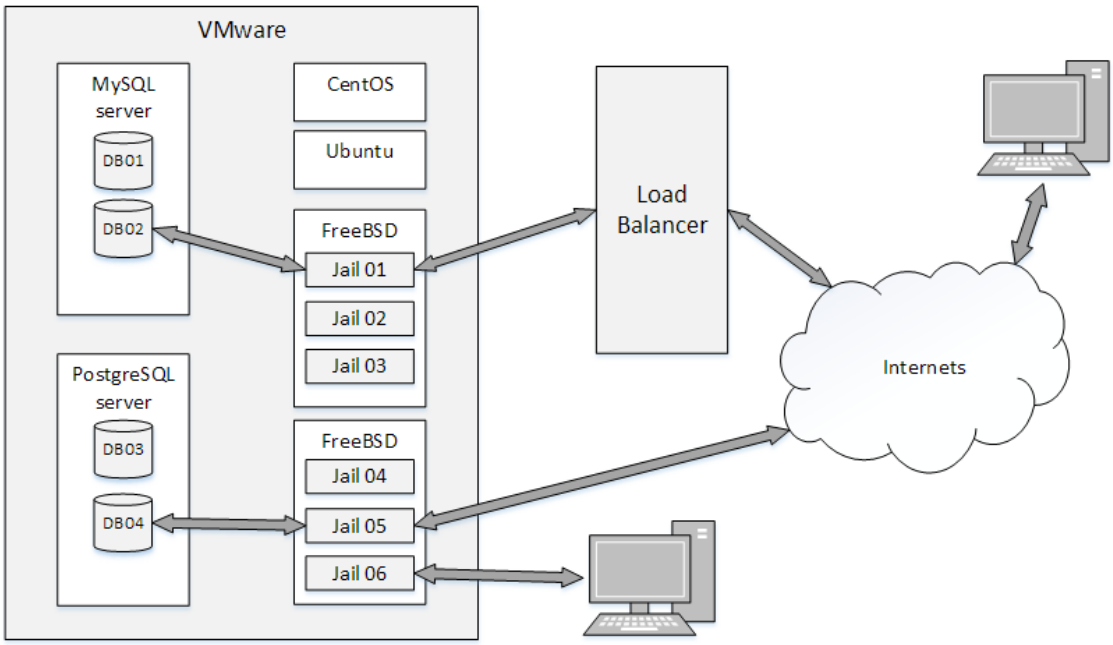
Organizācijas IT infrastruktūrā ir nodefinēta robeža starp klienta un datu centra atbildību. Datu centrs ir atbildīgs par fiziskās un virtuālās aparatūras, tīkla infrastruktūras un operētājsistēmas uzturēšanu. Kā arī datu centrs ir atbildīgs par tās programmatūras uzturēšanu, kura tiek piedāvāta klientam kā serviss – datu glabāšana, tīmekļa mitināšana, epasts, etc.

Klients ir atbildīgs par savas specifiskās programmatūras izstrādi un uzstādīšanu. Datu centra administrators sagatavo programmatūrai nepieciešamo vidi, saskaņā ar klienta iesniegto specifikāciju.

1.2 Arhitektūra un programmatūra

Patreizējā infrastruktūra ir izvietota uz daudziem virtuāliem serveriem, kuri darbojas VMware vidē. Vienkāršota infrastruktūras shēma ir redzama attēlā *Attēls **1.1***. Attēla uzskatāmības saglabāšanai tajā nav attēlota tā tīkla infrastruktūra un iekārtas, kuras tieši neietekmē pakalpojuma aritektūru un kurās netiek plānotas izmaiņas, tai skaitā ugunsmūri, maršrutētāji, etc.

Kā redzams attēlā ***Attēls 1.1***, pakalpojuma nodrošināšanai tiek izmantotas VMware vidē mitinātas virtuālās mašīnas. Datu bāzes ir izvietotas uz atsevišķām virtuālajām mašīnām. Lielākā daļa pārējo servisu tiek izmitināti uz FreeBSD serveriem pielietojot FreeBSD Jail, kā konteineru risinājumu. Konteineri tiek pārvaldīti izmantojot Ezjail [2] rīku. Paralēli FreeBSD, eksistē virtuālās mašīnas, kuras tiek darbinātas izmantojot Ubuntu un CentOS operētājsistēmas.



Attēls 1.1 ***Datu centra infrastruktūras pārskats.***

Infrastruktūrai attīstoties ir izveidojusies situācija, ka attālinātiem klientiem piekļuve pie servisiem ir nodrošināta gan tieši gan arī caur slodzes stabilizatoru. Klienti no lokālā tīkla pieslēdzas pie konteineriem tieši. Katram konteineram ir viena vai vairākas IP adreses, atkarībā no servisa, kas tiek darbināts, kā arī no attālinātās piekļuves veida. Nepastarpināta piekļuve datu bāzēm no klientu darbstacijām netiek nodrošināta. Datu bāzes ir pieejamas tikai un vienīgi no konteineriem vai datu centrā izvietotiem serveriem.

Laika gaitā organizācijas datu centrā ir nostiprinājusies pārliecība par konteineru noderīgumu attālinātu servisu pakalpojuma nodrošināšanā. Konteineri sniedz papildus drošības līmeni, izolējot procesus un lietotājus gan no saimnieksistēmas gan vienu no otra. Tie ļauj uz vienas sistēmas darbināt vienas programmatūras dažādas versijas, kā arī dažādas lietotnes ar atšķirīgām prasībām pret vidi. Izvēle darbināt programmatūru uz FreeBSD tika izdarīta 2008.gadā, pirms bija pienācīgi attīstījusies Linux konteineru tehnoloģija. Pirms

FreeBSD tika veikti eksperimenti ar procesu izolāciju uz Linux, izmantojot kontrolgrupas un chroot, taču šie līdzekļi nespēja nodrošināt nepieciešamo izolācijas līmeni. Ņemot vērā, ka pašu izvirzīts bija uzstādījums, lai drošs operētājsistēmas līmeņa virtualizācijas risinājums būtu iekļauts noklusētajā operētājsistēmas kodolā, tad tajā brīdī no drošības un lietojamības viedokļa FreeBSD šķita vispiemērotākais risinājums uzstādīto mērķu sasniegšanai.

2 PROBLĒMU ANALĪZE

2.1 Infrastruktūras izīrēšana

Kā jau minēts ***1.2*** nodaļā, aprakstītā arhitektūra tika ieviesta 2008.gadā un tajā pašā laikā tika ari izstrādāts un aprakstīts jauna servisa izvietošanas process. Jau tad datu centra administratoriem bija ideja par to, ka virtuālo mašīnu vietā klientiem varētu piedāvāt konteineru risinājumu savas programmatūras izvietošanai.

Praksē izrādījās, ka bieži vien klienti vēlas tieši Linux operētājsistēmu, kā arī savas lietotnes ir izstrādājuši pieņemot, ka tās tiks izvietotas uz kāda no Linux distributīviem vai arī iebilst pret FreeBSD, argumentējot, ka nav pieredzes darbā ar šo operētājsistēmu.

Pēc vairākiem gadījumiem, kad lietotājs kaut kāda iemesla dēļ nespēj uz FreeBSD korekti uzstādīt savu programmatūru, tika pieņemts lēmums, ka šādos gadījumos tiks turpināta prakse piešķirt klientam atsevišķu virtuālo mašīnu. Rezultātā ir izveidojusies situācija, kurā konteineri tiek izmantoti tiešsaistes servisu un tīmekļa mitināšanas pakalpojuma nodrošināšanai, savukārt infrastruktūras piegādei tiek izmantotas virtuālās mašīnas. Šāds sadalījums uzliek pienākumu individuāli noskaidrot un padziļināti izvērtēt katru klienta pieprasījumu, lai būtu iespējams pieņemt lēmumu, kādā veidā tiks nodrošināts viņam nepieciešamais pakalpojums. Tas ir atbildīgs pienākums, kurš bieži vien ietver komunikāciju ne tikai ar klientu, arī ar klienta programmatūras izstrādātāju, līdz ar to, tas ir resursu prasīgs process no administratora darbalaika viedokļa, jo nepareizs lēmums var novest pie lieka resursu patēriņa vai novēlota pakalpojuma piegādes laika.

Dotajā brīdī pakalpojumu sniegšanai tiek uzturētas divas platformas – Linux un FreeBSD. Dotajā brīdī tiek nopietni apsvērta doma ieviest Linux konteineru risinājumu arī infrastruktūras pakalpojuma nodrošināšanai, tādējādi samazināt aparatūras resursu virstēriņu, kas rodas no pilnās virtualizācijas izmantošanas. Ir vēlme arī atbrīvoties no FreeBSD un samazināt administrējamo operētājsistēmu skaitu datu centrā, tai pašā laikā nezaudējot priekšrocības, kuras sniedz konteinerizēta servisu pārvaldība. Šādu darbību rezultātā tiktu vienkāršota datu centra infrastruktūra un atslogoti administratori.

2.2 Ar FreeBSD saistītās problēmas

Lai arī FreeBSD Jail jau ir nobriedis konteineru risinājums, kurš nodrošina visas teorijā uzskaitītās konteineru priekšrocības pār virtuālajām mašīnām, eksistē laika gaitā konstatētas administrēšanas problēmas, kuras motivē meklēt iespējas migrēt iekš FreeBSD Jail esošās lietotnes un datus uz Linux konteineriem un ieviest kādu no Linux konteineru pārvaldības rīkiem.

Katrs jauns Jail tiek veidots atsaucoties uz klienta pieprasījuma formu, kurā ir precīzi aprakstīta nepieciešamā servisa konfigurācija. Eksistē gadījumi, kad klienta programmatūras uzstādīšanai ir nepieciešama kāda noteikta Linux distribūcija. Tādos gadījumos klientam tiek veidota atsevišķa virtuālā mašīna. Jāpiebilst, ka gadījumu, kad klients būtu specifiski pieprasījis FreeBSD praksē vēl nav bijis.

Katram klientam tiek izveidots tukšs konteiners no veidnes, kurā ir iekļauti faili un parametri, kuri visiem Jail būs identiski vai arī kam nepieciešamas minimālas izmaiņas, piemēram laika zona, resolv.conf, rc.conf faili. Tālākā programmatūras uzstādīšana tiek veikta izmantojot portu koku. Tā ir laika un resursu prasīga metode, kur secīgi tiek kompilēta katra programmatūras komponente. Atkarībā no servera resursiem, uzstādāmās programmatūras un tā, vai programmas pirmkods ir jālejupielādē no Interneta. Šī procedūra var aizņemt no dažām minūtēm līdz vairākām stundām, kā arī mākslīga Jail resursu ierobežošana šo procesu var paildzināt. Tajā laikā tiek radīta ievērojama papildus noslodze serverim.

Problēmas mēdz radīt situācijas, kad nepieciešams veikt izmaiņas lietotnes konfigurācijā, kas prasa papildus programmatūras uzstādīšanu vai atjaunošanu, pēc tam, kad ir ticis atjaunināts portu koks. Tas nozīmē, ka ikreiz pēc portu koka aktualizēšanas šīs problēmas potenciāli ietekmēto konteineru skaits palielinās.

Vistipiskākā no šādām situācijām ir saistīta ar php aktuālo versiju maiņu. Portu kokā aktuālā php versija atrodas direktorijā php5, savukārt vecākas un jaunākas versijas tiek apzīmētas pievienojot nākamo versijas ciparu – php52, php53, php55, etc. Atjauninot portu koku ir iespējams nonākt pie situācijas, kad aktuālā programmatūras versija mainās. Tādā situācijā, ja līdz šim direktorijā php5 atradās php5.3, tad pēc atjaunināšanas php5 direktorijā atrodas php5.4 un php5.3 tiek pārvietots uz php53 direktoriju. Šīs te izmaiņas netiek korekti tālāk izplatītas uz katra jail individuālo uzinstalētās programmatūras reģistru un tas rada

problēmas ar jaunu paplašinājumu pieinstalēšanu, php atjaunināšanu, kā arī problēmas ar paku savstarpējām atkarībām.

FreeBSD protams eksistē rīki, kas ir paredzēti pakotņu reģistra salabošanai taču fakts, ka šie rīki ir regulāri jāizmanto jau vien nozīmē, ka ar esošo sistēmu kaut kas nav kārtībā. Izmaiņu ieviešanas laiks palielinās, kā arī ir gadījumi, kad to palaišana problēmas neatrisina pilnībā. Rezultātā ir iespējams nonākt pie situācijas, kur nepieciešama pilnīga ietekmētās programmatūras un tās atkarīgo komponenšu noņemšana un uzstādīšana no jauna, kas, kā jau minēts iepriekš, ir dārgs process gan no administratora darba stundu gan arī no patērēto servera resursu skatpunkta. Kopš pkgng [3] paku pārvaldības rīka parādīšanās, programmatūras paku uzstādīšanas process ir vienkāršots, bet šī rīka agrīnajās versijās eksistē savas nepilnības, kas rada problēmas.

Ezjail būvē konteineru pamatattēlu (*basejail*) no operētājsistēmas, kur tas ir uzstādīts. Rezultātā veidojas atkarība starp konteineru un saimniekserveri, kura ietekmē iespējas atjaunināt saimnieksistēmu un konteinerus, kā arī ierobežo iespējas migrēt konteinerus starp vairākiem serveriem, ja tie nav vienādi. Piemēram līdzšinējā pieredze rāda, ka vienkārši pārkopēts Jail no FreeBSD 7.x uz 8.x versiju darbojas bez problēmām, savukārt no 7.x vai 8.x uz 9.1 tas, pēc identiskas ezjail uzstādīšanas procedūras, nedarbojas ir jāveic papildus traucējummeklešanas procedūra, lai tos iedarbinātu.

Šī ir aktuāla problēma situācijās, kad ir vēlme likvidēt serveri ar novecojušu operētājsistēmu, bet dažādu iemeslu dēļ nav iespējas pārcelt konteinerā esošo programmatūru uz citu. Kā arī šādā situācijā nav pārbaudīts un ir grūti paredzēt, kas notiks ar konteineriem, pēc operētājsistēmas versijas maiņas un attiecīgi Jail pamatattēla atjaunināšanas.

2012.gadā FreeBSD kļuva pieejams ilgi gaidīts jauns paku pārvaldības rīks – pkgng [3], kurš funkcionalitātes un lietojamības ziņā ir pietuvināts populāriem Linux distribūciju paku pārvaldības rīkiem. Taču praksē pāreja uz šo rīku nav bijusi tik gluda, kā gribētos. Konvertējot no portiem instalēto vai ar veco pkg_* rīku uzstādītāto paku reģistru uz pkgng ne vienmēr saglabātas vai izveidotas pareizas pakotņu atkarību saites. Rezultātā situācijā, kad daļa no programmatūras ir uzstādīta izmantojot portu koku, ar pkgng pievienotā programmatūra daļā gadījumu nedarbojas korekti. Kā arī gadās, ka pakotnēm no oficiālajiem repozitorijiem nav korekti nodefinētas atkarības, to uzstādīšana beidzas ar kļūdu paziņojumu un nākas manuāli meklēt un uzstādīt trūkstošo programmatūru.

Pēdējā problēma, kura ir aktuāla ir rezerves kopēšana. Datu centra rezerves kopēšanas risinājums ir integrēts ar VMware virtualizācijas platformu. Šāds risinājums ļauj neinstalēt uz

virtuālās mašīnas rezerves kopēšanas aģentu, bet veikt atsevišķu failu vai direktoriju kopēšanu VMware līdzekļiem, pa tiešo no virtuālās mašīnas failsistēmas. Diemžēl FreeBSD nav šīs funkcionalitātes atbalstāmo operētājsistēmu sarakstā. Tas sarežģī un padara lēnāku rezerves kopēšanas procesu vai ari rada papildus diska vietas patēriņu, ja rezerves kopēšana tiek veikta visai virtuālajai mašīnai. Otrā scenārija gadījumā tiek arī sarežģīts datu atkopšanas process, jo nākas restaurēt visu virtuālo mašīnu, lai no tās atgūtu kaut vai tikai viena faila rezerves kopiju.

2.3 Risināmo problēmu kopsavilkums

- Ilgs servisa piegādes laiks.
- resursu prasīgs servisa piegādes process,
- nehomogēna vide,
- potenciāli novēršams aparatūras resursu virstēriņš,
- Problēmas, saistītas tieši ar FreeBSD Jails patreizejo implementāciju:
    - apgrūtināta izmaiņu veikšana konteineros ar vecāku programmatūru,
    - konteinera atkarība no operētājsistēmas versijas (Ierobežotas migrācijas iespējas, ierobežotas iespējas atjaunināt saimnieksistēmu neietekmējot konteineru darbību).
- FreeBSD nav iespējama efektīvākā iespējamā rezerves kopēšanas procedūra.

3 MĀKOŅSKAITĻOŠANA

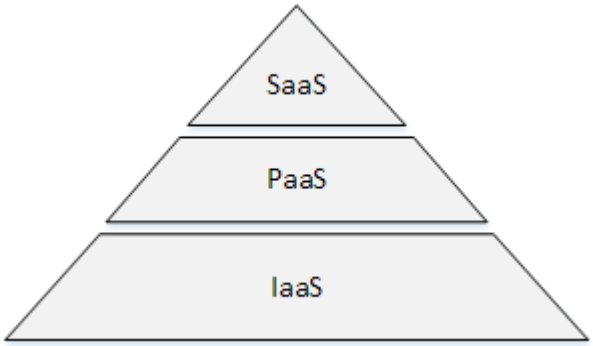
„Mākoņskaitļošana" dotajā brīdī ir viens no IT nozares populārākajiem terminiem. Šī apzīmējuma izplatītā lietojuma vietā un nevietā dēļ, brīžiem ir sarežģīti uztvert, ko precīzi šis termins apraksta un kāda infrastruktūra slēpjas zem vārda „mākonis" konkrētajā situācijā, taču viena kopīgā iezīme saglabājas – mākoņa mērķis ir nodrošināt patērētājam pieeju attālinātai videi, kur lietotnes darbojas elastīgā un mērogojamā infrastruktūrā, nevis uz lietotāja paša darbstacijas.

Mākoņskaitļošanas tehnoloģijas nodrošina lietotājiem iespēju izmantot dažādus servisus tiešsaistē, tādā veidā izvairoties no nepieciešamības uzturēt savu infrastruktūru šo servisu nodrošināšanai. Vairumā gadījumu mazos un vidējos uzņēmumos izmantot mākoņskaitļošanas pakalpojumus ir krietni lētāks un ērtāks risinājums, nekā pirkt savus serverus, programmatūru un algot kvalificētu personālu šīs infrastruktūras uzturēšanai. Mākoņskaitļošana ļauj klientam mērogot savu saimniecību atbilstoši augošām vai dilstošām prasībām, tādā veidā izvairoties no finansiāliem zaudējumiem, kuri var rasties nepareizas uzņēmējdarbības izaugsmes prognozes rezultātā

Lai arī termins „mākoņskaitļošana" ir radies salīdzinoši nesen, idejas un tehnoloģijas, uz kā šis termins balstās, vairs nav nekāds jaunums. Internets ir sāvis savu neatlaidīgo attīstību jau tālajā 1969.gadā, kad tika izveidots tā priekštecis ARPANET [4]. Tiek uzskatīts, ka pirmie virtualizācijas mēģinājumi notikuši jau 1960tajos [5] gados, kad IBM sāka darbu pie resursu un laika dalīšanas mehānismiem lieldatoros. Pirmais komerciālais klasteru produkts tika izveidots 1977.gadā [6]. Tas pierāda, ka vienmēr ir eksistējusi vēlme pēc ērti lietojamas, elastīgas un drošas IT vides, kurā iespējams maksimāli efektīvi izmantot pieejamos fiziskos resursus.

3.1 Mākoņu veidi

Eksistē vairāki veidi, kā iedalīt un klasificēt mākoņus. Tālāk ir aplūkoti divi populārākie no tiem.



Attēls 3.1 ***Mākoņpakalpojumu klasifikācijas grafisks attēlojums.***

Viens no tipiskiem veidiem, kā klasificēt mākoņskaitļošanas pakalpojumus ir pēc pakalpojuma veida *Attēls* ***3.1***.

- Infrastruktūra kā serviss (IaaS – Infrasructure as a service). Pakalpojuma sniedzējs piedāvā klientam infrastruktūru klienta programmatūras darbināšanai. Ar to var saprast, ka piedāvāts tiek virtuāli vai fiziski serveri, datu glabāšanas iekārtas, kā arī tīkla infrastruktūra, kas var būt IP adreses vai pat adresu apgabali, tīkla iekārtas, tādas kā ugunsmūri, slodzes stabilizatori, etc. Šajā modelī lietotājs pats ir atbildīgs par programmatūras uzstādīšanu un uzturēšanu.
- Platforma kā serviss (PaaS – Platform as a service): Šajā gadījumā pakalpojuma sniedzējs nodrošina skaitļošanas platformu, kas tipiski var saturēt operētājsistēmu, datu bāzi, programmēšanas vai programmu izpildes vidi, tīmekļa serveri, etc.
- Programmatūra kā serviss (SaaS – Software as a service): Pats augstākais pakalpojumu līmenis. šajā modelī lietotājam tiek nodrošināta piekļuve lietotnei.

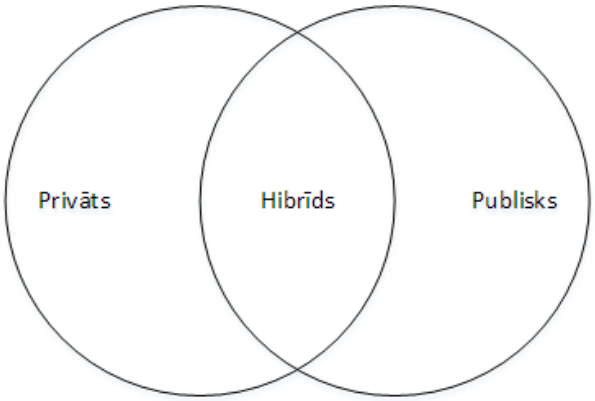
Dažādās tīmekļa vietnēs šī modeļa interpretācija nedaudz variē. Eksistē resursi, kur vispār tiek prognozēts, ka šī iemesla dēļ no šī modeļa drīzumā nāksies atteikties [7].

Vēl viens veids kā dalīt mākoņus ir pēc pielietojuma .

- Privāti mākoņi – infrastruktūra, kura ir paredzēta vienas organizācijas vajadzību apmierināšanai. Šāda veida mākoņa ieviešana un uzturēšana organizācijā ir nopietns

solis, jo tā izslēdz primāro reklamēto mākoņa priekšrocību – tā neatbrīvo organizāciju no fiziskās infrastruktūras uzturēšanas. Organizācija ir pati atbildīga par aparatūras un programmatūras nodrošinājumu servisu sniegšanai.

- Publiski mākoņi – Mākoņa sniegtie servisi tiek piedāvāti plašas publikas lietošanai izmantojot internetu. Publiski mākoņi var piedāvāt gan maksas gan bezmaksas servisus. Izmantojot šādu mākoņu pakalpojumus ir jāņem vērā drošības apsvērumi, kas rodas pārsūtot datus caur Internetu, kā arī uzticot sava uzņēmuma informācijas glabāšanu pakalpojuma sniedzējam.
- Hibrīdmākoņi – vairāku mākoņu tipu apvienojums, kur organizācija izmanto gan privāta gan publiska mākoņa sniegtos pakalpojumus.



Attēls 3.2 Mākoņu klasifikācijas pēc pielietojuma diagramma.

Dalījumā pēc pielietojuma ir iespējams atrast arī citas uzskaitītas kategorijas, taču tie pēc būtības šķiet vai nu atvasinājumi no šiem trīs, vai nu mēģinājumi apzīmēt cita veida projektus ar mākoņskaitļošanas terminu. Reizēm globālajā tīmeklī var atrast apgalvojumus, kur izkaistītās skaitļošanas projekti tādi kā Folding@home tiek dēvēti par mākoņskaitļošanas apakšklasi. Tiesa tas nav korekts apzīmējums, kaut arī ir atrodami tīmekļa resursi, kur ir pētītas iespējas šī projekta skaitļošanas procesu darbināt uz dažādām mākoņplatformām.

Abi šie dalījumi eksistē neatkarīgi viens no otra. Šī pētījuma ietvaros mūs interesē pamatā pirmais dalījums – pēc pakalpojuma veida.

4 PĒTĪTIE PRODUKTI UN RISINĀJUMI

4.1 Docker

Docker dotajā brīdī ir viena no visstraujāk augošajām OS virtualizācijas platformām. Tā sastāv no Docker dzinēja - lietotņu iesaiņošanas rīka, kā ari Docker Hub mākoņservisa lietotņu izplatīšanai un darbplūsmu automaizācijai [8]. Par pamatu Docker izmanto Linux konteinerus un skaitās zem operētājsistēmas virtualizācijas rīkiem. Izmaiņas failsistēmā tiek veiktas izmantojot slāņus (layers), kuriem iespējams veikt momentuzņēmumus, atriti un citas darbības.

Docker primārais reklāmas lozungs ir ātrs un ērts lietojumu izstrādes un publicēšanas process. Ir pieejami daudz gatavi konteineru attēli un tas izmanto slāņu mehānismu, lai ātri un ērti varētu izveidot konteineru, kas satur gatavu vidi lietojuma izstrādei. Lietotājam ir pašam iespējams saglabāt savu izveidoto lietotnes konteineri kā attēlu un īsā laikā pārvietot no izstrādes vides uz testa vidi un pēc tam tālāk uz produkciju.

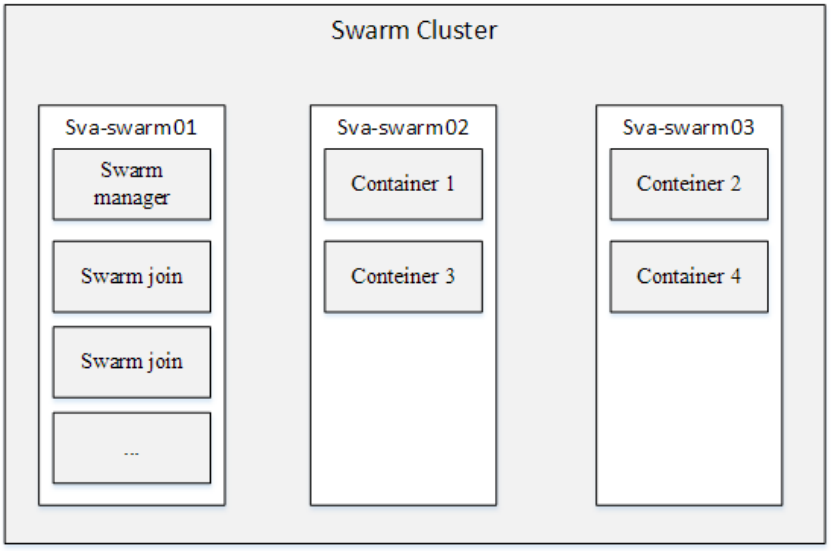
Kaut arī Docker pamatfunkcija ir atsevišķu procesu konteinerizācija, tas lieliski spēj strādāt arī ar konteineriem, kuri satur pilnu operētājsistēmas vidi. Šāda funkcionalitāte komplektā ar vienkārši lietojamiem pārvaldības rīkiem ir ļoti pievilcīga, domājot par konteineri kā infrastruktūras pakalpojumu.

Docker vēl aizvien atrodas ļoti straujā attīstības fāzē, tam regulāri tiek papildināa funkcionalitāte un regulāri nāk klāt jauni, dažādās izstrādes stadijās esoši pārvaldības rīki. Šī darba izstrādes ietvaros izveidoju arī nelielu Docker klasteri no trīs virtuāliem serveriem. Izmēģināju divus rīkus – Shipyard un Docker klasteru risinājumu Swarm.

4.1.1 Docker Swarm

Swarm ir paša Docker vietējs klasteru pārvaldības rīks [9]. Tā uzstādīšana ir iespējama vai nu kā serviss uz atsevišķa servera vai arī izmantojot Docker attēlu, kas pieejams Docker Hub. Es izmantoju otro variantu un ar šo paņēmienu klastera izveidošana, pieņemot, ka

virtuālās mašīnas, kas kalpos kā klastera mezgli, jau ir izveidoas un darbojas kā atsevišķi docker serveri, aizņem vien pāris minūtes. Rezultātā veidojas infrastruktūra, kur klastera pārvaldības process darbojas konteinerā uz viena no mezgliem. Izveidotā klastera shēma parādīta attēlā *Attēls* ***4.1***.



Attēls 4.1 ***Izveidotā Swarm klastera struktūra.***

Uz sva-swarm01 mezgla darbojas docker process, kurš darbina konteinerizētu klastera pārvaldnieka procesu un katram mezglam atsevišķu pievienošanas procesu. Šādam klasterim var arī pievienot jau esošus docker serverus, uz kuriem jau darbojas konteineri un swarm to ņems vērā.

```
root@sva-swarm01:~# docker -H tcp://172.30.100.11:8080 info
Containers: 13
Strategy: spread
Filters: affinity, health, constraint, port, dependency
Nodes: 2
 sva-swarm02: 172.30.100.12:2375
  └ Containers: 5
  └ Reserved CPUs: 0 / 2
  └ Reserved Memory: 0 B / 4.053 GiB
 sva-swarm03: 172.30.100.13:2375
  └ Containers: 8
  └ Reserved CPUs: 15360 / 2
  └ Reserved Memory: 1.25 GiB / 4.053 GiB
```

Attēls 4.2 ***Swarm klastera informācija***

Kā var redzēt attēlā *Attēls **4.2***, klasterī ir pievienoti divi mezgli, uz tiem ir izvietoti 13 konteineri, ieskaitot tos, kas pašreiz netiek darbināti, izvietošanas stratēģija ir *spread*, kas nozīmē, ka swarm mēģina vienlīdzīgi izdalīt konteinerus pa visu klasteri, tālāk tiek uzskaitīti filtri, kas tiek ņemti vērā pieņemot konteinera izvietošanas lēmumu, un pamatinformācija par katru no mezgliem.

Tā kā pašreizējā FreeBSD Jail risinājumā katram konteineram ir sava IP adrese, tad bija vēlme pārbaudīt, kā swarm rīkojas, kad ir nodefinēts atribūts –net=host, lai izmantotu nevis Docker pārvienojumu, bet piesaistītu servisu kādai no vairākām saimnieka IP adresēm.

Secināju, ka swarm savā konteineru izvietošanas loģikā neņem vērā vai norādītā IP adrese uz mezgla maz eksistē un negatīva rezultāta gadījumā konteiners tiek izveidots, bet netiek iedarbināts. Ņemot vērā, ka eksistē radniecības (*affinity*) filtrs, kurš ļauj noteikt, ka konteineru drīkst darbināt tikai uz mezgliem, kuri atbilst noteiktiem nosacījumiem, piemēram darbināt tikai uz tām mašīnām, uz kurām jau atrodas konteinera attēls, šajā situācijā prasās arī filtrs, kas nosaka, ka konteiners drīkst darboties tikai uz tā mezgla, kuram ir pievienota noteikta IP adrese. Tiesa jāpiezīmē, ka šāds serviss uz klastera būtu noderīgs tikai ļoti specifiskos gadījumos. Labāks risinājums IP adresu pārvaldībai būtu Docker tīkla realizācijas vietā izmantot kādu tīkla virtualizācijas risinājumu uz kādas no mākoņskaitļošanas platformām, vai vienkārši servisu, kurš nodrošina IP adresu pārvaldību klasterī. Neatradu nevienu risinājumu, kuram būtu norādīts, ka tas darbojas ar Swarm.

Diemžēl pašlaik Swarm uzstādīšanas dokumentācija, lai arī detalizēta, ir kļūdaina un vietām neviennozīmīgi saproama, kas atbilst personīgi novērotajai tendencei, ka Internetā pieejamā produktu dokumentācijas kvalitāte pēdējos gados strauji krītas. Swarm vēlaizvien atrodas izstrādes stadijā un tam ir pieejama tikai beta versija.

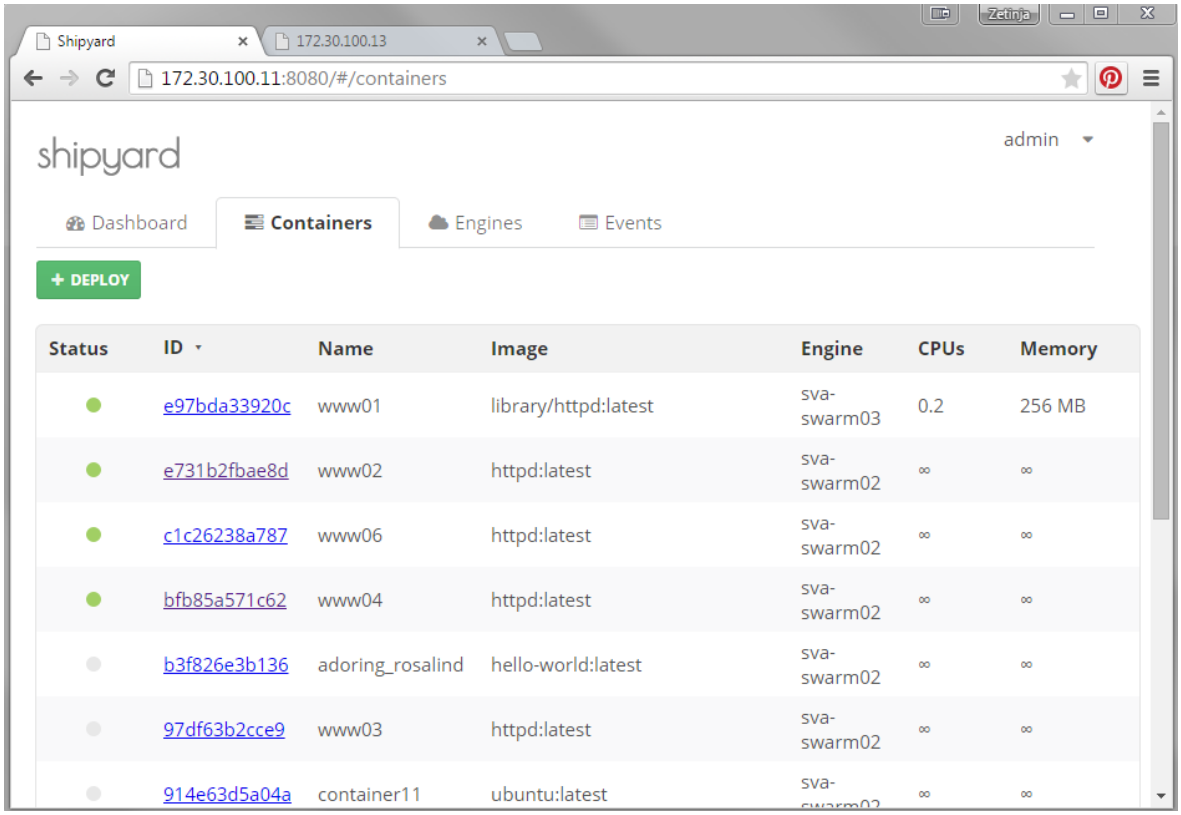
4.1.2 Shipyard

Vēlējos aplūkot arī kādu Docker klasteru pārvaldības rīku, kuram ir grafiska lietotāju saskarne. Tādu ir gana daudz, taču lielākoties tie atrodas izstrādes stadijā un ir ar ļoti ierobežotu funkcionalitāti, bet lieliem nākotnes plāniem.

Shipyard [10] kā jau lielākā daļa Docker lietotņu ir uzstādāms sekunžu laikā no Docker Hub. Tas ir darbināms gan no komandrindas, gan caur Interneta pārlūku. Klastera

mezglus pievienot izdevās ātri un vienkārši. Kā arī saskarne šķita pietiekoši viegli uztverama un intuitīva. Pašam Shipyard funkcionalitāte ir ierobežota – pārlūkot, izveidot, ieslēgt, izslēgt un dzēst konteinerus, bet tam ir pieejami vairāki paplašinājumi, ar kuriem to var papildināt ar žurnālēšanas, maršrutēšanas un slodzes sadalīšanas, rezerves kopiju veidošanas un citām funkcijām.

Attēlā *Attēls* ***4.3*** var aplūkot konteineru pārskata logu. Tajā ir dota pamatinformācija par konteinera stāvokli – ieslēgts/izslēgts, pamatdati par koneineru – ID, nosaukums, attēls un uz kura mezgla konteiners darbojas. Kā arī var redzēt konteineru atmiņas un CPU resursu ierobežojumus.



Attēls 4.3 ***Shipyard konteineru pārskata loga ekrānšāviņš***

Abi aplūkotie Docker pārvaldības produkti izskatās gana daudzsološi, lai būtu vērts sekot līdzi to attīstībai. Arī pats Docker, lai arī piegādā solīto pieredzi ar ātru lietotņu uzstādīšanu, manuprāt tam tomēr pietrūkst daudzas funkcijas, kuru nepieciešamība šķiet pašsaprotama.

Kā arī neviens no šiem rīkiem nerisina pastāvīgās datu glabāšanas jautājumu – kādā veidā nodrošināt, lai konteinera dati būtu pieejami tieši no tā mezgla, uz kura klastera pārvaldnieks ir izlēmis šo konteineru darbināt.

Viens no iespējamiem variantiem ir rast veidu, kā piesaistīt konteineru tam konkrētajam mezglam, uz kura glabājas šim konteineram nepieciešamie dati, bet tādā situācijā ir jābūt scenārijam, kā nodrošināt kļūmjpārlēces mehānismu uz citiem mezgliem, ja tas ir nepieciešams. Tiesa, ja ir kļūmjpārlēces mehānisms, kas ļauj pastāvīgo datu glabātuvei sekot konteineram, tad ir iespējams analoģisks mehānisms, kurš konteinera iedarbināšanas procesā piesaista datu glabātuvi attiecīgajam mezglam. Ņemot vērā, ka datu glabātuves piesiatīšana mezglam ir jāveic pirms konteinera iedarbināšanas, šīs funkcionalitātes atbalstam ir jābūt iekļautai klastera pārvaldības rīkā. Vēl viens variants ir izmnatot attālinātu datu glabātuvi, kura ir pieejama no visiem klastera mezgliem.

4.2 Openstack

OpenStack ir populārs atvērtā pirmkoda mākoņinfrastruktūru pārvaldības rīku kopums. Tas sastāv no vairākiem savstarpēji saistītiem projektiem un tam ir plašs konfigurāciju klāsts, kas ļauj izveidot savām vajadzībām individuāli piemērotu infrastruktūru. Projekta mērķis ir nodrošināt vienkāršu ieviešanu, plašu mērogojamību un bagātu funkciju klāstu.

OpenStack oficiālajā wiki norādīts, ka OpensStack arhitektūra pašreiz sastāv no 26 projektiem [11], taču to skaits laika gaitā pieaug, jo papildinoties funkcionalitātei projekti tiek sadalīti sīkākās daļās. Tālāk sarakstā identificēšu vairākus servisus, kuri parādās šī darba tekstā.

- Nova (compute) – serviss atbild par virtuālo mašīnu darbināšanu, plānošanu un pārtraukšanu.
- Horizon – informācijas panelis, kas nodrošina tīmekļa saskarni, caur kuru iespējams rīkoties ar OpenStack servisiem.
- Neutron – nodrošina tīklošanas servisu pārējiem OpenStack produktiem. Ļauj lietotājam veidot tīklus un pieslēgumus tiem.
- Glance – uzglabā un pārvalda virtuālo mašīnu attēlus. Nova serviss izmanto Glance jaunas virtuālās mašīnas veidošanas procesā.

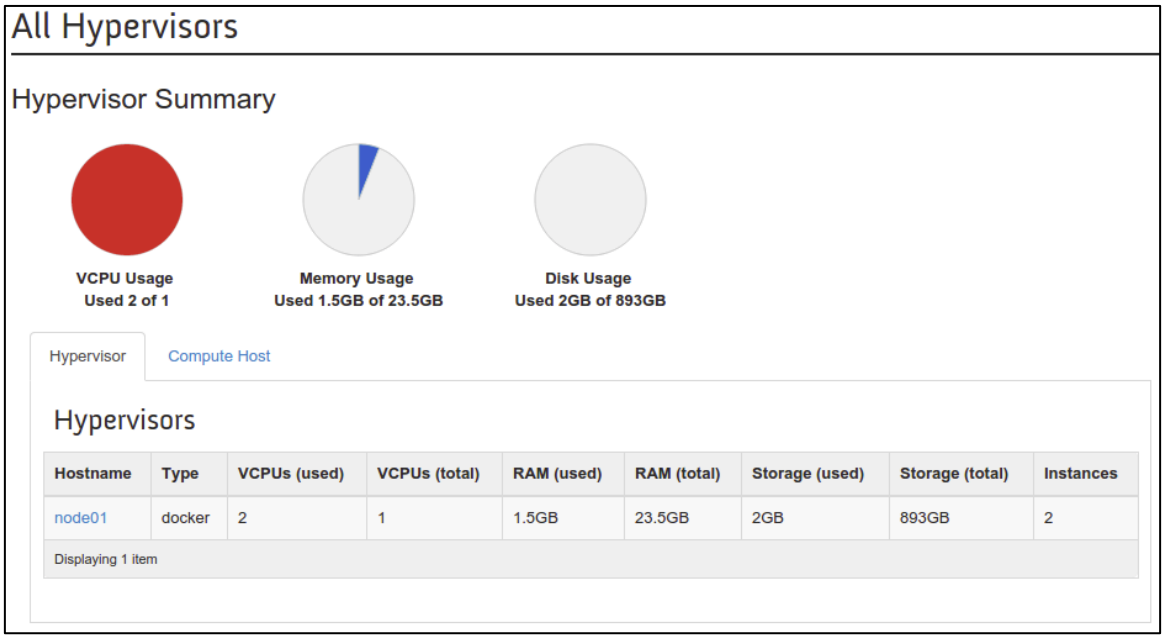
Darba izstrādes ietvaros tika izmēģināta OpenStack konfigurācija, kur Docker tiek izmantots kā Nova servisa virtualizācijas hipervizors. Šī funkcionalitāte dotajā brīdī nav iekļauta Nova projektā, bet tiek attīstīta paralēli. Šāda nodalījuma mērķis ir veicināt Nova-Docker dzinēja projekta straujāku attīstību. Tiek plānots, Docker atbalsts tiks iekļauts atpakaļ Nova izstrādes kokā nupat iznākušajā OpenStack Kilo versijā [12].

Docker virtualizācijas dzinis tika pievienots LU Linux centrā esošai OpenStack Juno versijai. Instalācijas process ir aprakstīts OpenStack oficiālajā wiki, Docker sadaļā [12], taču tajā aprakstītā procedūra, programmatūras arhitektūras izmaiņu dēļ, strādā tikai uz jaunākās – Kilo versijas, tādēļ nācās meklēt vairākus alternatīvus tīmekļa resursus, kā informācijas avotus, instalācijas procesa veikšanai un arī tad uzstādīšanas process nenoritēja bez kļūdām.

Īsumā instalācijas proces sastāv no vairākiem soļiem:

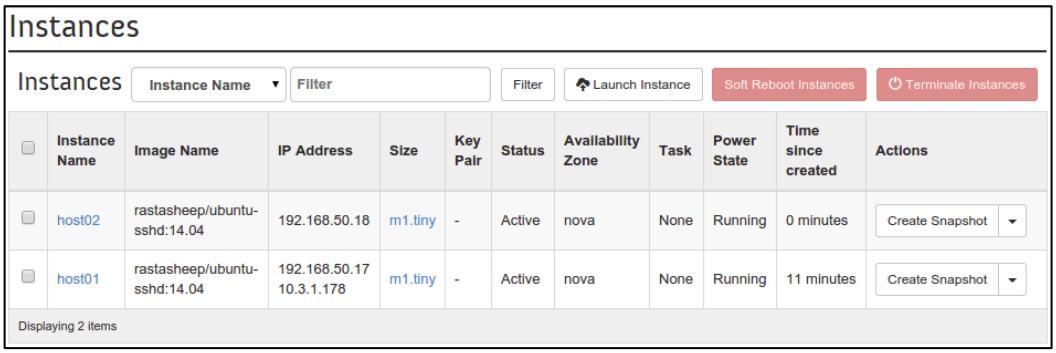
- Docker uzstādīšana uz Nova servera
- Nova lietotāja pievienošana docker grupā
- Nova-Docker virtualizācijas dziņa un moduļu instalācijas
- Nova servisa konfigurēšanas – jānorāda, lai Nova serviss izmanto docker dzinēju.
- Glance servisa konfigurēšanas – jāpievieno docker konteineru formāts konfigurācijas failā.

Pēc Nova servisa pārstartēšanas un ilgas un apnicīgas traucējummeklēšanas procedūtas būtu jābūt strādājošai OpenStack sistēmai balstītai uz Docker virtualizācijas platformu. Atverot Horizon saskarni un ieejot sadaļā Admin > System > Hypervisors būtu jāredz *Attēls* ***4.4*** attēlā redzamā informācija.



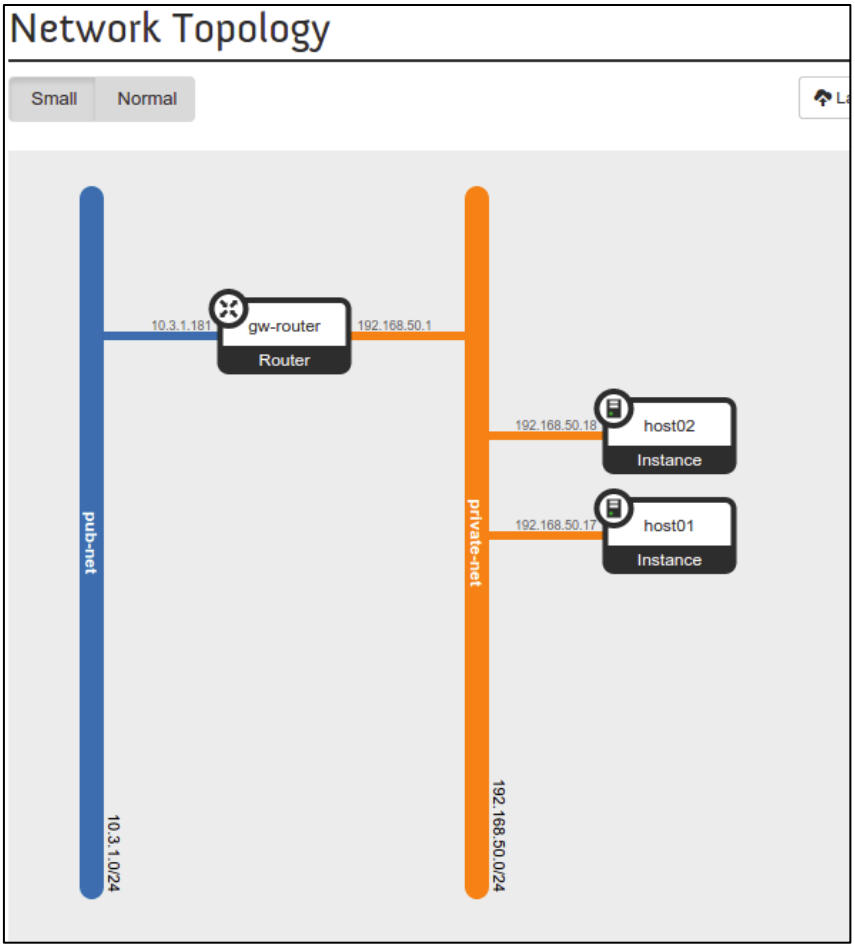
Attēls 4.4 ***OpenStack tīmekļa saskarnes hipervizoru pārskata sadaļa.***

Pirms konteinera izveides, interesējošie konteineru attēli ir jāielādē no reģistra izmantojot Docker līdzekļus un pēc tam tie ir jāieimportē Glance servisā. Pēc tam var doties uz tīmekļa saskarni un sadaļā Project > Compute > Instances ***Attēls 4.5*** izveidot jaunu konteineru.



Attēls 4.5 ***OpenStack instanču pārvaldības sadaļa.***

Kā redzams *Attēls* ***4.5*** attēlā, tika izveidoti divi konteineri host01 un host02, balstīti attēlu, kas satur Ubuntu 14.04 operētājsistēmu un SSH servisu. Attēlā ***Attēls 4.6*** ir redzams, ka konteineri ir izvietoti tiem speciāli izveidotā privātā tīklā, kurš caur maršrutētāju ir savienots ar publisko tīklu.



Attēls 4.6 ***Uz OpenStack izveidoto konteineru tīkla infrastruktūras pārskats Horizon saskarnē***

Atverot komandrindu un aplūkojot pārvienojumu informāciju redzam, ka Docker pilnībā izmanto OpenStack Neutron nodrošināto tīkla servisu un konteiners ir pievienots Neutron veidotajam tīkla pārvienojumam. Attēlā ***Attēls 4.7*** redzam, ka docker0 pārvienojumam neviens konteiners nav pievienots, savukārt host01 izveidotais konteiners ir pievienots brq6a750cd7-9f, kurš ir Neutron servisa veidotais pārvienojums.

```
user@node01:~$ brctl show
bridge name     bridge id               STP enabled     interfaces
brq6a750cd7-9f          8000.f01fafda3cda       no              eth1.71
                                                                tape60a68e4-4c
docker0         8000.56847afe9799       no
user@node01:~$
```

Attēls 4.7 ***Servera pārvienojumu informācijas izvads komandrindā***

Izvadā neparādās host02 savienojums, jo ekrānšāviņi nav hronoloģiski un attēls *Attēls **4.7*** ir uzņemts pirms host02 konteinera izveidošanas.

```
user@node01:~$ sudo docker exec -i -t 11e00ce68566 bash
[sudo] password for user:
root@instance-000000c9:/# ifconfig
lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

nse60a68e4-4c Link encap:Ethernet  HWaddr fa:16:3e:6e:60:31
          inet addr:192.168.50.17  Bcast:192.168.50.255  Mask:255.255.255.0
          inet6 addr: fe80::f816:3eff:fe6e:6031/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:13 errors:0 dropped:0 overruns:0 frame:0
          TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1038 (1.0 KB)  TX bytes:648 (648.0 B)

root@instance-000000c9:/# ping tvnet.lv
PING tvnet.lv (159.148.168.103) 56(84) bytes of data.
^C64 bytes from 159.148.168.103: icmp_seq=1 ttl=55 time=4.21 ms

--- tvnet.lv ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 4.211/4.211/4.211/0.000 ms
root@instance-000000c9:/# ping tvnet.lv
PING tvnet.lv (159.148.168.103) 56(84) bytes of data.
64 bytes from real.tvnet.lv (159.148.168.103): icmp_seq=1 ttl=55 time=1.84 ms
64 bytes from real.tvnet.lv (159.148.168.103): icmp_seq=2 ttl=55 time=1.65 ms
64 bytes from real.tvnet.lv (159.148.168.103): icmp_seq=3 ttl=55 time=1.77 ms
^C
--- tvnet.lv ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2000ms
rtt min/avg/max/mdev = 1.650/1.755/1.840/0.086 ms
```

Attēls 4.8 ***koneinera host01 tīkla informācija***

Pieslēdzoties vienam no izveidotajiem konteineriem tika konstatēts, ka konteiners un tā tīkls ir pilnībā funkcionāli ***Attēls 4.8***. Tam ir piešķirta IP adrese iepriekš izveidotajā privātajā tīklā un darbojas ping uz adresi Internetā tvnet.lv. Tas nozīmē, ka ir saņemta korekta tīkla konfigurācijas informācija un darbojas arī maršrutēšana.

Veicot pārbaudi vai izdodas no viena konteinera pieslēgties uz pie otra konteinera SSH servisa un tā noritēja veiksmīgi.

Secinājums – Docker šķiet saprātīga alternatīva infrastruktūras servisa nodrošināšanai uz OpensStack bāzētos mākoņos. Tiesa pirms izmantot šādu arhitektūru produkcijā, es noteikti nogaidītu līdz Docker dzinis tiek iekļauts Nova pirmkodā, lai likvidētu problēmas uzstādīšanas procesā un samazinātu kļūdu iespēju lietošanas laikā.

4.3 VMware

VMware ir viena no populārākajām komerciālajām virtualizācijas patformām. Mūsu situācijā platformas galvenā priekšrocība ir tāda, ka organizācijas datu centrā jau eksistē VMware vSphere risinājums un ir apmācīti specialisti darbā ar šo produktu.

Šobrīd, kad notiek ļoti strauja mākoņskaitļošanas platformu attīstība VMware ir ņēmusi aktīvu dalību risinājumu izstrādē. Neatkarīgi no sava piedāvātā mākoņu veidošanas produkta vCloud Suite, ir izstrādāts arī VMware un OpenStack integrācijas risinājums. Diemžēl praksē izmēģināt, kā tas darbojas, man īsti nav, kur.

VMware atbalsta operētājsistēmas līmeņa virtualizāciju un iesaistās tās rīku attīstības procesā. Ir izsludināta sadarbība ar Docker. VMware BDE (Big Data Extensions) paplašinājumā ir iekļauts Kubernetes atbalsts [13], kas savukāt nozīmē, ka izmantojot Kubernetes ir iespējams no VMware saskarnes pārvaldīt Docker konteinerus, bet kā arī tiek solīts, ka drīzumā var gaidīt vēl kaut kāda līmeņa Docker integrāciju ar vmware produktiem.

4.4 Kubernetes

Kubernetes [14] ir Linux konteineru klasteru pārvaldības risinājums, kas ir tieši atvasināts no Google Borg. Google vairāk kā desmit gadus izmanto principu, kurš dotajā brīdī tiek plaši dēvēts par jaunu paradigmu un nākotnes skaitļošanas tehnoloģijām – proti koneinerizētu servisu pārvaldības pieeju, izmantojot Linux chroot jails un cgroup mehānismus [15]. Kubernetes ir izstrādāts ņemot pieredzi, kas gūta darbā ar Borg, un tajā tiek ieviesti jauni principi un funkcionalitāte, izmantojot tehnoloģijas, kuras guvušas vispārēju atzinību un popularitāti salidzinoši nesen. Kubernetes atbalsta Docker konteinerus un izstrādātājiem ir lieli plāni attiecībā uz šī produkta attīstību un popularitātes pieaugumu.

Kubernetes izmanto etcd servisu klastera mezglu pārvaldībai un atpazīšanai. Kubernetes klastera risinājumam ir *master-slave* arhitektūra un master mezgls var vienlaicīgi pildīt abas šīs lomas. Kā arī Kubernetes iekļauj gan plānošanas mehānismus, gan augstas pieejamības, gan slodzes izlīdzināšanas mehānismus.

Kubernetes darbības principa pamatā ir konteineru čaulas (*pod*). Tās ir konteineru vienības, grupas, kuras tiek administrētas kā individuāli objekti. Visi vienā čaulā ietilpstošie konteineri darbosies uz viena un tā paša mezgla. Katrai čaulai tiek automātiski piešķirta IP adrese no sava iekšējā tīkla. Izmantojot šo adresi no citām čaulām ir pieejami visi tajā esošie konteinerizētie servisi. Tiesa IP adresu pārvaldība ir dinamisks process, līdz ar to tiešai, savstarpējai konteineru komunikācijai tas nav izmantojams. Kā arī, atšķirībā no Docker, Kubernetes pilnībā nodala čaulas no saimnieksistēmas tīkla. Arī situācijās, kad kāds no čaulā esošajiem servisiem tiek publicēts uz kādu no servera portiem, šī operācija tiek veikta pilnīgi caurspīdīgi un konteinerā esošais serviss to neredz.

Slodzes izlīdzināšanas mehānisma nodrošināšanai tiek izmantots papildus abstrakcijas līmenis – Serviss. Servisss sastāv no čaulām, kurām ir nodefinētas attiecīgās birkas un katram servisam ir nosaukums, pēc kura jebkurš klasterī esošs konteiners spēj pie tā pieslēgties. Kubernetes seko līdzi, uz kuriem mezgliem darbojas servisa čaulas un izlīdzina starp tām servisa slodzi. Īpaši pievilcīga ir funkcionalitāte šķiet situācijā, kad nepieciešams konteinerizētajam servisam pieķļūt izmantojot IP adresi, kas likvidē konkurenci uz portiem, kad uz servera tiek izvietotas vairākas vienādas lietotnes.

Tā kā Kubernetes izmanto Docker, kurš, kā iepriekš minēts, ir izmantojams ne tikai atsevišķu procesu, bet arī konteineru, kas satur pilnu operētājsistēmu pārvaldībai. Tādā

gadījumā, apvienojot šo funkcionalitāti ar Kubernetes papildus ieviestajiem abstrakcijas līmeņiem, veidojas plašs izveidojamo infrastruktūras konfigurāciju klāsts.

Fakts par Kubernetes, kas man šķiet īpaši saistošs ir, ka tiek izstrādāti rīki, kas spēj pārvaldīt ne tikai Docker konteinerus, bet arī virtuālās mašīnas. Ņemot vērā, ka datu centrā ir VMware risinājums, tad visinteresantākais šķiet Kubernetes vSphere apgādnieks [16], kurš ļauj Kubernetes pārvaldīt virtuālās mašīnas šajā vidē. Izveidojas situācija, kur eksistē rīks, kas spēj gan pārvaldīt konteinerus, gan virtuālās mašīnas, kur šos konteinerus izvietot.

Izvērtējot Kubernetes konteineru pārvaldības rīku datu centra infrastruktūras optimizācijai, atkal pievērsos jautājumam par konteineru IP adresu pārvaldību klasterī un uzgāju rīku, kas saucas Kiwi [17]. Īsumā tas ir serviss, kurš monitorē Kubernetes aktivitāti attiecībā uz servisiem un attiecīgi tiem servisiem, kuriem ir norādīts publiskās IP adreses atribūts, piešķir šo IP adresi attiecīgā klastera mezglam un pievieno filtrus ugunsmūrī. Šādi projekti kā Kiwi liecina, ka ir vēl cilvēki, kuriem šādi jautājumi par konteineru tīklošanu klasterī ir aktuāli.

Pamēģināju arī izveidot Kubernetes klasteri no trīs virtuālajām mašīnā uz Ubuntu 14.04 platformas.

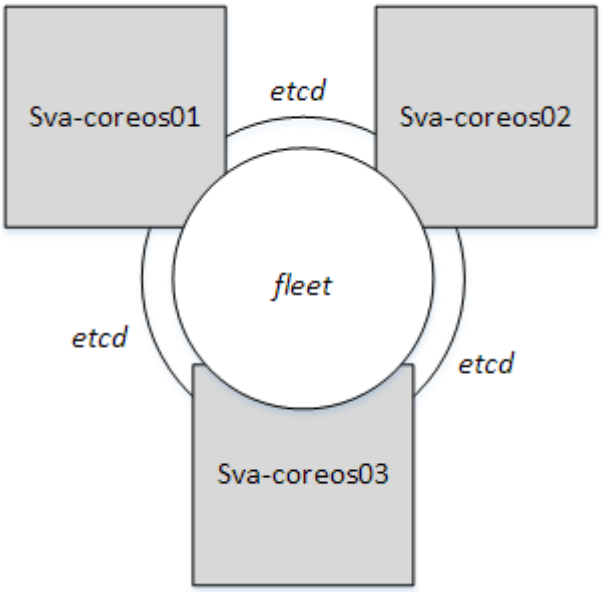
4.5 CoreOS

CoreOS ir kompakta, uz Linux kodola balstīta operētājsitēma, kuras darbības pamatprincips ir balstīts uz lietotņu konteinerizāciju un viens no tās uzbūves stūrakmeņiem ir Docker. CoreOS programmatūras uzstādīšanai neizmanto klasiskos pakotņu pārvaldniekus, bet gan pilnīgi visas lietotnes tiek uzstādītas ieslēgtas konteinerī kopā ar tām nepieciešamajām atkarībām. Dotajā brīdī CoreOS ir viena no populārākajām operētājsistēmām Docker konteineru mitināšanai.

CoreOS ir savietojams ar praktiski visām populārākajām virtualizācijas platformām un tam ir pieejami virtuālo mašīnu attēli teju visos populārākajos virtualizācijas formātos, kā arī projektam ir pieklājīga dokumentācija. Operētājsistēmā pēc noklusējuma ir uzstādīts *etcd* serviss, kas nodrošina jaunu mezglu atklāšanu un konfigurācijas pārvaldību un redundanci.

Tāpat CoreOS ir iekļauts fleet serviss, kurš nodrošina vienu servisu pāvaldības sistēmu visā klasterī. Ar fleetd palīdzību var nodrošināt tādas funkcijas, kā redundance un kļūmjpārlēce.

Darba izstrādes ietvaros izveidoju nelielu CoreOS klasteri ***Attēls 4.9*** uz trīs VMware virtuālajām mašīnām. Klastera uzstādīšanai izmantoju vienu no fiziska servera sagatavošanas procedūrām – instalēšanu no skripta. Katra mezgla uzstādīšanas galvenais elements ir cloud-config fails, kurā tiek nodefinēti klastera parametri.



Attēls 4.9 ***Izveidotā CoreOS klastera shēma.***

CoreOS uzstādīšanas dokumentācijā ir apgalvots, ka CoreOS ir iespējams uzstādīt dažu minūšu laikā un tas ari nav tālu no patiesības.

Pamatdarbības, ko veicu klastera mezgla izveidei ir :

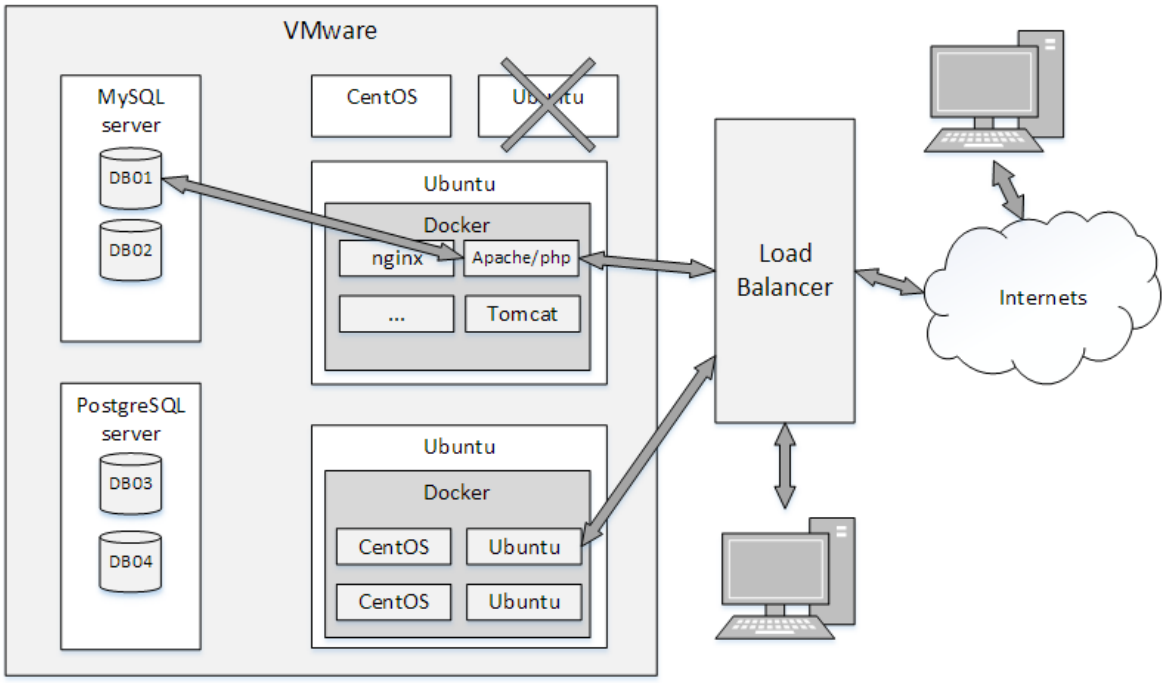
- virtuālās mašīnas izveide,
- virtuālās mašīnas sāknēšana no Ubuntu darbstacijas instalācijas diska kā LiveCD,
- cloud-config faila izveide, tai skaitā klastera unikālā marķiera ģenerēšana,
- instalācijas skripta palaišana, norādot laidiena kanālu „stable” un ceļu uz cloud-config failu

Process patiešām aizņēma solītās dažas minūtes, no tā var secināt, ka CoreOS uzstādīšana pilnībā atbilst apgalvojumam par ātru klastera mezglu pievienošanu.

5 DATU CENTRA INFRASTRUKTŪRAS SERVISA ARHITEKTŪRAS VĪZIJA

Veicot pašreiz populāro OS līmeņa virtualizācijas rīku izpēti, var secināt, ka Linux konteineri ir pareizais virziens, kurā skatīties attiecībā uz datu centra infrastruktūras servisa nodrošināšanas vajadzībām. Izmantojot Docker konteineru pārvaldības rīku ir iespējams izveidot homogēnu arhitektūru *Attēls* ***5.1,*** kuras galvenā priekšrocība ir vienkāršs evolūcijas process izmantojot Docker konteineru tehnoloģijas priekšrocības, kā arī standartizējot klientu piekļuves veidus konteinerizētajiem servisiem un infrastruktūrai.

Jāsaprot, ka ar plānotajām izmaiņām nav mērķis izveidot mākoņinfrastruktūru, bet gan pielietot pētījumā iegūto informāciju, lai veiktu izmaiņas, kuras, pēc to ieviešanas, atvieglos administratoru darbu, samazinās pakalpojuma piegādes un kļūdu novēršanas reakcijas laikus un to ieviešanas laikā būtiski nemazinās pakalpojuma kvaliāti. Kā arī jaunizveidotā infrastruktūra būs spējīga pielāgoties virtualizācijas tehnoloģiju attīsības tendencēm nākotnē.



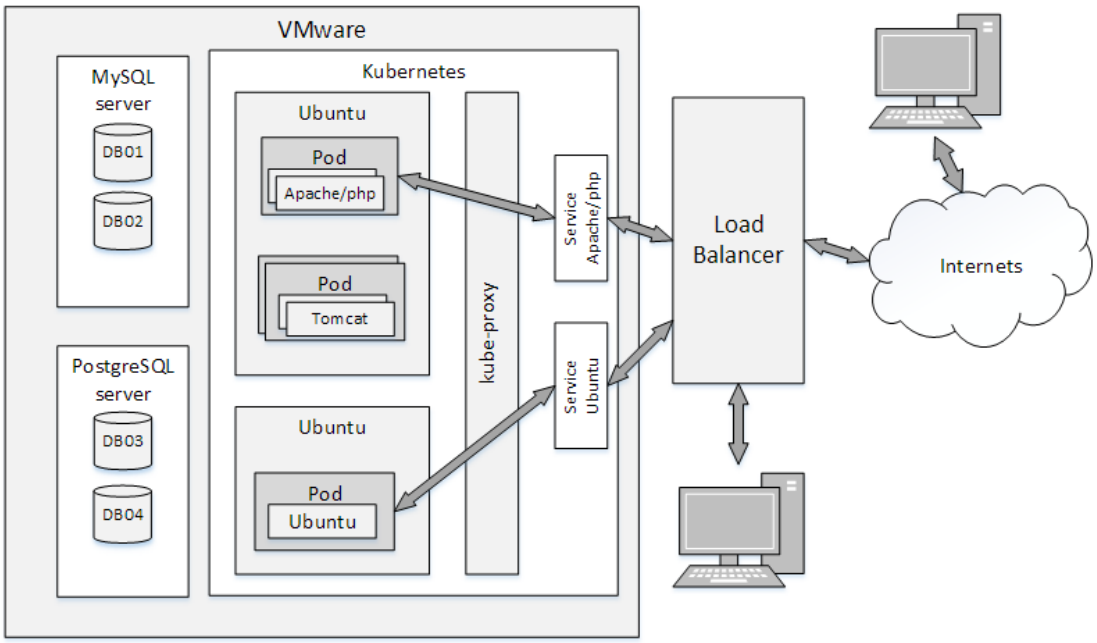
Attēls 5.1 ***Infrastruktūras attēlojums pēc pirmā izmaiņu soļa veikšanas.***

Jau sākotnēji aizstājot FreeBSD Jail virtualizācijas risinājumu ar Linux konteineriem tiek samazināts dažādu serveru operētājsistēmu skaits un līdz ar to vienkāršota to pārvaldība, vienlaicīgi atbrīvojoties no problēmām, kuras ir saistītas ar FreeBSD Jail uzturēšanu.

Nākošais solis ir infrastruktūras servisa pakāpeniska pārcelšana uz Docker konteineriem. Tas nozīmē, ka jauni infrastruktūras pieprasījumi tiktu apmierināti ar konteineru tehnoloģiju, savukārt esošās virtuālās mašīnas turpinātu darboties līdz tās klientiem vairs nav nepieciešamas.

Trešais izmaiņu moments ir nodefinēt visiem klientiem vienotu piekļuves ceļu pie konteinerizētajiem pakalpojumiem izmantojot datu centra slodzes sadalītāju. Šis solis ļauj veikt no klienta puses pilnīgi caurspīdīgas izmaiņas iekšējā datu centra infrastruktūrā, kā arī mazināt šīs te infrastruktūras sarežģītību tādējādi atvieglojot uzturēšanu.

Tehnoloģijām ar laiku attīstoties un docker koneineru klasteru pārvaldības rīkiem nobriestot, kļūstot stabilākiem un funkcionāli bagātākiem, pastāv iespēja, ka nākotnē varētu tikt apsvērta klasteru risinājuma ieviešana *Attēls* ***5.2***. Ņemot vērā Docker konteineru vieglo pārnesamību, šāda pāreja būtu nodrošināma no klienta puses pilnīgi caurspīdīgi.



Attēls 5.2 ***Uz Kubernetes balstīts Docker konteineru klastera risinājuma variants***

Tiesa izvietojot konteinerus klasterī ir jāņem vērā vairāki būtiski momenti. Viens no tiem ir konteineru IP adresu jautājums, jau iepriekš minēts sadaļās, kur tiek aprakstīts Swarm

un Kubernetes. Tas nozīmē, ka arī klasterētā vidē katram konteineram arī pārvietojoties no viena mezgla uz otru, ir jāspēj saglabāt sava IP adrese, kura ir tieši pieejama no slodzes sadalītāja. Papildus ir jāņem vērā, ka infrastruktūras konteiners nesastāv no viena paša servisa, bet svarīgi ir arī nodrošināt, lai konteinera dati sekotu konteineram, ja tas tiek pārvietots uz citu klastera mezglu. Tātad ir nepieciešams lai eksistētu mehānisms, kurš pārvalda konteineru datus. Kā arī, lai pieņemu galēju lēmumu par šāda risinājuma ieviešanu produkcijā ir nepieciešams veikt rūpīgus iespējamo situāciju scenāriju testus, lai apzinātu iespējamos „zemūdens akmeņus".

SECINĀJUMI

Dotajā brīdī ar Linux konteineru tehnoloģiju straujo attīstību, Operētājsistēmas līmeņa virtualizācijas koncepts piedzīvo ārkārtīgi strauju popularitātes pieaugumu. Mākoņskaitļošana šobrīd ir vispopulārākā tendence un teju visbiežāk lietotais vārds IT pasaulē. Ņemot vērā abus šos faktus nav brīnums, ka patreis ir tik ārkārtīgi liels platformu skaits, kas strādā pie tā, lai šīs abas tehnoloģijas apvienotu un iegūtu no tām maksimālo funkcionalitāti.

Dotajā brīdī ideja par Docker, kā infrastruktūras pakalpojuma virtualizācijas rīku eksistē, bet tā nav ne tuvu tik populāra, kā Docker pielietojamība servisu konteineru pārvaldībai. Šī situācija ir diezgan loģiska, ņemot vērā, ka Docker tiek izstrādāts ar tieši šādu mērķi. Tas nozīmē, ka arī lielākā daļa Docker klasteru pārvaldības rīku tiek attīstīti paturot prātā, ka tie tiks izmantoti lietotņu konteineru darbināšanai. Tajā pašā laikā visi šie rīki tiek veidoti ar domu par savstarpēju integrāciju un ir iespējams no šiem projektiem uzbūvēt saprātīgu risinājumu infrastruktūras konteineru vajadzībām. Pavisam cits jautājums ir, vai laika un resursu ieguldījums, lai šo risinājumu izplānotu un ieviestu, atmaksājas, ja tas tiek izvietots uz kādas no virtualizācijas platformām.

Pētot situāciju un izmēģinot dažādus produktus, nācu pie secinājuma, ka operētājsistēmas līmeņa virtualizācija ir vērā ņemama alternatīva pilnās virtualizācijas infrastruktūrai, paturot prāta ierobežojumus, kas rodas dalot kodolu ar saimnieksistēmu un pārējiem konteineriem. Eksperimentējot uz OpenStack platformas ar Docker virtualizācijas dzini secināju, ka uz šīs mākoņskaitļošanas platformas no lietojamības un infrastruktūras izveides viedokļa konteineru pārvaldīšana ir vienlīdz ērta kā virtuālo mašīnu pārvaldība.

Izmēģinot dažādus rīkus, kas ir pielietojami, kā mākoņskaitļošanas platformu elementi, secināju, ka lielākoties šo rīku attīstība tiek virzīta ņemot vērā populārās tendences mūsdienu IT sfērā – tiek nodrošinātas plašas savstarpējās integrācijas iespējas, kas ļauj veidot daudz un dažādas IT infrastruktūras ar plašu funkcionalitāti. Tajā pašā laikā ļoti liela daļa no šiem darbā aplūkotajiem konteineru pārvaldības rīkiem atrodas dažādās izsrādes stadijās un vēl nav gatavi pielietošanai produkcijas vidē. To straujā attīstība nozīmē, ka bieži mainās to funkcionalitāte un lietojamība, bet tajā pašā laikā var just, ka publicētā dokumentācija netiek līdzi izstrādes tempiem.

Pēdējais secinājums par šo tēmu ir, ka šis laiks vienlaicīgi ir gan labs, gan slikts šādu infrastruktūras izmaiņu plānošanai. Labs tas ir no viedokļa, ka ir plašas izvēles iespējas, kādus produktus lieot un kā tos savstarpēji kombinēt. Slikts tas ir no tāda viedokļa, ka lielāk daļa no šiem produktiem ne tuvu nav gatavi darbināšanai datu centra produkcijas vidē.

IZMANTOTĀ LITERATŪRA UN AVOTI

[1] Valsts valodas centrs, «Terminoloģijas datu bāze,» [Tiešsaiste]. Available: http://termini.vvc.gov.lv/?term=IP+adrese&from=1&to=1&subject%5B%5D=6. [Piekļūts 31 05 2015].

[2] «ezjail – Jail administration framework,» [Tiešsaiste]. Available: http://erdgeist.org/arts/software/ezjail/ . [Piekļūts 25 05 2014].

[3] «FreshPorts -- ports-mgmt/pkg,» [Tiešsaiste]. Available: http://www.freshports.org/ports-mgmt/pkg. [Piekļūts 30 05 2015].

[4] «History of the Internet,» [Tiešsaiste]. Available: http://www.newmedia.org/history-of-the-internet.html. [Piekļūts 24 05 2015].

[5] «1.1.1. Brief History of Virtualization,» [Tiešsaiste]. Available: http://docs.oracle.com/cd/E26996_01/E18549/html/VMUSG1010.html. [Piekļūts 26 05 2015].

[6] S.S.Jadhav, Advanced Computer Architecture and Computing, Technical Publications, 2009.

[7] B. Butler, «Forget about IaaS, PaaS and SaaS - it's all about the platform,» [Tiešsaiste]. Available: http://www.networkworld.com/article/2898133/cloud-computing/forget-about-iaas-paas-and-saas-it-s-all-about-the-platform.html. [Piekļūts 25 05 2015].

[8] «What is Docker?,» [Tiešsaiste]. Available: https://www.docker.com/whatisdocker/. [Piekļūts 26 05 2015].

[9] «Docker Documentation,» [Tiešsaiste]. Available: https://docs.docker.com/swarm/. [Piekļūts 31 05 2015].

[10] «Shipyard,» [Tiešsaiste]. Available: https://shipyard-project.com/. [Piekļūts 31 05 2015].

[11] «OpenStack,» [Tiešsaiste]. Available: https://wiki.openstack.org/wiki/Main_Page. [Piekļūts 29 05 2015].

[12] «Docker - OpenStack,» [Tiešsaiste]. Available: https://wiki.openstack.org/wiki/Docker. [Piekļūts 29 05 2015].

[13] VMware, «Container Orchestration on vSphere with Big Data Extensions,» [Tiešsaiste]. Available: https://labs.vmware.com/flings/big-data-extensions-for-vsphere-standard-edition. [Piekļūts 30 05 2015].

[14] «kubernetes,» [Tiešsaiste]. Available: http://kubernetes.io/. [Piekļūts 30 05 2015].

[15] Google Inc., «Large-scale cluster management at Google with Borg,» Bordeaux, 2015.

[16] «Kubernetes, Getting started with vSphere,» [Tiešsaiste]. Available: https://github.com/GoogleCloudPlatform/kubernetes/blob/master/docs/getting-started-guides/vsphere.md. [Piekļūts 31 05 2015].

[17] L. Kellogg-Stedman, «Kiwi,» [Tiešsaiste]. Available: https://github.com/larsks/kiwi/. [Piekļūts 30 05 2015].

Bakalaura darbs „OS konteineru pielietojamība mūsdienu infrastruktūras mākoņos” izstrādāts LU Datorikas fakultātē.

Ar savu parakstu apliecinu, ka pētījums veikts patstāvīgi, izmantoti tikai tajā norādītie informācijas avoti un iesniegtā darba elektroniskā kopija atbilst izdrukai.

Autore: _____ Liene Rieksta

Rekomendēju/nerekomendēju darbu aizstāvēšanai

Vadītājs: lektora p.i. Dr.sc.comp. Leo Trukšāns _____ 01.06.2015

Recenzents: pasniedzējs Mg.sc.comp. Reinholds Zviedris

Darbs iesniegts Datorikas fakultātē 01.06.2015.

Dekāna pilnvarotā persona: vecākais metodiķis Ārija Sproģe _____

Darbs aizstāvēts bakalaura gala pārbaudījuma komisijas sēdē

_____ prot. Nr. _____

Komisijas sekretārs: _____