LATVIJAS UNIVERSITĀTE
DATORIKAS FAKULTĀTE

LINUX UN DOCKER KONTEINERI KĀ SISTĒMAS VIRTUALIZĀCIJAS RĪKI

KVALIFIKĀCIJAS DARBS

Darba autors: Ģirts Dālbergs
Studenta apliecības numurs: gd13020
Darba zinātniskais vadītājs: Dr. sc. comp. Leo Trukšāns
Lektors

RĪGA 2015

ANOTĀCIJA

Šobrīd arvien vairāk attīstās mākoņskaitļošana, un līdz ar to aug arī nepieciešamība pēc labām un efektīvām sistēmas virtualizācijas iespējām. Līdz šim pārsvarā tiek izmantota pilna sistēmas virtualizācija ar virtuālajām mašīnām, bet to realizācija prasa daudz laika un datora resursu.

Šī darba mērķis ir izpētīt konteinerus kā sistēmas virtualizācijas iespēju un apsvērt tos kā alternatīvu virtuālajām mašīnām, kā arī izpētīt un mēģināt ieviest kādu no konteineru realizācijas iespējām. Vēl vajadzētu mēģināt integrēt konteinerus mākonī.

Darba gaitā ir iepazīti Linux un mazāk Docker konteineri un to realizācijas iespējas, kā arī īstenota konteineru izveide un pārvaldīšana. Papildus tam ir izpētīta minimālu konteineru veidošana un pārnešana starp dažādām pamatsistēmām, un arī ar mākoni savietojamu konteineru realizācija izmantojot hipervizoru Libvirt.

ABSTRACT

Linux and Docker containers as system virtualization tools

At the moment cloud computing is growing by the second and because of that the need for good and efficient system virtualization options. For now mostly full virtualization is being used, but it consumes a lot of time and computing resources.

The goal of this paper is to research containers as a system virtualization tool and consider them as an alternative for virtual machines, as well as research and try to implement some of the container deployment options. Furthermore containers should be integrated in a cloud.

During the research the author has gotten to know about Linux and a little bit less about Docker containers and their implementation options, as well as created and managed containers. In addition to that, minimal container deployment and compatibility between multiple hosts, as well as cloud compatible containers using Libvirt have been implemented.

SATURS

APZĪMĒJUMU VĀRDNĪCA

IP - šo simbolu pāris ir saīsinājums terminam *Internet Protocol*, kas latviešu valodā nozīmē Interneta Protokols.

Wi-Fi - lokāls bezvadu tīkls.

Designate - tā ir *Openstack* mākoņa komponente DNS kā serviss. Šī komponente ir atbildīga par domēna vārdu un IP adrešu attiecību uzglabāšanu.

QEMU - ta sir vispārējs atvērtā koda mašīnu un aparatūras emulators.

Chroot - Tā ir operācija, kas maina šķietamo augstākā slāņa mapi uz citu. Tā liek datoram domāt, ka nomainītā mape ir augstākā līmeņa mape. Tas ir veids, kā izolēt lietojumprogrammas no pārējās sistēmas.

Sudo tiesības - Sudo ir programma, kas ļauj lietotājam izpildīt komandas ar cita lietotāja privilēģijām. Parasti tās ir superlietotāja root privilēģijas. Ar apzīmējumu sudo tiesības saprot superlietotāja tiesības.

Root lietotājs - kā root lietotāju saprot superlietotāju, kuram nav nekādu ierobežojumu. Kā root mapi saprot augstākā līmeņa mapi.

Apparmor - tā ir sistēma, kas ir kodola papildinājums procesiem pieejamo resursu ierobežošanai.

Ssh - Secure Shell ir tīkla protokols, kas nodrošina drošu datu apmaiņu izmantojot tīkla savienojumu starp divām ierīcēm.

Loopback - tas apzīmē signāla maršrutēšanu atpkaļ uz tā pirmavotu. Loopback interfeiss ir, interfeiss, kuru izmantojot visi dati tiek sūtīti atpakaļ uz avotu. To parasti izmanto tīkla sistēmas funkcionalitātes pārbaudīšanā.

Mb - ar šo simbolu pāri darbā tiek apzīmēti megabaiti.

XML - paplašināmā iezīmēšanas valoda. Ar vārdiem XML fails tiek saprasts fails, kura saturs ir rakstīts paplašināmā iezīmēšanas valodā.

IEVADS

Mūsdienās arvien pieprasītāka kļūst mākoņskaitļošana un pieaug pieprasījums pēc mākoņskaitļošanas pakalpojumiem. Arvien vairāk palielinās nepieciešamība pēc "izīrējamiem" datora resursiem un tos pārsvarā nosdrošina ar virtuālajām mašīnām mākonī. Lai to īstenotu lielākoties izmanto pilno virtualizāciju, kas emulē datora iekārtas un sniedz klientam iespēju izmantot datora resursus neklātienē, taču virtuālo mašīnu veidošana un uzturēšana patērē daudz laika un datorresursus, tāpēc arvien nopietnāk apskata konteinerus kā sistēmas virtualizācijas rīku, un daži uzņēmumi jau šo iespēju ir realizējuši, jo konteineri ir ar krietni mazākiem virstēriņiem gan atmiņas, gan citu datorresursu ziņā.

Kvalifikācijas darba pētāmā problēma ir efektīvu un viegli un ātri veidojamu virtuālo mašīnu trūkums mākonī. Šobrīd galvenais mākonī izmantotais virtuālo mašīnu veidošanas rīks ir *KVM* (Kernel-based Virtual Machine), kas katrai virtuālajai mašīnai emulē visas fiziskās iekārtas un tas aizņem laiku un atmiņu pamatsistemā, bet konteineru integrēšana ļautu samazināt gan patērēto laiku, gan nepieciešamo atmiņu.

Darba mērķis ir izpētīt konteineru darbības principus un to realizēšanas iespējas, un ieviest kādu no konteineru risinājumiem, kā arī mēģināt ieviest konteinerus mākonī kā alternatīvu *KVM* virtuālajām mašīnām, lai samazinātu klientam piedāvājamo virtuālo mašīnu izveides laiku un tām nepieciešamo atmiņu, kā arī uzlabotu to ātrdarbību. Papildus tam ir plānots izpētīt, kā radīt pēc iespējas mazākus konteinerus, to ātrākai darbībai un sagatavotu savu konteinera šablonu, ko piedāvāt klientiem.

Darbs sastāv no vairākām daļām, kurās apskata esošās struktūras stāvokli, darba problēmas un to risinājumus, risinājumu plānu un ieviešanu, konteinerus kā rīku, minimāla konteinera veidošanu, konteineru realizācijas iespējas, to pārvaldību un integrāciju mākonī, kā arī drošības apsvērumus.

Darba izstrādes laikā tika izmantota informācija, kura tika atrasta internetā un ir uzrādīta izmantotās literatūras sadaļā, kā arī darbs ticis noformēts pēc LU DF kvalifikācijas darba standartiem.

Darba gaitā ir izdevies izpētīt, izprast un ieviest *LXC* un zemā līmenī arī *Docker* konteinerus, kā arī ieviest *Libvirt-lxc* konteineru risinājumu *CentOS* operētājsistēmā. Ir izdevies arī izpētīt konteineru pārvaldību un migrēšanu no vienas vides uz citu, realizēt atmiņas ziņā mazāko iespējamo konteineri un beigās integrēt *Libvirt-lxc* konteinerus mākonī, taču nav izdevies mēģināt ieviest *LXD* konteinerus un sīkāk izpētīt *Docker* konteinerus.

1. ESOŠĀS STRUKTŪRAS APRAKSTS

Darbs tika veikts biroja telpās, kurās tika izmantoti divi stacionārie datori un viens personīgais klēpjdators. Abi stacionārie datori (1.1.att. un turpmāk Ubuntu pamatsistēma un CentOS pamatsistēma) bija pieslēgti darbinieku apakštīklam ar privātajām IP adresēm. No darbinieku apakštīkla ir piekļuve internetam, taču no interneta puses tiek uzturēti tikai jau izveidotie savienojami un no ārpuses visas iekārtas darbinieku tīklā ir nesasniedzamas. Bija pieejams tikai viens *Ethernet* tīkla kabelis un viens komplekts ar perifērijas ierīcēm, tādēļ šie abi datori reizē darboties nevarēja, bet tika darbināti pamīšus pēc vajadzības. Arī 1.1.att. šīs abas pamatsistēmas ir attēlotas tikai ar vienu savienojumu uz darbinieku apakštīkla komutatoru. Uz viena datora bija *Ubuntu 14.04 LTS Trusty Tahr* operētājsistēma un uz otra *CentOS 6.6 Server* operētājsistēma bez grafiskās vides. Augstāk minētajā darbinieku apakštīklā bija pieslēgtas arī citas nesaistītas iekārtas, kolēģu datori.

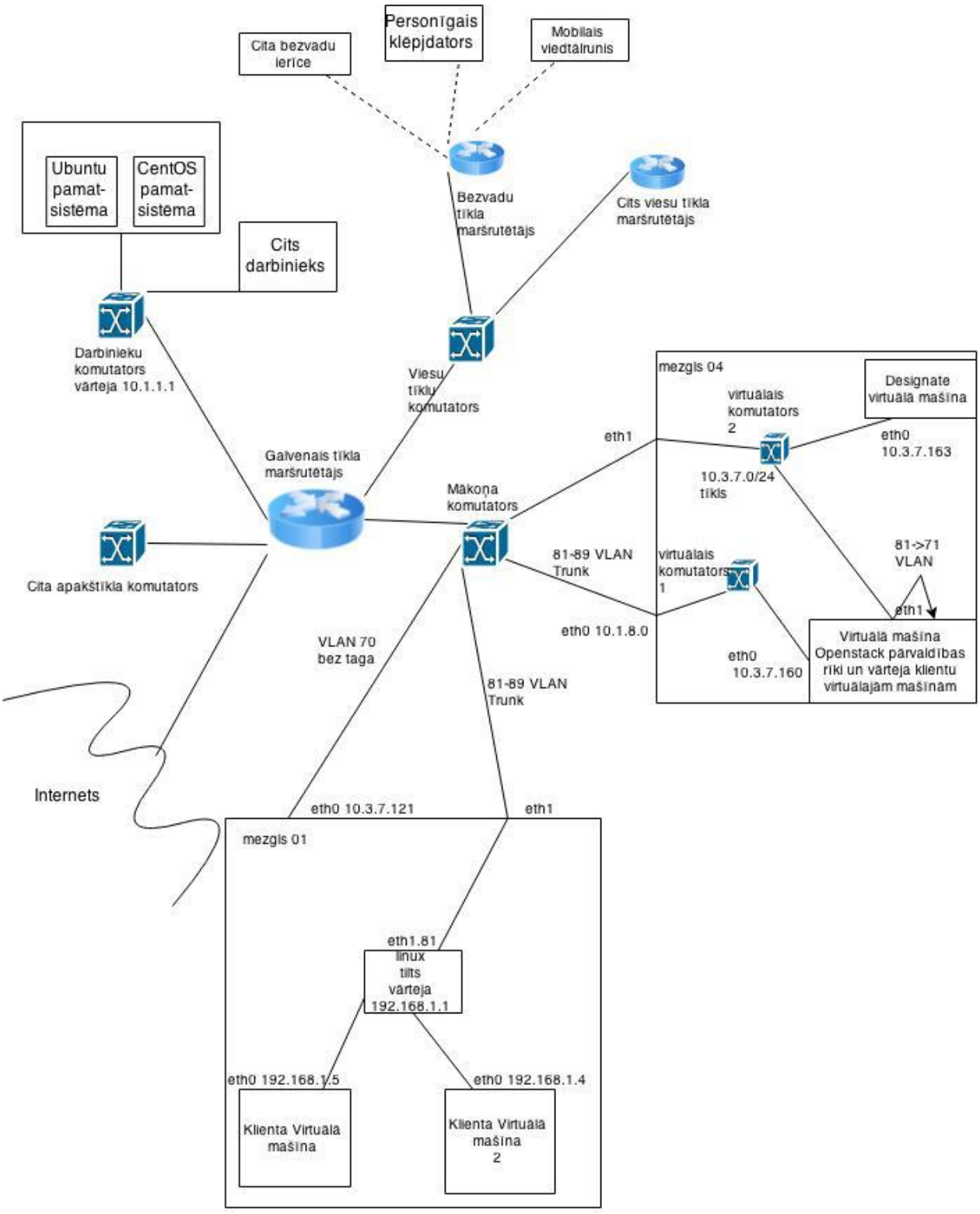
Darbinieku apakštīkla komutators bija pieslēgts galvenajam tīkla maršrutētājam, kuram bija tieša piekļuve internetam. Šim maršrutētājam bija arī citi apakštīkli un to skaitā ir viesu apakštīkls. Viesu apakštīklā bija pieslēgts komutators, kurš tālāk veidoja vēl citus zemāka līmeņa apakštīklus, no kuriem viens bija darbinieku telpas *Wi-Fi* apakštīkls, kurā bija pieslēgts personīgais klēpjdators. Šajā bezvadu apakštīklā arī tika izmantotas privātās IP adreses un bija tādi pat savienojumu ierobežojumi kā darbinieku apakštīklā, taču no neviena no publiskajiem apakštīkliem nebija piekļuve citiem apakštīkliem galvenajā tīklā.

Galvenajā tīklā bija pieslēgts arī mākoņa komutators, kurš veidoja mākoņa apakštīklu. *Openstack* mākoņa realizācijai tika izmantoti divi serveri, Mezgls01 un Mezgls04, kuri abi bija pieslēgti mākoņa apakštīkla komutatoram. Darba gaitā tika izmantota *Juno Openstack* versija.

Serveris Mezgls04 tika izmantots kā *Openstack* pārvaldības serveris, kurā bija uzstādītas divas virtuālās mašīnas. Pirmā virtuālā mašīna tika izmantota priekš *Designate* komponentes, un otra tika izmantota kā galvenais *Openstack* pārvaldības rīks, un tajā bija uzstādīti *Compute, Network, Image, Identity* un citi servisi, un vēl tā kalpoja kā noklusētā vārteja klientu virtuālajām mašīnām. Šīs otrās virtuālās mašīnas *eth0* tīkla interfeiss caur virtuālu komutātoru (1.1.att. virtuālais komutators 1) bija savienots ar servera *eth0* tīkla interfeisu, savukārt *eth1* tīkla interfeiss caur otru virtuālo komutatoru bija savienots ar servera *eth1* tīkla interfeisu un šis savienojums izmantoja 70. virtuālo lokālo tīklu. Šīs virtuālās mašīnas *eth1* tīkla interfeisasm bija arī cilpa pašam uz sevi, un šis savienojums izmantoja 71.-81. virtuālos lokālos tīklus. Papildus

tam ar *eth0* tīkla interfeisu pie šī otrā virtuālā komutatora bija pieslēgta arī servera pirmā virtuālā mašīna, uz kuras bija uzstādīta augstāk minētā *Designate* komponente.

Serveris Meszgls01 kalpoja kā vide, kurā uzstāda klientu virtuālās mašīnas un tajā bija uzstādīti *Nova-compute, Neutron-linux-bridge-agent* un *Ceilometer-agent-compute* servisi, kā arī virtuālo mašīnu pārvaldības rīks *KVM,* kurš šī darba ietvaros tika aizstāts ar konteineru realizācijas rīku. Veidojot klienta virtuālās mašīnas, tām tiek veidoti savi apakštīkli, kuri izmanto virtuālos lokālos tīklus. Tas tiek realizēts sekojoši: katram virtuālo mašīnu apakštīklam tiek piešķirts virtuālais lokālais tīkls un virtuālais tīkla interfeiss *eth1.x* (x ir virtuālā lokālā tīkla numurs), kurš tiek pakļauts servera *eth1* tīkla interfeisam. Šis virtuālais tīkla interfeiss tiek savienots ar, speciāli virtuālo mašīnu apakštīklam izveidotu, tiltu, kurš savukārt ir savienots ar visām šī apakštīkla virtuālajām mašīnām, un tām tiek piešķirtas privātās IP adreses. Šis tilts kalpo kā virtuālo mašīnu apakštīkla noklusētā vārteja. Jau augstāk minētais servera *eth1* tīkla interfeiss ar mākoņa komutatora starpniecību ir savienots ar otra servera Mezgls04 *eth0* tīkla interfeisu, un šis savienojums izmanto 81.-89. virtuālo lokālo tīklu maģistrāli. Savukārt šī servera *eth0* tīkla interfeiss caur mākoņa komutatoru ir savienots ar otra servera *eth1* tīkla interfeisu un papildus tam starp šo *eth0* tīkla interfeisu un mākoņa komutatoru tiek izmantots 70. virtuālais lokālais tīkls bez taga.



1.1.att. **Esošās struktūras topoloģija.** *Avots: Autora zīmējums programmā draw.io*

2. RISINĀJUMA PLĀNS

Lai sasniegtu darbam uzstādītos mērķus, tiek sekots risiājuma plānam, kurš ir nepieciešams strukturētas un mērķtiecīgas darbības realizēšanai. Kā pirmo soli būtu jāveic izpēte, kas vispār ir konteineri un kādi ir to realizācijas veidi. Darbu sākot tika ieteikts palasīt par *LXC* konteineriem tāpēc ar to arī jāsāk. Tiek plānots izpētīt *LXC* un *Docker* konteinerus, un kā tos realizēt. Nākamais solis ir mēģināt ieviest *LXC* konteinerus lokālā darbstacijā un izprast to darbības principus.

Pēc konteineru izpētes un ieviešanas ir plānots izpētīt un mēģināt izveidot konteineri, kurš prasa pēc iespējas mazāk atmiņu tā glabāšanai, jo konteinera apjoms ir viens no galvenajiem faktoriem, kas nosaka tā startēšanās ātrumu, un tas ir svarīgi, jo darba mērķis ir realizēt virtuālās mašīnas, kas strādā ātrāk kā pilnā sistēmas virtualizācija. To ir plānots panākt pētot, kā konteineri vispār tiek veidoti un mēģinot izmainīt to izveides šablonus, kā arī "apgraizot" jau gatavus izveidotus konteinerus. Lai zinātu, kuras pakotnes nav sistēmas pamatfunkciju veikšanai svarīgas, tiks izmantota programma *Debtree*, kas ļauj izprast pakotņu savstarpējās atkarības. Pēc tam būtu nepieciešams atrast alternatīvu konteineru izveidei, piemēram izmantojot *Debootstrap* rīku failsistēmas realizācijai, un pēc tam padarot šo failsistēmu par konteineri.

Kad ir pabeigta nepieciešamās atmiņas ziņā minimālā konteinera izveide, ir jāmēģina realizēt šie konteineri citā vidē, kas būs *CentOS* operētājsistēmas vidē, jo sākuma pamatsistēmas *Ubuntu* un *CentOS* sistēmas virtualizācijas ir visizplatītākās, no konteineriem virtualizēt iespējamajām operētajsistēmām. Konteinerus būtu vēlams realizēt to dabiskajā vidē, tāpēc ir plānots *CentOS* pamatsitēmā realizēt *CentOS* konteinerus un *Ubuntu* pamatsistēmā realizēt *Ubuntu* konteinerus. Tā kā otrai darbstacijai ir *CentOS 6.6 Server* operētājsistēma, kurai nav pieejama *LXC* pakotne, šajā vidē konteineri tiks realizēti izmantojot *Libvirt-lxc* konteineru realizācijas rīku. Pēc konteineru realizācijas abās šajās vidēs ir plānots mēģināt migrēt izveidotos konteinerus no vienas vides uz otru, jo, lai sasniegtu darbā izvirzīto mērķi, ieviest konteinerus mākonī, ir nepieciešams, lai konteinerus būtu iespējams migrēt uz mākoni.

Pēc tam tiek plānots izpētīt konteineru drošības faktorus un mēģināt ieviest iespējamo darbību ziņā limitētu konteineri, pēc kā sekos konteineru integrācija mākonī.

3. ALTERNATĪVO RISINĀJUMU APSKATS

Par konteineru realizācijas rīku tiek lietots *LXC* un *Libvirt-lxc* un visas uz mērķi vērstās darbības tiek veiktas ar šiem konteineriem, jo tie atšķirībā no *Docker* konteineriem ir paredzēti tieši veselas sistēmas virtualizācijai. Docker konteineri drīzāk ir paredzēti atsevišķu procesu nodalīšanai un izolēšanai, nekā pilnas sistēmas virtualizēšanai un izolēšanai.

Konteinerus ir iespējams realizēt arī ar *Solaris Containers, FreeBSD jail, HP-UX Containers* un citām programmām, taču tās ir mazāk izplatītas un ar grūtāk pieejamām dokumentācijām un palīdzības forumiem, kā arī svarīgs faktors ir tas, ka mākonī integrējamie konteineri ir tieši *Libvirt-lxc.* Arī *Docker* konteinerus var ieviest mākonī, taču par galveno rīku ir izvēlēts *LXC.*

4. KAS IR KONTEINERI

Konteineru virzīta sistēmas virtualizācija ir virtualizācijas paveids, kas ļauj visiem pamatsistēmas viesiem izmantot vienu (pamatsistēmas) kodolu. Izmantojot konteinerus, virtualizācijas slānis strādā kā lietojumprogramma pamatsistēmā, un katrs pamatsistēmas viesis tiek saukts par konteineru. Konteineru virtualizācija piedāvā arī resursu sadalījumu, līdzīgi kā strādājot ar pilnās vitrualizācijas virtuālajām mašīnām, kas ļauj izolēt konteinerus vienu no otra, kā arī nodrošināt, ka viens konteineris neizmanto visus pieejamos pamatsistēmas resursus. Konteineru galvenais pluss ir tas, ka tiem ir mazi virstēriņi, salīdzinot ar pilno virtualizāciju, gan atmiņas, gan ātrdarbības ziņā, jo izmantojot konteineru virtualizāciju nav jāveic sistēmas kodola un iekārtu emulācija, un konteineri izmanto pamatsistēmas sistēmas izsaukumus. Tomēr šis virtualizācijas veids nav tikai ar plusiem, bet tam ir arī mīnusi, un galvenais mīnuss ir tas, ka konteineru virtualizācija spēj nodrošināt virtualizāciju tikai vienas operētājsistēmas ietvaros, kas neļauj uz linux sistēmām virtualizēt *Windows* operētājsistēmu, jo tai ir cits kodols, taču spēj nodrošināt visu *Linux* distributīvu virtualizāciju, piemēram, *Ubuntu, Debian, CentOS, CirrOS, OpenSUSE* un citas.[1]

Konteineri var tikt izmantoti arī priekš parastu procesu izolēšanas un darbināšanas, kas paver lieliskas iespējas programmatūras testēšanā, neuztraucoties par sekām uz pamatsistēmu, un tādu konteineru piemērs ir arī tālāk darbā apskatītie *Docker* konteineri. Tas nodrošina arī pilnas sistēmas virtualizāciju, taču parasti tiek izmantots individuāla procesa izolēšanai.

5. KVM VIRTUĀLĀS MAŠĪNAS

Ir svarīgi runāt arī par pilnās virtualizācijas virtuālajām mašīnām, jo to mērķis ir tāds pat kā konteineriem un tieši KVM virtuālās mašīnas tiks aizstātas ieviešot LXC konteinerus mākonī. KVM *(Kernel-based Virtual Machine)* ir uz kodolu bāzēts pilnās virtualizācijas risinājums paredzēts izmantošanai ar *Linux* pie *x86* aparatūras un satur gan *Intel VT*, gan *AMD-V* virtualizācijas paplašinājumus. Tas sastāv no ielādējama kodola moduļa *kvm.ko*, kas nodrošina pamata virtualizācijas infrastruktūru un īpašu kodola modeli *kvm-intel.ko* priekš *Intel VT* un *kvm-amd.ko* priekš *AMD-V* virtualizācijas paplašinājuma. Tam ir nepieciešams pārveidots *QEMU*, kas ir virtuālo mašīnu emulators, bet izstrādātāji strādā pie šo izmaiņu ieviešanas jau uzstādāmajā pakotnē. Izmantojot *KVM* var darbināt vairākas virtuālās mašīnas, izmantojot neizmainītus *Linux* un *Windows* attēlus.[8]

6. DOCKER KONTEINERI

6.1 Docker konteineru pamatprincipi

Docker ir atvērta platforma izstrādātājiem un administratoriem, kas ļauj veidot, nodrošināt un darbināt lietojumprogrammas. Tas nodrošina lietojumprogrammu ātru uzstādīšanu no komponentēm. Tā konteineri ir pilnībā pārnesami starp ierīcēm un operētājsistēmām, kurām ir *Docker*. Ar to ir viegli sākt strādāt, jo *Docker* centrmezglā ir pieejamas vairāk kā 13000 gatavas lietojumprogrammas. Tas ir paredzēts galvenokārt lietojumprogrammu konteineru veidošanai, un ļauj izolēt atsevišķas programmas vai procesus, taču iespējams izmantot arī pilnas sistēmas virtualizācijai, saglabājot iespēju sadalīt pieejamos resursus un pasargāt pamatsistēmu.

6.2 Docker konteineru realizācija

Docker konteinerus var realizēt tikai vienā veidā, uzstādot *Docker* pakotni. Tā tika uzstādīta *Ubuntu* pamatsistēmas vidē un vienkāršā līmenī tika pārbaudīta tā darbība. Pirms pakotnes uzstādīšanas ir nepieciešams atjaunināt sistēmu

```
# apt-get update
```

Pēc sistēmas atjaunināšanas var instalēt *Docker*

```
# apt-get -y install docker.io
```

Arguments '*-y*' nodrošina automātisku pakotnes instalēšanas apstiprināšanu. Pēc tam ir jāiespējo automātiska teksta pabeigšana *Docker* komandām.

```
# source /etc/bash_completion.d/docker.io
```

Tā arī ir visa nepieciešamā vides sagatavošana un pēc šīs darbības veikšanas var sākt veidot *Docker* konteinerus. Kā uzstādāms konteineris tika izvēlēts konteineris, kurš saturēs *Ubuntu* sistēmu.

```
# docker run -I -t ubuntu /bin/bash
```

Ja pamatsistēmā *Docker* vēl nav strādājis ar šo konteineri, tad tam nebūs pieejams attēls šim konteinerim, tādēļ tas atvelk repozitoriju 'ubuntu', un no tā lejupielādē 5 attēlus. Konteineris darbojas normāli, un nav nekādu problēmu. Pēc pāris vienkāršu darbību veikšanas tika secināts, ka konteinerim viss ir kārtībā un sistēma tikusi virtualizēta līdzīgi kā izmantojot *LXC*. Pēc konteinera darbības pārbaudes tā vidi var pamest.

```
exit
```

Pēc konteinera izveides un *Docker* darbības principu izpratnes tika apskatīta informācija par pašu *Docker*:

```
# docker info
```

```
Containers: 1
Images: 5
Storage Driver: aufs
 Root Dir: /var/lib/docker/aufs
 Dirs: 7
Execution Driver: native-0.2
Kernel Version: 3.13.0-45-generic
WARNING: No swap limit support
```

No šīs komandas izvada var saprast, ka šobrīd ir pieejami piece iepriekš lejupielādētie attēli konteineru veidošanai, un viens konteineris ir izveidots. Tiek parādīta informācija, kur glabājas konteineri, dzinis un tā versija, kā arī kodola versija.

Pēc darba ar *Docker* konteineriem var apturēt visus esošos konteinerus un tos likvidēt

```
# docker stop $(docker ps -a -q)
# docker rm $(docker ps -a -q)
```

Darbs pie *Docker* konteineriem bija krietni īsāks kā pie *LXC* konteineriem, jo bija priekšstats, ka *LXC* konteineri ir labāki pilnas sistēmas virtualizācijai, kā arī, jo tika atrasta informācija, ka *Docker* konteineri ir paredzēti vairāk procesu izolēšanai, kā arī svarīgs faktors bija tas, ka *LXC* piedāvā vieglāk saprotamu un ērtāku lietotāja darbības telpu un saskarni. [25][26]

7. LINUX KONTEINERI

7.1 LXC pamatprincpi

Linux konteineri ir sistēmas virtualizācijas risinājums, kas ir vairak līdzīgs pilnveidotai *Chroot* videi, kā kādam no pilnās virtualizācijas risinājumiem, piemēram *KVM*, tādēļ, ka nav jāemulē apparatūra un tie izmanto tādu pat operētājsistēmu kā pamatsistēma. *LXC* konteineriem ir pieejami divi 'režīmi'. Var veidot priviliģētus un nepriviliģētus konteinerus. Priviliģētus konteinerus veido, ja ir pieejamas *sudo* tiesības, un tiem ir mazāka drošība, savukārt nepriviliģētiem konteineriem ir lietotāju un grupu identifikatoru kartēšanas iespēja. Tas nozīmē, ka, piemēram, konteinera *root* lietotājs, kuram vienmēr ir 0. identitāte, pamatsistēmā tiks uztverts kā 100000. identitātes lietotājs. Tas pasargā pamatsistēmu dažādos gadījumos, piemēram, ja konteinera *root* lietotājs iemanās pamest konteinera failsistēmu, jo bez identifikatoru kartēšanas, šis konteinera lietotājs, pametot konteinera failsistēmu, varētu veikt jebkuru darbību.

LXC konteinerus var ieviest divos veidos, kuri abi izmanto vienādus kodola līdzekļus. Pirmā no realizācijas iespējām ir *Libvirt*, kas ļauj izmantot konteinerus, pieslēdzoties *LXC* dzinim. Tā izmantošana ir tik pat vienkārša kā citu *Libvirt* dziņu. Otra konteineru implementācija vienkārši tiek saukta par *LXC,* un tā nav savietojama ar *Libvirt*, bet ir elastīgāka, un piedāvā vairāk rīkus lietotājam, kas atvieglo konteineru pārvaldīšanu. Ir iespējams arī pārslēgties no viena uz otru, taču tas var radīt problēmas un apjukumu. Lokālām vajadzībām parasti izmanto *LXC*, jo to ir ērtāk lietot un *Libvirt* ir nedrošāks, jo tam pietrūkst *Apparmor* aizsardzība.

7.2 LXC uzstādīšana Ubuntu vidē

Lai uzstādītu *LXC*, ir nepieciešams lietotājs ar *sudo* tiesībām. Pirms pakotnes uzstādīšanas ir nepieciešams atjaunināt sistēmā jau esošās pakotnes, un tikai tad jāuzstāda *LXC* pakotne.

Komandas pakotnes instalēšanai:

```
# apt get update
# apt get install lxc
```

Tālāk ar „*y*" ir jāapstiprina pakotnes instalācija. Pēc apstiprināšanas tiks instalēta *LXC* pakotne, kā arī visas pakotnes, kuras tai ir nepieciešamas.

7.3 LXC konteineri Ubuntu vidē

Pēc *LXC* uzstādīšanas var sākt veidot konteinerus. Tiek izveidots pirmais konteineris, izmantojot iebūvēto lejupielādes šablonu, kuram var norādīt, kādu sistēmu instalēt, vai izdarīt izvēli, atbildot uz jautājumiem par sistēmas specifikācijām.

Pirmā konteinera izveide, izmantojot lejupielādes šablonu:

```
# lxc-create –t download –n mysql -- -d ubuntu –r trusty –a amd64
```

Argumentu skaidrojums:

- *-t* ir arguments, kas nosaka, kādu šablonu izmantot instalējot
  Vērtība ‘download’ apzīmē lejupielādes šablonu
- *-n* arguments apzīmē veidojamā konteinera vārdu, ar kuru tam varēs piekļūt
- -- nozīmē, ka tālāk sekos argumenti specifiski lejupielādes šablonam
- *-d* arguments apzīmē uzstādāmā konteinera distributīvu
- *-r* arguments apzīmē uzstādāmās distributīvas laidienu
- *-a* arguments norāda, kādu arhitektūru izmantos uzstādāmais konteineris

Pēc konteinera izveides, mapē *‘/var/lib/lxc’* tiek izveidota mape ar tādu pat nosaukumu kā konteinerim, un tajā atrodas konteinera konfigurācijas fails, kā arī mape ‘*rootfs’*, kurā glabājas konteinera failsistēma.

Drošības apsvērumu dēļ jaunajam konteinerim pēc uzstādīšanas nav neviena lietotāja konta un, lai tādu izveidotu, iesaka izmantot *lxc-attach* komandu vai *chroot* iespēju, kas ļauj pieslēgties konteinera failsistēmai. Aktivizēt *root* lietotāju var arī vienkārši piešķirot tam paroli, un to var izdarīt arī nepislēdzoties failsistēmai, vienkārši ‘*shadow’* failā nomainot *root* lietotāja paroli uz sev zināmas paroles jaucējkodu. *Ubuntu* pamatsistemai *root* lietotājam parole ir ‘root’, un konteinerim var uzstādīt tādu pat paroli, pārkopējot paroles jaucējkodu no pamatsistemas ‘*shadow’* faila uz konteinera ‘*shadow’* failu.

Atrodam paroles jaucējkodu pamatsistēmā:

```
# nano /etc/shadow
```

Faila pirmā rindiņa satur informāciju par *root* lietotāju un ir sekojoša:

```
root:$6$zC7cP/Tr$2DXGJqOMr2fyiZ2dbuYW6s0DlpToTystQDpZNJ3F1UF4Gg4G6hS
hDP0c7XzePq2T18hqc9Jz/4ATdHToxC/sY1:16483:0:99999:7:::
```

Viss starp pirmo un otro rindiņas kolu ir *root* lietotāja paroles jaucējkods. Pārkopējot šo jaucējkodu uz konteinera ‘*shadow’* failu, uzstādam *root* lietotājam tādu pat paroli kā pamatsistēmā un iespējojam šo lietotāju. Konteinera *‘shadow’* failā šī jaucējkoda vietā ir izsaukuma zīme, kas nozīmē, ka lietotājs ir atspējots.

Tālāk ir jāatver konteinera *‘shadow’* fails, un tajā jāiekopē jaucējkods starp pirmo un otro kolu izsaukuma zīmes vietā.

```
# nano /var/lib/lxc/mysql/rootfs/etc/shadow
```

Iekopē jaucējkodu.

```
Ctrl+o
Enter
Ctrl+x
```

Pēc izmaiņu veikšanas jānospiež taustiņu kombinācija '*ctrl + o*', kas saglabā veiktās izmaiņas un jāasptiprina darbība ar 'Enter' taustiņa nospiešanu. Pēc tam aizveram *Nano* komandrindas teksta redaktoru ar taustiņu kombināciju '*ctrl + x*'.

Kad ir iespējots *root* lietotājs, var startēt konteineri:

```
# lxc-start –n mysql –daemon
```

Argumenta *–n* nozīme ir norādīt izvēlētā konteinera nosaukumu un arguments *–daemon*, liek konteinerim startēties fonā un automātiski neslēgties klāt konteinera komandrindai. Tas ir nepieciešams, jo *LXC* pakotnē vēlprojām ir nenovērstas kļūdas, un, ja, slēdzot iekšā automātiski pieslēdzas komandrindai, tad nestrādā komandrindas izbēgšanas taustiņu kombinācija.

Tālāk var paskatīties, kādi ir šobrīd uzstādītie konteineri:

```
# lxc-ls –fancy
```

--fancy arguments uzrāda pilnīgāku skatu.

```
NAME   STATE    IPV4        IPV6  AUTOSTART
----------------------------------------
mysql  RUNNING  10.0.3.237  -     NO
```

Var redzēt, ka šobrīd ir uzstādīts viens konteineris ar vārdu 'mysql' un tas šobrīd ir ieslēgts, un tam ir piešķirta privātā *LXC* tīkla IP adrese 10.0.3.237.[9]

Tālāk var pieslēgties konteinera komandrindai un strādāt konteinerī:

```
# lxc-console –n mysql
```

Jāievada konteinera lietotājs un tā parole, un tiek nodrošināta pieeja konteinera komandrindai. Tajā var strādāt tā pat, kā tad, ja konteineris būtu pilnīgi patstāvīga sistēma, bet darbības tajā neietekmēs pamatsistēmu.[10] Veicam dažas darbības, lai pārliecinātos, ka viss strādā kā nākas.

```
# apt-get install nano
# useradd –m –G sudo user
```

Jāuzstāda *Nano* teksta redaktoru konteinerī, un pēc tam jāizveido jaunu sistēmas lietotāja kontu. Arguments *–m* nozīmē, ka jaunajam lietotājam būs mājas mape, un arguments *–G* nozīmē,

ka šis lietotājs tiks pievienots grupai, kuru norādam ar vērtību ‘sudo’, tādējādi pievienojot lietotāju *sudo* lietotāju grupai, un piešķirot tam *sudo* tiesības.

```
# passwd user
```

Lietotājam ir jāpiešķir parole, lai tas tiktu iespējots un varētu pieslēgties sistēmai. Pēc šīs komandas divas reizes ir jāievada jaunā parole un tā tiks nomainīta.

```
# apt-get install openssh-server
```

Uzstādam *ssh* serveri, lai konteinerim varētu pieslēgties ar *ssh* savienojumu. To varēs izdarīt tikai no pamatsistēmas, jo konteineris atrodas *LXC* iekšējajā tīklā, un, lai pieslēgtos no cita datora, ir nepieciešams piešķirt konteinerim publisko IP adresi. Arī pēc šīs komandas ir nepieciešams apstiprināt darbību ar ‘*y*’ simbolu, kā pēc visām pakotņu instalācijas komandām.

Pēc dažu darbību veikšanas konteinerī un tā darbības pārbaudes var iziet no konteinera komandrindas nospiežot taustiņu kombināciju ‘*ctrl+a’* un uzreiz pēc tam taustiņu ‘*q*’.

Analogi pirmajam mysql konteinerim tika izveidots vēl viens tāds pat konteineris, tikai ar nosaukumu ‘apache’.

7.4 LXC tīkla topoloģija

Lai izprastu konteineru pārvaldnieka *LXC* tīkla topoloģiju, ir nepieciešams apskatīt izvadus sekojošām komandām:

```
# brctl show
```

```
bridge name     bridge id       STP enabled     interfaces
docker0         8000.56847afe9799    no
lxcbr0          8000.fe6f59aec594    no         vethGYIHCA
                                                vethOWBBGB
```

Pēc šīs komandas izraksta var redzēt, ka sistēmā ir izveidoti divi tilta savienojumi ‘docker0’ un ‘lxcbr0’, un var redzēt arī visas ierīces, kuras ir pieslēgtas šiem tiltiem. ‘docker0’ tiltam nav pieslēgta neviena ierīce, jo visi *Dcoker* konteineri tika likvidēti, taču *LXC* izveidotajam ‘lxcbr0’ tiltam ir pieslēgtas divas ierīces, kas ir ‘mysql’ un ‘apache’ konteineri. Pie šo konteineru informācijas izvadiem zemāk var redzēt, ka to saites ir tieši tādas, kā šajā izrakstā redzamie interfeisi.

```
# ifconfig
```

```
docker0   Link encap:Ethernet  HWaddr 56:84:7a:fe:97:99
          inet addr:172.17.42.1  Bcast:0.0.0.0  Mask:255.255.0.0
          inet6 addr: fe80::5484:7aff:fefe:9799/64 Scope:Link
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:8 errors:0 dropped:0 overruns:0 frame:0
```

```
          TX packets:36 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:536 (536.0 B)  TX bytes:5749 (5.7 KB)

eth0      Link encap:Ethernet  HWaddr 4c:72:b9:97:ef:44
          inet addr:10.1.1.39  Bcast:10.1.1.255  Mask:255.255.255.0
          inet6 addr: fe80::4e72:b9ff:fe97:ef44/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:168837 errors:0 dropped:0 overruns:0 frame:0
          TX packets:78244 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:195174774 (195.1 MB)  TX bytes:10242774 (10.2 MB)
          Interrupt:20 Memory:f7e00000-f7e20000

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:7709 errors:0 dropped:0 overruns:0 frame:0
          TX packets:7709 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:1221058 (1.2 MB)  TX bytes:1221058 (1.2 MB)

lxcbr0    Link encap:Ethernet  HWaddr fe:6f:59:ae:c5:94
          inet addr:10.0.3.1  Bcast:10.0.3.255  Mask:255.255.255.0
          inet6 addr: fe80::94a4:c8ff:fec2:f80f/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:6182 errors:0 dropped:0 overruns:0 frame:0
          TX packets:6857 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:413172 (413.1 KB)  TX bytes:16816615 (16.8 MB)

vethGYIHCA Link encap:Ethernet  HWaddr fe:6f:59:ae:c5:94
          inet6 addr: fe80::fc6f:59ff:feae:c594/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:2732 errors:0 dropped:0 overruns:0 frame:0
          TX packets:2934 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:213854 (213.8 KB)  TX bytes:7521386 (7.5 MB)

vethOWBBGB Link encap:Ethernet  HWaddr fe:ed:3c:16:2c:27
          inet6 addr: fe80::fced:3cff:fe16:2c27/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:3450 errors:0 dropped:0 overruns:0 frame:0
          TX packets:4052 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:285866 (285.8 KB)  TX bytes:9308500 (9.3 MB)
```

Pēc šīs komandas izraksta var redzēt visus interfeisus sistemā un informāciju par tiem. Var redzēt, ka ir *Docker* un *LXC* izveidotie tilti, kas veido savus privāto adrešu apakštīklus, *eth0* interfeiss, *lo* interfeiss, kas ir *loopback* interfeiss un abu konteineru saites uz 'lxcbr0' tiltu.

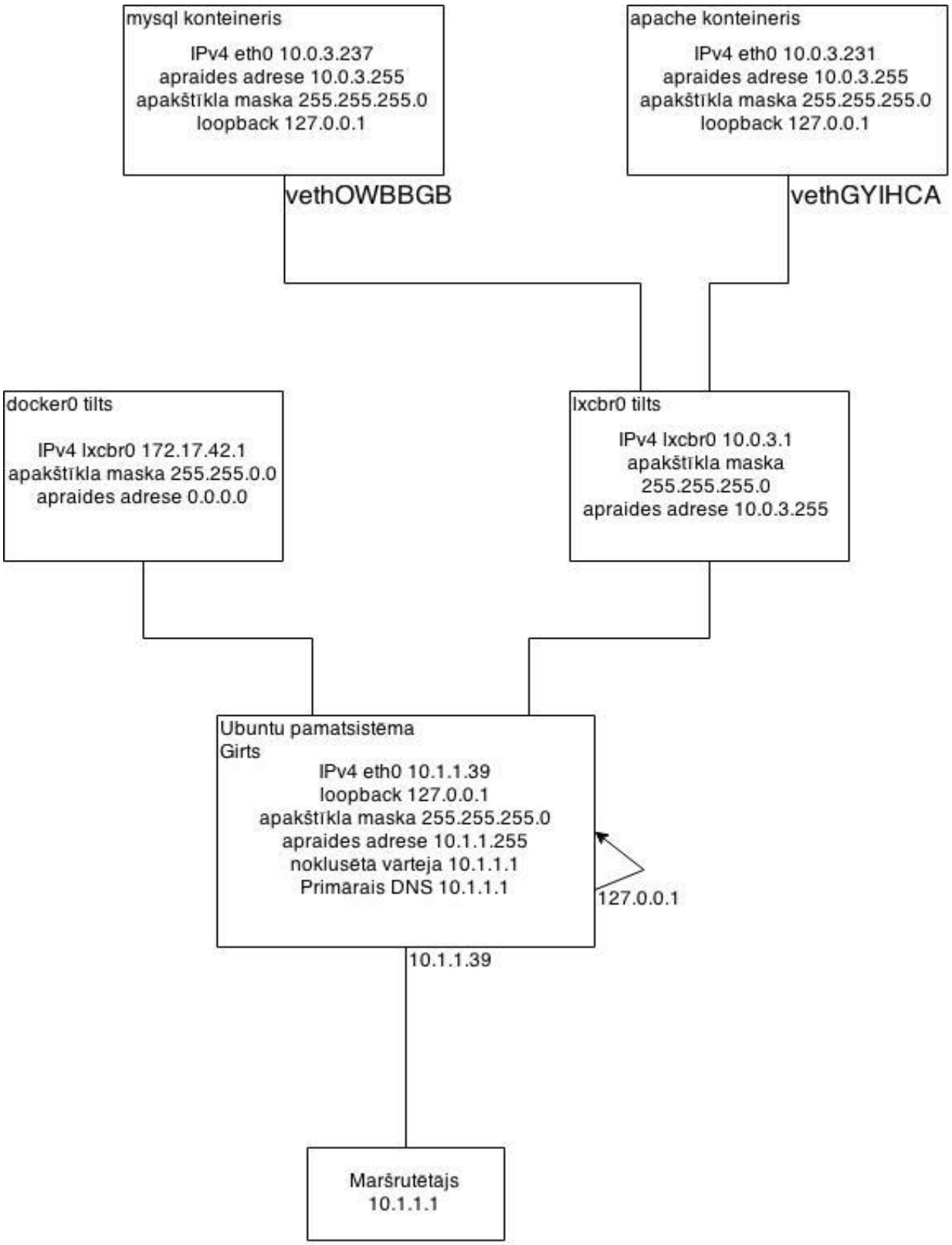
```
# lxc-info -n apache
```

```
Name:           apache
State:          RUNNING
PID:            4353
IP:             10.0.3.231
CPU use:        54.51 seconds
BlkIO use:      129.65 MiB
Memory use:     94.58 MiB
KMem use:       0 bytes
Link:           vethGYIHCA
 TX bytes:      208.84 KiB
 RX bytes:      7.17 MiB
 Total bytes:   7.38 MiB
```

```
# lxc-info -n mysql
```

```
Name:           mysql
State:          RUNNING
PID:            2934
IP:             10.0.3.237
CPU use:        51.53 seconds
BlkIO use:      257.71 MiB
Memory use:     274.27 MiB
KMem use:       0 bytes
Link:           vethOWBBGB
 TX bytes:      279.17 KiB
 RX bytes:      8.88 MiB
 Total bytes:   9.15 MiB
```

Abu konteineru informācijas izvados ir redzamas to IP adreses, kā arī saites uz 'lxcbr0' tiltu. No šo komandu izvadiem var izprast, visu tīkla topoloģiju. Pamatsistēmā ir tilta interfeiss, kuram tiek pieslēgti visi *LXC* konteineri. Tālāk ir bilde ar tīkla topoloģiju, kurā ir attēlota šobrīdējā situācija. Pie *Docker* tilta nav neviena pieslēgta ierīce, jo beidzot darbu ar *Docker* visi konteineri tika likvidēti.



7.1.att. **LXC tīkla topoloģija** *Avots: Autora zīmējums programmā draw.io*

7.5 LXC konteinera sasniegšana no tīmekļa

Strādājot ar priviliģētajiem konteineriem kā tas ir darīts līdz šim, tiem var piekļūt tikai no pamatsistēmas, jo visi konteineri ir pieslēgti *LXC* iekšējā tīklā, un tiem ir piešķirtas privātās IP adreses. No tīmekļa šie konteineri ir nepieejami, taču to var labot dažādos veidos. Viens no tiem ir izveidot jaunu tilta savienojumu, kas aizstās *LXC* veidoto, tā, ka šis *LXC* apakštīkls ir pievienots lokālajam tīklam un visiem konteineriem adreses piešķir tīkla maršrutētājs. Šis veids tiek realizēts pēcāk darbā, taču ir arī cits veids, kas ir drošāks. Var izveidot ugunsmūra likumu, kas ļaus piekļūt specifiskām konteinera pieslēgvietām. Piemēram, konteinerī var uzstādīt tīmekļa serveri, kuram noklusējuma pieslēgvieta ir 80. Pamatsistēma izveido ugunsmūra likumu, kas nosaka, ka, ja slēdzas klāt pamatsistēmas 8080. pieslēgvietai, tad pieprasījums tiek pārsūtīts uz konteinera 80. pieslēgvietu. Tādā veidā tiek nodrošināta piekļuve konteinera tīmekļa serverim, neliedzot piekļuvi pamatsistēmas tīmekļa serverim.

Šādu likumu var izveidot vienkārši ar vienu komandu komandrindā:

```
# iptables –t nat –A PREROUTING –I eth0 –p tcp –dport 8080 –j NAT –to
10.0.3.231:80
```

Tiek izveidots likums, kas pārsūta visas paketes, kas tiek sūtītas uz pamatsistēmas *eth0* interfeisa 8080. pieslēgvietu, uz konteinera ar IPv4 adresi 10.0.3.231 80. pieslēgvietu.[11]

7.6 Minimāla konteinera veidošana

Pēc pāris vienkāršu konteineru izveides, tika izveidots parasts konteineris, izmantojot piedāvāto ‘ubuntu’ šablonu, un netika veiktas nekādas papildus darbības ar šo konteineri.

```
# lxc-create –t ubuntu –n ubuntu
```

Pēc izveides atkal tika iespējots ‘*root*’ lietotājs, iekopējot ‘*shadow*’ failā sev zināmas paroles jaucējkodu. Tālāk vajadzeja noskaidrot, cik daudz atmiņas šis konteineris aizņem pamatsistēmā:

```
# cd /var/lib/lxc/ubuntu
du –hs rootfs
```

```
369M rootfs
```

Šī konteinera failsistēma aizņem 369 Mb atmiņas pamatsistēmā, kas ir par daudz.

Nākamais solis ir veidot pašam savu konteinera sistēmu un neizmantot iebūvētos konteineru šablonus. Ir pāris veidi, kā var veidot pats savu konteineri, piemēram esošo šablonu

pārveidošana, gatava konteinera ‘apgraizīšana’ un failsistēmas instalācija un pārveidošana par konteineri. Kā pirmais veids tika izmēģināts šablonu mainīšanas un veidošanas paņēmiens.

7.6.1 Konteineru veidošanas šabloni

Uzstādot *LXC*, automātiski tika noglabāti arī konteineru veidošanas skripti jeb šabloni. Tie atrodas *‘/usr/share/lxc/templates’* mapē un tādi ir 17. Pēc pamatinformācijas apgūšanas tika atrasts avots, kurā ir iszskaidrots, kā jāveic sava skripta izveide, un kas ir svarīgs veidojot savu konteinera izveides skriptu. Avotā bija arī pievienots fails ar izveides skriptu, kuru arī tika mēģināts ieviest. Skatīt pirmo pielikumu.[12] Lai šis skripts būtu lietojams, to ir jāpārceļ uz mapi, kur glabājas *LXC* šabloni un jānoņem faila nosaukuma paplašinājums ‘.sh’, kā arī jānomaina tiesības, lai tās sakrīt ar pārējiem šabloniem.

```
# cp –i /home/girts/Downloads/lxc-minimal.sh /usr/share/lxc/templates/
# mv /usr/share/lxc/templates/lxc-minimal.sh
/usr/share/lxc/templates/lxc-minimal
# chmod 755 /usr/share/lxc/templates/lxc-minimal
```

Pēc šablona skripta sagatavošanas var mēģināt taisīt konteineri, izmantojot šo šablonu.

```
# lxc-create –t lxc-minimal –n minimals
```

Šai darbībai tika izvadīts kļūdas paziņojums, ka šis šablons ir nederīgs. Mēģināju arī norādīt pilnu ceļu uz šablona atrašanās vietu, taču tas arī nelīdzēja.

```
# lxc-create –t /usr/share/lxc/templates/lxc-minimal –n minimals
```

Pēc šī mēģinājuma tika meklēti citi avoti par to, kā veidot savus šablonus vai mainīt esošos. Nākamais avots veidoja, šablonu no gatava konteinera, kas neder minimāla konteinera veidošanā un šis šablons nemaz nebija konteinera izveides skripts, bet cita veida šablons, un šai darbībai var vienkārši izmanto klonēšanas komandu. [13] Pēc ilgas pētīšanas tika atrasti tikai avoti, kā veidot šablonu no gatavas sistēmas, bet ne konteinera izveides skriptu, un netika atrasts neviens avots, kas paskaidro, kā veidot šo skriptu pašam, izņemot jau augstāk minēto gatavo skriptu, kurš nenostrādāja. Arī oficiālajā dokumentācija ir teikts, ka var izmantot piedāvātos noklusējuma šablonus un viss darbosies labi, taču nav teikt kā veidot savu šablonu.

Pēc secinājuma, ka tāda darbība nav aprakstīta, tika atrasts avots, ka minimālu sistēmu var viegli izveidot, lietojot gatavo šablonu, bet tā izveides komandai pievienojot argumentu ‘*--trim*’. Izmantojot šo argumentu, konteinera izveides skripts neuzstāda pakotnes, bez kurām var iztikt un neuzstāda dažus servisus, tādējādi samazinot konteinera izmērus.[14] Pēc šī argumenta atrašanas tas tika izmēģināts:

```
# lxc-create –t ubuntu –n minimal –trim
```

Šī komanda atgrieza izvadu, ka tāda opcija nav atpazīstama, un šķita, ka problēma ir vecā *LXC* pakotnes laidienā, tādēļ tika nolemts, ka vajag noskaidrot vai nav pieejams jaunāks *LXC* laidiens, kā uz *Ubuntu* pamatsistēmas uzstādītais.

Vispirms noskaidroju uzstādītās pakotnes laideinu:

```
# dpkg –s lxc | grep ‘Version’
Version: 1.0.7-0ubuntu0.1
```

Tika noskaidrots, ka uz doto brīdi bija tikai viena jaunāka pakotnes versija, taču pēc pāris dienām tika izlaista vēl jaunāka versija.[15] Tika nolemts, ka jāmēģina uzstādīt jaunākā versija, taču oficiālajos repozitorijos jaunākā pieejamā versija bija jau uzstādītā.

Pēc secinājuma, ka jaunāka versija nav pieejama, tika meklēta papildus informāciju par ‘*--trim*’ argumentu, un tika atrasta publikācija no 2013. gada, kurā viens no galvenajiem *LXC* izstrādātājiem, uzsver, ka šis arguments ir bīstams, un tādēļ šo iespēju turpmākās pakotnes versijas vairs neatbalstīs, tātad arī šī iespēja nav izdevusies.[16]

Beidzot darbu ar šabloniem tika atrasts vēl viens failsistēmas izveides skripts, kas neveido konteineri, bet gan tikai pašu failsistēmu, un tas tika izmēģināts. Skripts ir otrajā pielikumā.[17] Skripts izsaucam no komandrindas un tam jāpadod arguments ar ceļu uz mapi, kurā uzstādīt failsistēmu, un tas izmantojot *debootstrap* programmatūru, uzstādīs failsistēmu.

```
# apt-get install debootstrap
# cd /var/lib/lxc
# mkdir tryout
# cd tryout
# mkdir rootfs
# nano little.sh
```

Jaunizveidotajā failā ‘little.sh’ jāiekopē atrastais skripts, un tad fails jāsaglabā. Pēc tam jānomaina failam tiesības un var izpildīt failu.

```
# ctrl+o
# enter
# ctrl+x
# chmod 755 little.sh
# ./little.sh /var/lib/lxc/tryout/rootfs
```

```
du –hs rootfs
```

```
369M rootfs
```

Skripts nostrādā un mapē rootfs ir uzinstalēta *Ubuntu* failsistēma izmērā 369 Mb, kas ir tikpat, cik no šablona veidotajam konteinerim, tātad no šī skripta jēga nav. Pēc šīs darbības šabloni un izveides skripti tiek nolikti malā, un jāmēģina 'apgraizīt' esošs konteineris.

7.6.2 Esoša konteinera samazināšana

Nākamais veids, kā var veidot minimālu konteineri ir apskatīt jau gatavu uzstādītu konteineri, un likvidēt tajā esošas pakotnes, kuras nav obligāti nepieciešamas sistēmas normālai funkcionalitātei. Lai veiktu nevajadzīgo pakotņu likvidēšanu, ir nepieciešams apskatīt visas pakotnes, kuras ir uzstādītas izvēlētajā konteinerī, un izpētīt, kāds ir katras pakotnes mērķis un kādas ir tās funkcijas, lai pieņemtu lēmumu, vai bez šīs pakotnes var iztikt. Šī konteinera 'apgraizīšana' tika veikta ar konteineri 'ubuntu'.

Pirmais solis ir pieslēgties konteinerim un paskatīties visas uzstādītās pakotnes tajā.

```
# lxc-start –n ubuntu –daemon
# lxc-console –n ubuntu
```

Konteineris ir iedarbināts un ievadīta pieslēgšanās komanda, pēc kuras ir jāievada lietotāja vārds un parole. Tiek izmantots iespējotais *root* lietotājs. Pēc pieslēgšanās konteinerim ir jāapskatās visas sistēmā esošās pakotnes.

```
# dpkg –l
```

Šīs komandas izvads ir redzams 3. pielikumā, un tajā ir redzamas visas sistēmā esošās pakotnes un informācija par tām tabulas formā. Pirmā kolonna ir divi simboli, kas apzīmē pakotnes stāvokli un vērtība '*ii*' nozīmē, ka pakotne ir uzstādīta un nav likvidēta. Otrā kolonna apzīmē pakotnes vārdu, trešā arhitektūru, kādai pakotne ir paredzēta un ceturtā pakotnes īsu aprakstu. Tālāk ir jāizpēta katras pakotnes nozīme, lai varētu pieņemt lēmumu, vai bez tās var iztikt, un tas ir izdarīts. Pakotņu aprakstus var redzēt 4. pielikumā. Izpētot katras pakotnes nozīmi, tika pieņemts lēmums, ka var mēģināt likvidēt sekojošas pakotnes: *debconf-i18n, eject, gcc-4.8-base:amd64, gcc-4.9-base:amd64, gpgv, iputils-ping, kbd, less, lockfile-progs, makedev, mawk, module-init-tools, netcat-openbsd, ntpdate, openssh-sftp-server, plymouth, ssh, tzdata, vim, vim-common, vim-runtime, vim-tiny*. Pakotnes dzēst labāk ir ar *purge* opciju, kas likvidēs arī visus šīs pakotnes konfigurācijas failus, neatstājot nekādu lieku informāciju:

```
# apt-get purge debconf-i18n
```

Pēc šīs komandes ievades tiek izvadīts paziņojums, ka pēc šīs komandas darbības apstiprinājuma tiks likvidētas divas pakotnes: ‘*debconf-i18n’* un ‘u*buntu-minimal’. Ubuntu-minimal* ir metapakotne, kura nodrošina minimālas *Ubuntu* sistēmas instalāciju, un ir atkarīga no visām minimālās sistēmas pakotnēm, un, šo pakotni likvidējot, var rasties problēmas ar sistēmas atjaunināšanu, tāpēc šī pakotne ir jāatstāj, kas noved pie secinājuma, ka visas atrastās pakotnes, kuras varētu likvidēt, nemaz likvidēt nedrīkst. Pēc šī secinājuma tika nonākts pie slēguma, ka ir jāuztaisa pakotņu atkarības grafs, lai redzētu, kuras pakotnes ir likvidējamas. To var viegli un ērti izdarīt ar *Debtree* programmu, kura ir atsevišķi jāuzstāda:

```
# apt-get install debtree
```

Jāapstirpina pakotnes instalācija, un programma tiek uzstādīta. Izveidoju pakotņu atkarības grafu pakotnei ‘*dpkg*’, lai izprastu programmas darbību.

```
cd /home/girts/Desktop
# debtree dpkg > out_dpkg.dot
dot –T png –o out_dpkg.png out_dpkg.dot
```

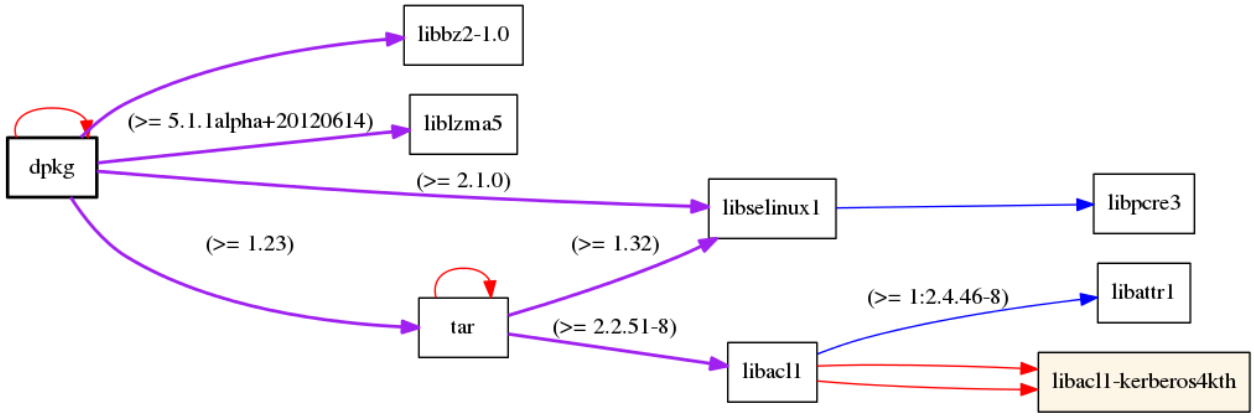
Uz darba virsmas tiek izveidots fails ‘out_dpkg.dot’, kurā ir uzskaitītas visas atkarības šai pakotnei, un ar nākamo komandu no šī faila tiek izveidota bilde ar atkarību grafu, ar nosaukumu ‘out_dpkg.png’. Bildes formātu norāda ar argumentu ‘-T’.

/home/girts/Desktop/out_dpkg.dot

```
digraph "dpkg" {
    rankdir=LR;
    node [shape=box];
    "dpkg" -> "libbz2-1.0" [color=purple,style=bold];
    "dpkg" -> "liblzma5" [color=purple,style=bold,label="(>=
5.1.1alpha+20120614)"];
    "dpkg" -> "libselinux1" [color=purple,style=bold,label="(>=
2.1.0)"];
    "libselinux1" -> "libpcre3" [color=blue];
    "dpkg" -> "tar" [color=purple,style=bold,label="(>= 1.23)"];
    "tar" -> "libacl1" [color=purple,style=bold,label="(>= 2.2.51-
8)"];
    "libacl1" -> "libattr1" [color=blue,label="(>= 1:2.4.46-8)"];
    "libacl1" -> "libacl1-kerberos4kth" [color=red];
    "libacl1" -> "libacl1-kerberos4kth" [color=red];
    "tar" -> "libselinux1" [color=purple,style=bold,label="(>=
1.32)"];
    "tar" -> "tar" [color=red];
    "dpkg" -> "dpkg" [color=red];
    "dpkg" [style="setlinewidth(2)"]
```

```
    "libacl1-kerberos4kth" [style=filled,fillcolor=oldlace];
}
// Excluded dependencies:
// libc6 multiarch-support zlib1g
// total size of all shown packages: 8426496
// download size of all shown packages: 2493454
```

/home/girts/Desktop/out_dpkg.png



7.2.att. **Pakotņu atkarības grafs pakotnei dpkg** *Avots:veidots ar programmu Debtree*

Pēc šīs bildes izveides ir izprasta *Debtree* darbība, un var ķerties pie '*ubuntu-minimal*' pakotnes atkarību grafa veidošanas. Lai to īstenotu, ir nepieciešams uz izvēlētā konteinera uzstādīt *Debtree* programmu un taisīt atkarību grafu konteinera '*ubuntu-minimal*' pakotnei.

```
# lxc-start -n ubuntu -daemon
# lxc-concole -n ubuntu
```

Jāautorizējas sistēmā ar iespējoto *root* lietotāju.

```
# apt-get install debtree
# cd /
# debtree ubuntu-minimal --no-skip --show-all --no-recommends --show-
installed > ubuntu_minimal_kont.dot
# dot -T png -o ubuntu_minimal_kont.png ubuntu_minimal_kont.dot
```

Ir uzstādīta '*debtree*' pakotne un izveidota bilde ar '*ubuntu-minimal*' pakotnes atkarību grafu konteinera *root* mapē. Bilde ir nesamērojami liela, 11.7 Mb ar izmēru 10923 x 9525 pikseļi, jo šajā grafā ir ļoti daudz pakotnes, tāpēc uzskatāmībai šī bilde ir pievienota piektajā pielikumā ar 8% samazinājumu. Arguments '*--no-skip*' liek uzrādīt arī pēc noklusējuma apspieztās pakotnes, '*--show-all*' liek ģenerēt pilnu atkarības koku, '*--no-recommends*' liek nerādīt ieteiktās pakotnes, '*--show-installed*' liek rādīt pakotnes, kas ir uzinstalētas sistēmā. Papildus tam esot konteinerī izveidoju failu 'pakotnes', kurā ierakstu visas konteinerī esošās pakotnes

```
# nano pakotnes
```

Saglabāju failu un izeju no redaktora.

```
ctrl+o
enter
ctrl+x
# dpkg –l >>pakotnes
```

Ierakstu jaunajā failā ‘pakotnes’ visas sistēmas uzstādītās pakotnes. Tālāk var pamest konteinera vidi un atgriezties pie pamatsistēmas komandrindas, ievadot taustiņu kombināciju ‘*ctrl+a*’ un pēc tam nospiežot taustiņu ‘*q*’. Pamatsistēmas vidē jāpārkopē ‘.dot’ un ‘.png’ faili, kā arī fails ‘pakotnes’, no konteinera failsistēmas uz pamatsistēmas darba virsmu ērtākai lietošanai un jānomaina failiem tiesības.

```
# cp –i /var/lib/lxc/ubuntu/rootfs/ubuntu_minimal_kont.png
/home/girts/Desktop/
# cp –i /var/lib/lxc/ubuntu/rootfs/ubuntu_minimal_kont.dot
/home/girts/Desktop/
# cp –i /var/lib/lxc/ubuntu/rootfs/pakotnes /home/girts/Desktop/
# cd /home/girts/Desktop
# chmod 664 ubuntu_minimal_kont.dot
# chmod 664 ubuntu_minimal_kont.png
# chmod 664 pakotnes
```

Tālāk, lai atrastu visas pakotnes, kas neietilpst šajā atkarību grafā, no šī ‘.dot’ faila jaunā failā tiek ierakstītas tikai pašas atkarības un to fails tiek nosaukts par ‘check1’. Pēc tam no šī faila jaunā failā ‘grep1’ tiek ierakstīti tikai paku nosaukumi no ‘check1’ faila.

```
# grep –o –P ‘(?<=”)[a-z0-9\-\.\+]*(?=”)’ check1 | sort | uniq >>grep1
```

Failam ‘pakotnes’ jāpiešķir īpašas tiesības aktīvajam sistēmas lietotājam:

```
# setfacl –m u:girts:rwx pakotnes
```

Pēc tam no šī faila ir jāiegūst tikai paku nosaukumi un jāieraksta jaunā failā ‘pakugrep’:

```
# cut –b 5-36 –n pakotnes >>pakugrep
# sed –i “s/ //g” pakugrep
# sed –i “s/:amd64//g” pakugrep | sort | uniq
```

Tālāk ir jāmeklē pakotņu nosaukumi, kuri ir failā ‘pakotnes’, bet nav failā ‘grep1’.

```
# comm pakugrep grep1
```

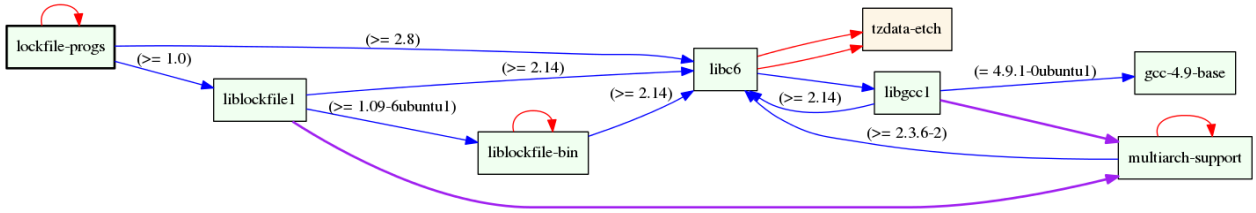
Šī komanda salīdzina abu padoto failu katru rindiņu un izvada 3 kolonnas, no kurām pirmajā ir pirmā faila unikālās rindiņas, otrajā otrā faila unikālās un trešajā kopējās. Tās pakotnes, kuras ir pirmajā kolonnā ir tās, kuras ir uzstādītas konteinerī, bet neietilpst ‘*ubuntu-minimal*’ atkarību grafā. Tālāk jāatrod šajā kolonnā tās pakotnes, kuras tika atrastas kā likvidējamas, un tādas ir tikai 4: *lockfile-progs, openssh-sftp-server, ssh, vim.* Tā kā lielākajai daļai pakotņu ir saistība ar citām pakotnēm, likvidējot šīs 4 pakotnes var palikt pāri citas nederīgas, kuras bija nepieciešamas tikai šīm, tāpēc tālāk ir jāizveido atkarību grafi katrai no šīm 4 pakotnēm un jāpārbauda, vai nav vēl kāda lieka pakotne. Šos grafus var veidot arī pamatsistēmā, jo šīs pakotnes sakrīt ar konteinera pakotnēm.

```
# debtree lockfile-progs --no-skip --show-all --no-recommends -show-
installed >> lockfile_progs.dot
# dot –T png –o lockfile_progs.png lockfile_progs.dot
# debtree openssh-sftp-server --no-skip --show-all --no-recommends -
show-installed >> openssh_sftp_server.dot
# dot –T png –o openssh_sftp_server.png openssh_sftp_server.dot
# debtree ssh --no-skip --show-all --no-recommends -show-installed >>
ssh.dot
# dot –T png –o ssh.png ssh.dot
# debtree vim --no-skip --show-all --no-recommends -show-installed >>
vim.dot
# dot –T png –o vim.png vim.dot
```

Pakotnes ‘*vim*’ atkarību grafā var redzēt, ka tam ir piesaistīta arī ‘*vim-tiny*’ pakotne, tāpēc arī tai jāizveido atkarību grafs, un varbūt arī to var likvidēt.

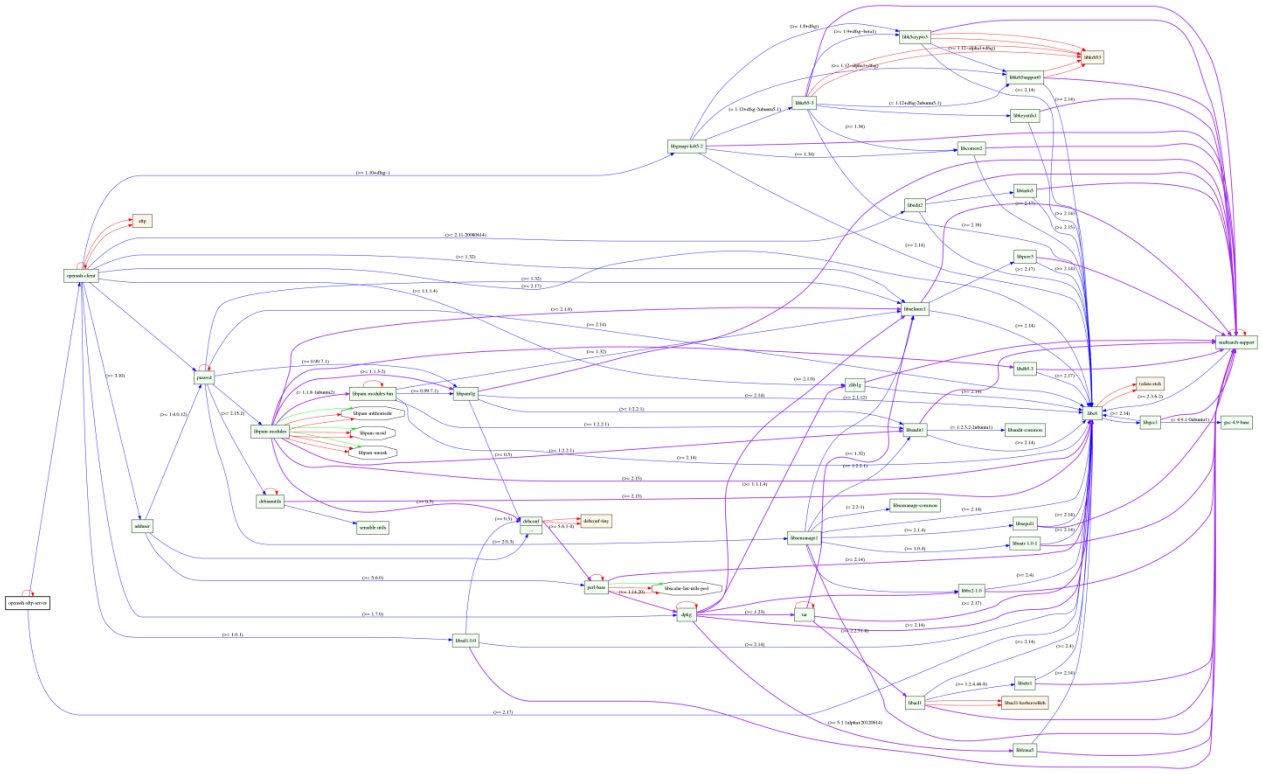
```
# debtree vim-tiny --no-skip --show-all --no-recommends -show-installed
>> vim_tiny.dot
# dot –T png –o vim_tiny.png vim_tiny.dot
```

/home/girts/Desktop/lockfile_progs.png



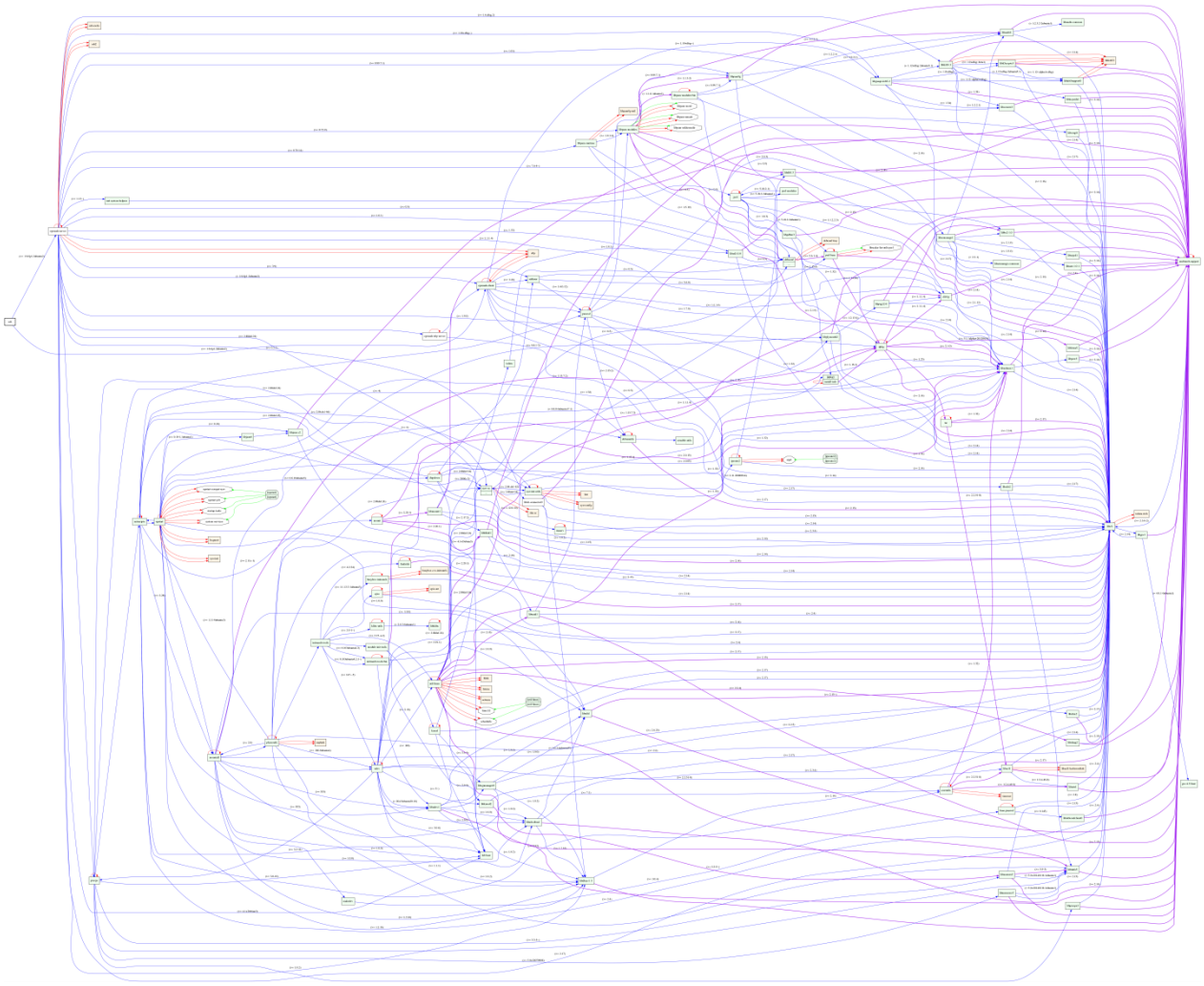
7.3.att. **Pakotņu atkarības grafs pakotnei lockfile-progs** *Avots:veidots ar programmu Debtree*

/home/girts/Desktop/openssh_sftp_server.png



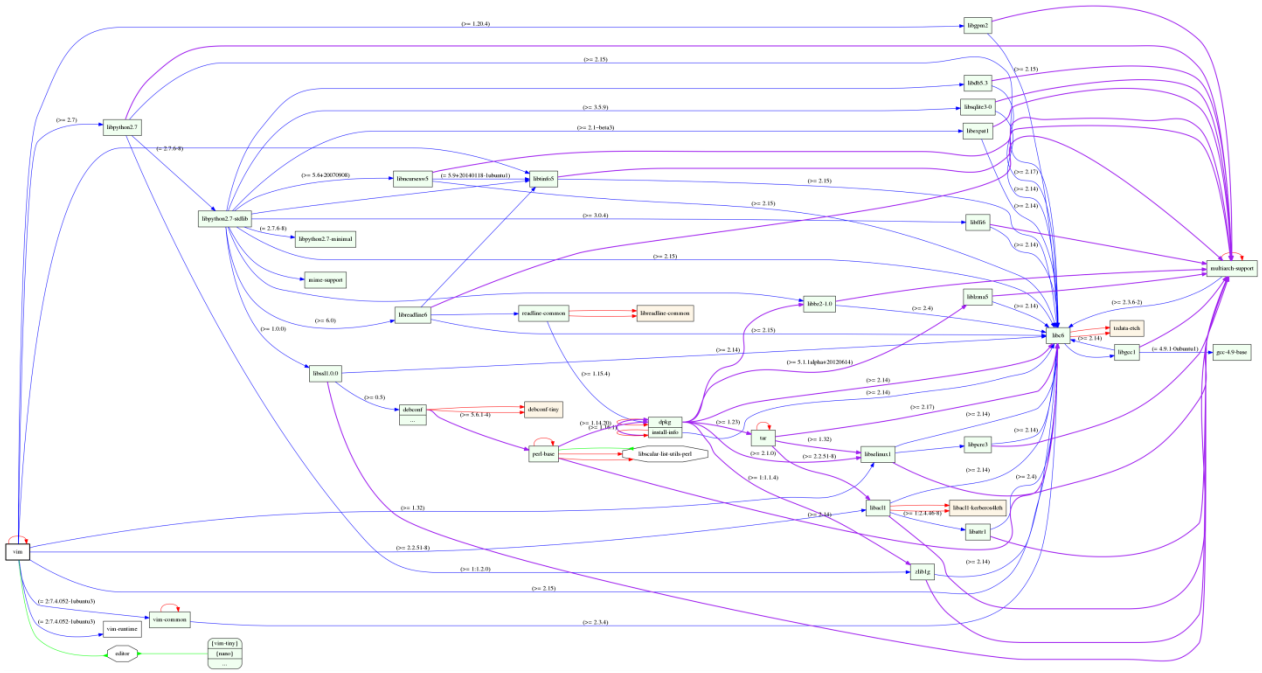
7.4.att. **Pakotņu atkarības grafs pakotnei openssh-sftp-server** *Avots:veidots ar programmu Debtree*

/home/girts/Desktop/ssh.png



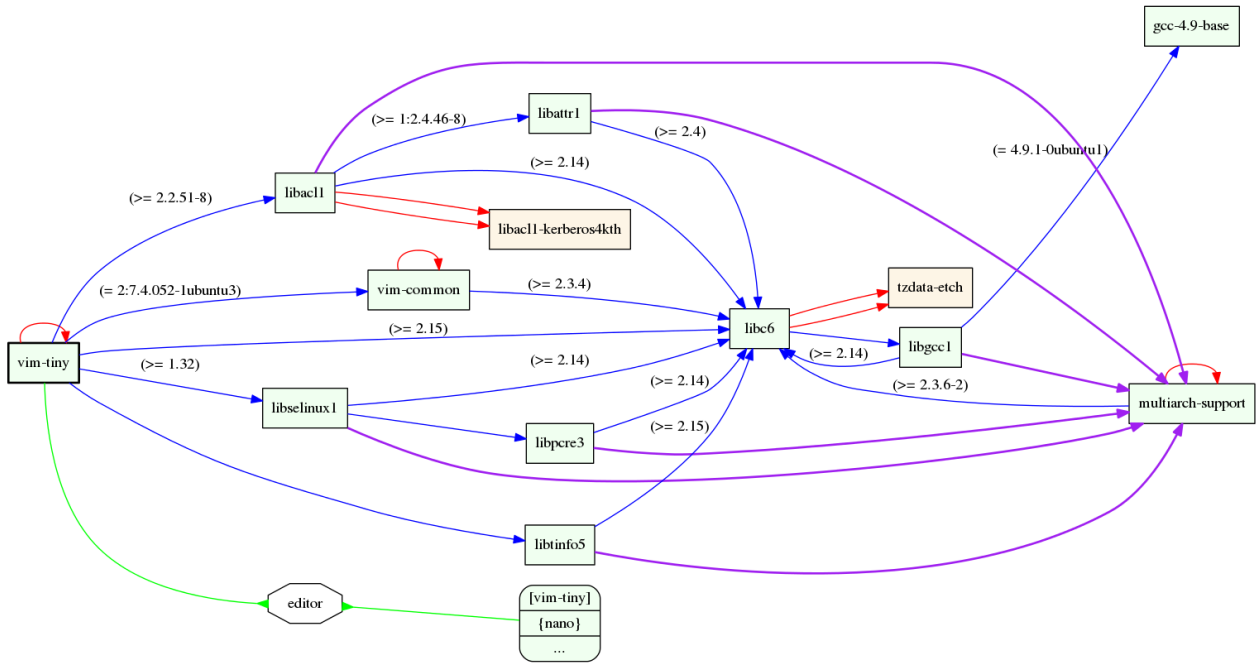
7.5.att. **Pakotņu atkarības grafs pakotnei ssh** *Avots:veidots ar programmu Debtree*

/home/girts/Desktop/vim.png



7.6.att. **Pakotņu atkarības grafs pakotnei vim** *Avots:veidots ar programmu Debtree*

/home/girts/Desktop/vim_tiny.png



7.7.att. **Pakotņu atkarības grafs pakotnei vim-tiny** *Avots:veidots ar programmu Debtree*

Pēc šim bildēm var secināt, kuras pakotnes ir jālikvidē. Kopā var likvidēt sekojošas pakotnes: *lockfile-progs, liblockfile1, liblockfile-bin, openssh-sftp-server, ssh, vim, vim-runtime.*

Kad ir atrastas visas likvidējamās pakotnes, jāizveido jauns ubuntu konteineris, kuram likvidēt atrastās pakotnes.

```
# lxc-create –t ubuntu –n mazais
```

Atkal jāpamaina '*shadow*' fails, lai iespējotu *root* lietotāju un varētu pieslēgties konteinerim.

```
# lxc-start –n mazais –daemon
# lxc-console –n mazais
```

Jāievada *root* lietotāja vārdu un paroli, un pirms izmaiņu veikšanas jāatjaunina sistēma:

```
# apt-get update
# apt-get purge lockfile-progs
# apt-get purge liblockfile1
# apt-get purge liblockfile-bin
# apt-get purge openssh-sftp-server
# apt-get purge ssh
# apt-get purge vim
# apt-get purge vim-runtime
```

Pēc pakotņu likvidācijas vēlreiz jāatjaunina sistēma, un pēc tam jānodzēš visas lejupielādētās pakotņu instalācijas, lai atbrīvotu atmiņu.

```
# apt-get update
# apt-get clean
```

Tālāk var pamest konteinera vidi un pārbaudīt tā apjomu:

```
ctrl+a
q
# du –hs /var/lib/lxc/mazais/rootfs
300M mazais
```

Konteinera izmērs ir sarucis līdz 300 Mb, un, pēc šo darbību veikšanas, darbs pie gatavu konteineru 'apgraizīšanas' var beigties.[18]

7.6.3 Sistēmas instalācija un pārveidošana par konteineri

Trešais veids, kā veidot minimālu konteineri, ir veidot pilnvērtīgu sistēmu pamatsistēmas mapē. Pastāv rīks *Debian Installer*, kas nodrošina sistēmas uzstādīšanu. Tas ir paredzēts, lai uzstādītu uz 'tukša' datora pilnvērtīgu sistēmu vai veiktu sistēmas pārinstalāciju, ja tā ir nobrukusi. Tā darbības laikā viens no posmiem ir pamata sistēmas pakotņu instalācija un to nodrošina iekļauts rīks *Debootstrap*, kas ir *Bootstrap* versija *Debian* sistēmām. *Debootstrap* ir rīks, kas nodrošina *Debian* sistēmas instalāciju citas pilnvērtīgas sistēmas mapē, neizmantojot datu krātuves. Tam ir nepieciešama tikai tīkla un *Debian* repozitorija piekļuve.[19][20]

Vispirms ir nepieciešams uzstādīt pašu *Debootstrap* pakotni:

```
# apt-get install debootstrap
```

Pēc pakotnes instalācijas var veidot pilnvērtīgu sistēmu norādot instalējamo distributīvu, mērķa mapi, kā arī var norādīt papildus argumentu '*--exclude*', kas ļauj norādīt pakotnes, kuras nevajag iekļaut instalācijā. Pirms vajag tikai izveidot mapi, kurā instalēt sistēmu.

```
# mkdir /var/lib/lxc/bootstrapped
# mkdir /var/lib/lxc/bootstrapped/rootfs
```

Tālāk var veidot sistēmu norādot, kuras pakotnes neinstalēt.

```
# debootstrap --exclude=ubuntu-minimal,less,iputils-ping,eject,debconf-
i18n,mawk,netcat-openbsd,ntpdate,vim-tiny,whiptail,libgnutls-
openss127,libgnutls126,libbsd0,vim-common,libtext-wrapi18n-
perl,libtext-iconv-perl,libtext-charwidth-perl,libnewt0.52,libpopt0
trusty /var/lib/lxc/bootstrapped/rootfs/
```

Šī komanda izveidos failsistēmu, neiekļaujot norādītās pakotnes, '/var/lib/lxc/bootstrapped/rootfs/' mapē.[22] *Debootstrap* automātiski iekļaus visas pakotnes, kuras ir nepieciešamas pamatsistēmas instalācijai, un iekļaus arī pakotnes, no kurām kāda cita instalējama pakotne ir atkarīga, pat, ja būs norādīts, ka to nevajag iekļaut instalācijā, tāpēc tika norādīts, lai neiekļauj pilnu pakotnes koku, citādāk to norādīto pakotni tik un tā uzstādīs. Pēc šīs komandas tika uzstādīta sistēma, neiekļaujot norādītās pakotnes, izņemot '*libgnutls126*' un '*libpopt0*'. Pēc failsistēmas izveides tai ir nepieciešams pieslēgties ar *chroot* komandu, un veikt pāris darbības. Jāveic sistēmas pakotņu atjaunināšana, nevajadzīgo instalāciju failu likvidēšana, sistēmas atjaunināšana, *Nano* teksta redaktora uzstādīšana. Pēc teksta redaktora pakotnes uzstādīšanas komandas tika izvadīti vairāki kļūdu paziņojumi, ka sistēmā ir trūkumi, tāpēc izmantos citas iespējas, piemēram, paziņo, ka nav neviena dialoga priekša, tāpēc izmanto

Readline priekšu. Šīs kļūdas netraucē sistēmas darbībai, bet tās parādās, jo ir likvidētas pakotnes, kas ir bijušas nepieciešamas šīs darbības veikšanai. Kā jau tika minēts, bez šīm pakotnēm var iztikt, bet var nākties samierināties ar dažiem kļūdu paziņojumiem.

```
# chroot /var/lib/lxc/bootstrapped/rootfs
# apt-get update
# apt-get upgrade
# apt-get install nano
# apt-get clean
```

Pēc šo darbību veikšanas var pamest *chroot* vidi un veikt pēdējos soļus failsistēmas sagatavošanā.

```
exit
```

Pēc vides pamešanas bija nepieciešams montēt dažas pamatsistēmas mapes jaunajā failsistēmā, lai nodrošinātu pareizu sistēmas darbību un neizvadītu kļūdu paziņojumus.

```
# mount –o bind /proc /var/lib/lxc/bootstrapped/rootfs/proc
# mount –o bind /sys /var/lib/lxc/bootstrapped/rootfs/sys
# mount –o bind /dev /var/lib/lxc/bootstrapped/rootfs/dev
# mount –o bind /dev/pts /var/lib/lxc/bootstrapped/rootfs/dev/pts
```

Failsistēma ir sagatavota [21] un tālāk var pārbaudīt izveidotās failsistēmas izmēru:

```
# du –hs /var/lib/lxc/bootstrapped/rootfs
207M rootfs
```

Pēc failsistēmas izveides jau pareizajā *LXC* konteineru mapē šo failsistēmu ir nepieciešams padarīt par *LXC* saprotamu konteineri. To var izdarīt vienkārši izveidojot konteinera konfigurācijas failu mapē ‘/var/lib/lxc/bootstrapped’ un fails ir jānosauc par ‘config’. Vajag izveidot arī tukšu failu ‘fstab’.

```
# nano /var/lib/lxc/bootstrapped/config
ctrl+o
enter
ctrl+x
# nano /var/lib/lxc/bootstrapped/fstab
ctrl+o
enter
```

```
ctrl+x
```

Šī faila saturu veidoji analogi citu konteineru konfigurācijas failiem, un tam ir jābūt sekojošam:

/var/lib/lxc/bootstrapped/config

```
# Common configuration
lxc.include = /usr/share/lxc/config/ubuntu.common.conf

# Container specific configuration
lxc.rootfs = /var/lib/lxc/bootstrapped/rootfs
lxc.mount = /var/lib/lxc/bootstrapped/fstab
lxc.utsname = bootstrapped
lxc.arch = amd64

# Network configuration
lxc.network.type = veth
lxc.network.flags = up
lxc.network.link = lxcbr0
lxc.network.hwaddr = 00:16:3e:72:14:da
```

Konteinera *MAC* adresi izdomāju savu. Tālāk ir nepieciešams sakonfigurēt tīkla interfeisus šim konteinerim, un to var izdarīt pārveidojot konteinera '/etc/network/interfaces' failu par sekojošu:

/var/lib/lxc/bootstrapped/rootfs/etc/network/interfaces

```
# interfaces(5) file used by ifup(8) and ifdown(8)
# Include files from /etc/network/interfaces.d:
source-directory /etc/network/interfaces.d

# The loopback network interface
auto lo
iface lo inet loopback

auto eth0
iface eth0 inet dhcp
```

Kad tas ir izdarīts, arī šim konteinerim ir jāiespējo *root* lietotājs, pamainot tā '*shadow*' failu, un jāpārbauda vai šī failsistēma no *LXC* puses tiek uztverta kā konteineris:

```
# lxc-ls –fancy
```

```
NAME          STATE    IPV4        IPV6  AUTOSTART
-------------------------------------------------
bootstrapped  STOPPED  -           -     NO
apache        STOPPED  -           -     NO
mysql         STOPPED  -           -     NO
```

```
mazais          STOPPED  -            -      NO
ubuntu          STOPPED  -            -      NO
```

Var redzēt, ka *LXC* ir atradis šo failsistēmu, un saprot to kā konteineri ar vārdu ‘bootstrapped’, jo tā sauc mapi, kurā glabājas konteinera failsistēma un konfigurācijas fails.

Pēc konteinera izveides jāpārbauda, cik atmiņas aizņem ši konteineris:

```
# du –hs /var/lib/lxc/bootstrapped/rootfs
207M rootfs
```

Ir izveidots konteineris, kura izmērs ir sarucis līdz 207 Mb, kas ir gandrīz divas reizes mazāk, kā pirmajam konteinerim, kas tika izveidots. Šis var tikt uzskatīts par minimālu *Ubuntu* sistēmas konteineri, ja tas darbojas.

Tālāk ir nepieciešams iedarbināt šo konteineri un pārbaudīt tā darbību:

```
# lxc-start –n bootstrapped –daemon
# lxc-console –n bootstrapped
```

Jāauterizējas kā *root* lietotājam, un pēc tam jāpārbauda konteinera darbība, atjauninot sistēmas pakotnes.

```
# apt-get update
```

Pēc šīs darbības viss darbojas kā nākas un darbs ar *LXC* konteineriem *Ubuntu* pamatsistēmā ir pabeigts. Atliek vienīgi uzstādīt, izmantojot citus konteineru veidošanas šablonus, arī *CirrOS, CentOS, Debian, OpenSUSE* sistēmu konteinerus, un arī tie darbojās normāli.

```
# lxc-create –t debian –n debian
# lxc-create –t opensuse –n opensuse
# lxc-create –t cirros –n cirros
# lxc-create –t centos –n centos
```

7.7 Nepriviliģētu konteineru veidošana

Lai veidotu nepriviliģētus konteinerus ir nepieciešams lietotājs, kuram pēc noklusējuma nav *sudo* tiesības, lai viņš, neievadot komandai *LXC* komandai priekšā ‘sudo’, varētu veidot nepriviliģētus konteinerus. Šim nolūkam tika izveidots speciāli jauns sistēmas lietotājs. Turpmākās darbības tika veiktas no sistēmas pamatlietotāja konta:

```
# useradd user
```

Jaunajam lietotājam jāpiešķir parole un jāizveido mājas mape, jo tajā glabāsies dati par konteineriem:

```
# passwd user
```

Pēc šīs komandas divreiz jāievada jaunā parole.

```
# mkhomedir_helper user
```

Kad lietotājs ir izveidots, tam ir jāpiešķir lietotāja un grupas apakšidentifikātori, kurus izmantos šī lietotāja veidotie konteineri.

```
# usermod --add-subuids 100000-165536 user
# usermod --add-subgids 100000-165536 user
```

Pēc šīm darbībām ir jāautentificējas sistēmā ar jaunizveidoto lietotāju, lai tā mājas mapē izveidotos visas nepieciešamās mapes, un tad var pārslēgties atpakaļ uz galveno sistēmas lietotāju.

Tālāk jāizveido '*lxc*' mape mapē '*/home/user/.config/*'.

```
# mkdir /home/user/.config/lxc
```

Pēc tam šajā mapē ir jāizveido sekojošs '*default.conf*'' fails:

/home/user/.config/lxc/default.conf

```
lxc.network.type = veth
lxc.network.link = lxcbr0
lxc.network.flags = up
lxc.network.hwaddr = 00:16:3e:xx:xx:xx
lxc.id_map = u 0 100000 65536
lxc.id_map = g 0 100000 65536
```

Pēc tam ir jāpārveido '*/etc/lxc/lxc-usernet*' fails par sekojošu:

/etc/lxc/lxc-usernet

```
# USERNAME TYPE BRIDGE COUNT
user veth lxcbr0 10
```

Pēdējā izpildītā komanda nosaka, ka šim lietotājam varēs būt tikai 10 konteineri.

Pēc tam ir jānodrošina pareizas tiesības jaunā lietotāja mājas mapei:

```
# chmod +x /home/user
```

Pēc šo darbību veikšanas vide nepriviliģētiem konteineriem ir sagatavota. Lai izveidotu neprivliģētu konteineri, jāpārslēdzas uz jaunizveidoto lietotāja kontu un jāveido konteineris izmantojot '*download*' šablonu, jo parasite šabloni nepriviliģētiem konteineriem nav pieejami. Arī nepriviliģēti konteineri darbojās normāli un līdz ar to darbs pie *LXC* bija beidzies.[27]

8. LIBVIRT LXC KONTEINERI

8.1 Libvirt LXC uzstādīšana un konteineru izveide

Lai varētu izmantot konteinerus *CentOS 6.6 Server* pamatsistēmā, ir jāuzstāda nevis *LXC*, bet *Libvirt* hipervizors, kuram ir iespēja realizēt *LXC* konteinerus. *Libvirt LXC* dzinim nav nekādu atkarību pret *LXC* lietotāja telpas rīkiem, jo tas tieši nodrošina svarīgās kodola īpašības, lai uzbūvētu konteinera vidi.

Pirms *Libvirt* uzstādīšanas ir jāveic sistēmas atjaunināšana, un tad var uzstādīt *Libvirt*:

```
# yum update
# yum install libvirt libvirt-client python-virtinst
```

Pēc šo komandu ievades ir jāapstiprina instalācija ar '*y*' taustiņu, un tiks uzinstalētas papildus citas pakotnes, no kurām instalējamās ir atkarīgas. Kad vide ir sagatavota, nepieciešams iedarbināt *Libvirt* dēmonu:

```
# service libvirtd start
```

Tālāk ir nepieciešams pārslēgt *Selinux* rīku, kas nodrošina sistēmas drošību ar likumu palīdzību, kuri nosaka, kuri rīki un procesi varēs izmantot kurus resursus, uz '*Permissive*' režīmu, jo konteineriem būs nepieciešama piekļuve resursiem, kuriem piekļuve nav atļauta. Var arī izveidot savu likumu, kas nodrošinās piekļuvi šiem resursiem, taču šī papildus drošība testa vidē nav nepieciešama.

```
# setenforce 0
```

Pēc vides sagatavošanas var sākt pirmā konteinera izveidi. Vispirms ir nepieciešams izveidot mapi, kurā glabāsies konteinera failsistēma un tajā mapi 'etc', kurā savukārt mapi '*yum*':

```
# mkdir /var/lib/libvirt/lxc/centos-6-x86_64/etc/yum.repos.d/ -p
```

Šajā '*yum*' mapē ir jāiekopē pamatsistēmas repozitoriju fails:

```
# cat /etc/yum.repos.d/CentOS-Base.repo |sed s/'$releasever'/6/g >
/var/lib/libvirt/lxc/centos-6-x86_64/etc/yum.repos.d/CentOS-Base.repo
```

Pēc tam var instalēt sistēmas pamatu un pāris papildus pakotnes izveidotajā mapē:

```
# yum groupinstall core --installroot=/var/lib/libvirt/centos-6-x86_64/
--nogpcheck –y
# yum install plymouth libselinux-python --
installroot=/var/lib/libvirt/lxc/centos-6-x86_64/ --nogpcheck –y
```

Pēc pamata sistēmas instalācijas jāpieslēdzas tai ar *chroot* iespēju un jāiespējo *root* lietotājs piešķirot tam paroli:

```
# chroot /var/lib/libvirt/lxc/centos-6-x86_64/
# echo girtsd | passwd root --stdin
```

Pēc tam ir jāiespējo *root* lietotāja autorizēšanās sistēma pievienojot papildus komandrindu '*/etc/securetty*' failā:

```
# echo "pts/0" >>/etc/securetty
```

Tālāk jāiespējo tīklošana sistēmā konfigurējot *eth0* tīkla interfeisu un uzstādot sistēmas vārdu:

```
# cat > /etc/sysconfig/network << EOF
NETWORKING=yes
HOSTNAME=lxc1.test.centos.org
EOF
# cat > /etc/sysconfig/network-scripts/ifcfg-eth0 << EOF
DEVICE=eth0
BOOTPROTO=dhcp
ONBOOT=yes
EOF
```

Pēc tam jāiedarbina *sshd* serviss, lai iespējotu *ssh* savienojumus, un tad var pamest *chroot* vidi:

```
# chkconfig sshd on
exit
```

Tālāk ir nepieciešams izmantojot komandrindas teksta redaktoru izkomentēt sekojošas rindiņas '*/var/lib/libvirt/lxc/centos-6-x86_64/etc/pam.d/login*' failā, pierakstot tām priekšā restīti:

```
session    required     pam_selinux.so close
session    required     pam_selinux.so open
session    required     pam_loginuid.so
```

un pēc tam izkomentēt sekojošas rindiņas '*/var/lib/libvirt/lxc/centos-6-x86_64/etc/pam.d/sshd*' failā:

```
session    required     pam_selinux.so close
session    required     pam_loginuid.so
session    required     pam_selinux.so open env_params
```

Pēc šo darbību veikšanas jādefinē izveidotā failsistēma kā konteineris, un tiks izveidots *XML* fails ar konteinera konfigurāciju mapē '*/etc/libvirt/lxc/*' ar nosaukumu '*test.xml*':

```
# virt-install --connect lxc:/// --name test --ram 512 --vcpu 1 --
filesystem /var/lib/libvirt/lxc/centos-6-x86_64/,/ --noautoconsole
```

Arguments '*--connect*' ar vērtību '*lxc:///*' paziņo *Libvirt*, lai tas pieslēdzas *LXC* dzinim, '*--name*' apzīmē konteinera vārdu, '*--ram*' apzīmē cik daudz operatīvās atmiņas resursus piešķirt konteinerim (noklusētā vērtība ir MB), '*--vcpu*' nosaka cik procesora kodolus piešķirt konteineim un '*--filesystem*' nosaka, kur atrodas 'konteinerizējamā' failsistēma.

Uzreiz pēc konteinera definīcijas tas tiek iedarbināts, taču, lai sāktu ar to strādāt, tas ir vispirms jārestartē.

```
# virsh --connect lxc:/// destroy test
# virsh --connect lxc:/// start test
```

Pēc konteinera restartēšanas var pieslēgties ta komandrindai:

```
# virsh --connect lxc:/// console test
```

Lai varētu strādāt ar konteineri, ir jāauterizējas ar iespējoto *root* lietotāju un uzstādīto 'girtsd' paroli. Pēc pieslēgšanās jāpamēģina veikt pāris vienkāršas darbības, piemēram, atjaunināt sistēmu un uzstādīt *Nano* komandrindas teksta redaktoru, lai pārliecinātos par pilnībā funkcionālu sistēmu:

```
# yum update
# yum install nano
```

Viss ir izdevies un konteineris ir pilnībā strādājošs. Lai pamestu konteinera vidi, ir līdzīgi kā *Ubuntu* pamatsistēmā jāievada izbēgšanas simbolu kombinācija, kas ir '*ctrl+]*'. Arī *Libvirt LXC* konteineri ir normāli strādājoši, un var ķerties pie nākamā darba posma, kas ir konteineru migrēšana starp pamatsistēmām.[23]

8.2 Libvirt LXC konteineru migrēšana uz Ubuntu pamatsistēmu

Tika pavadīts ilgs laiks, meklējot, kā migrēt konteinerus starp hipervizoriem un sistēmām, taču neizdevās atrast nevienu rakstisku paskaidrojumu vai pamācību, kā to izdarīt, lai arī bija atrasta informācija, ka tas ir izdarāms. Tika nolemts mēģināt šo migrēšanu veikt līdzīgi failsistēmas instalēšanai ar *Debootstrap*, vispirms pārceļot visu *CentOS* konteineri uz *Ubuntu* vidi, un tad izveidojot konteinerim tukšu 'fstab' un *LXC* konfigurācijas failu.

Pirmā lieta, kas ir jāizdara ir jāizveido no *CentOS* vides konteinera failsistēmas '.tar.gz' arhīvs, jo tas ļaus pārcelt failsistēmu saglabājot simboliskās saites, un jāpiešķir šim arhīvam citas tiesības. Sekojošās darbības tiek veiktas *CentOS* pamatsistēmā:

```
# cd /var/lib/libvirt/lxc
# tar -czvf container.tar.gz centos-6-x86_64/
# chmod 666 container.tar.gz
```

Tālāk šis izveidotais arhīvs ir jāpārceļ un *Ubuntu* pamatsistēmu, un tas tika darīts, kopējot failu pa tīklu. Tā kā abas darbstacijas nevarēja būt reizē iedarbinātas, tika izmantots starpniekdators, kurā tika izveidota mape '*/home/user/girts/*', un lietotājam 'user' tika piešķirtas pilnas tiesības pār šo mapi. Vispirms jāpārkopē fails uz starpniekdatoru:

```
# scp /var/lib/libvirt/lxc/container.tar.gz
user@10.3.7.162:/home/user/girts/
```

Pēc šīs komandas ievades ir jāievada lietotāja 'user' parole, lai apstiprinātu sūtīšanu. Pēc tam jāpārslēdzas uz *Ubuntu* pamatsistēmu un jāpārkopē fails uz to. Turpmākās komandas tika veiktas *Ubuntu* pamatsistēmā:

```
# scp user@10.3.7.162:/home/user/girts/container.tar.gz
/home/girts/Desktop/
```

Tālāk ir jāizveido mape, kurā glabāsies failsistēma un tajā jāatver arhīvs:

```
# mkdir /var/lib/lxc/centu
# cd /var/lib/lxc/centu
# tar –xzvf /home/girts/Desktop/container.tar.gz
```

Pēc tam ir jāizveido konteinera konfigurācijas fails ar nosaukumu 'config' tajā pat mapē, un faila saturam būtu jābūt sekojošam:

/var/lib/lxc/centu/config

```
# Common configuration
lxc.include = /usr/share/lxc/config/centos.common.conf

# Container specific configuration
lxc.rootfs = /var/lib/lxc/centu/centos-6-x86_64
lxc.utsname = centu
lxc.arch = x86_64

lxc.autodev = 0

# Network configuration
```

```
lxc.network.type = veth
lxc.network.flags = up
lxc.network.link = lxcbr0
lxc.network.hwaddr = 00:16:3e:32:14:5a
```

Pēc šī faila izveides konteinerim būtu jādarbojas, taču tā nebija, un kļūdas paziņojums informēja, ka failsistēma ir tikai lasīšanas režīmā. Šo kļūdu radīja serviss *udev*, ko var izkļaut no sistēmas startēšanās, lai kļūdu novērstu. Failā '*/var/lib/lxc/centu/centos-6-x86_64/etc/rc.d/rc.sysinit*' jāizkomentē sekojoša rindiņa, pierakstot tai priekšā restīti:

```
/sbin/start_udev
```

Pēc tam, darbinot konteineri, vēl projām neizdodas to ieslēgt, un kļūdas paziņojums informē, ka nav atrasts 'fstab' fails, toties vairs nav kļūda par tikai lasāmu failsistēmu. Tālāk jāpārkopē 'ubuntu' konteinera 'fstab' fails uz pārmigrēto konteineri.

```
# cp –i /var/lib/lxc/ubuntu/rootfs/etc/fstab /var/lib/lxc/centu/centos-6-x86_64/etc/fstab
```

Atkal startējot konteineri tas sastingst un neiesāknējas, taču vairs nav kļūdas paziņojuma par trūkstošu 'fstab' failu. Pēc šīs darbības tika nolemts paskatīties migrētā konteinera '/dev/' mapes saturu, jo šajā mapē glabājas iekārtu faili, kuri ir atbildīgi par saziņu starp programmatūru un aparatūru. Šaja mapē neatradās neviens iekārtas fails, tāpēc tika pārbaudīta arī *Ubuntu* pamatsistēmas 'ubuntu' konteinera '/dev/' mape, un tajā atradās vairāki iekārtu faili. Pēc šī atklājuma tika atrasts avots, kurā ir teikts, ka *Libvirt* ir atbildīgs par savu konteineru iekārtu failiem un tos nodrošina, emulējot tos.[24] Pēc šīs informācijas iegūšanas tika noprasts, ka vai nu tomēr nav paredzēts migrēt konteinerus no *Libvirt* uz *LXC*, vai arī tas prasa ļoti daudz darbības un zināšanas, kuras autoram nebija. Vēlreiz jāuzsver, ka netika atrasts neviens avots, kas šo migrāciju izskaidrotu.

8.3 LXC konteineru migrēšana uz Libvirt CentOS pamatsistēmā

Ar konteineru migrēšanu no *Libvirt* uz *LXC* radās problēmas, taču pretējā viziena migrēšanai, šīm problēmām nevajadzētu rasties, tāpēc tam vajadzētu izdoties. Līdzīgi kā iepriekš, arī migrējot konteinerus no *LXC* uz *Libvirt*, vispirms ir jāizveido arhīvs ar migrējamā konteinera failsistēmu, un par migrējamo konteineri tika izvēlēts 'apgraizītais' no šablona veidotais konteineris 'mazais'. Tātad vispirms ir jāizveido arhīvs un jānomaina tam tiesības. Turpmākās darbības tika veiktas *Ubuntu* pamatsistēmā

```
# cd /var/lib/lxc/mazais
# tar –czvf mazais.tar.gz rootfs/
```

```
# chmod 666  mazais.tar.gz
```

Pēc tam tā pat kā iepriekš jāpārkopē arhīvs uz otru darbstaciju, izmantojot starpnieku.

```
# scp /var/lib/lxc/mazais/mazais.tar.gz
user@10.3.7.162:/home/user/girts/
```

Pēc arhīva pārsūtīšanas ir jāpārslēdzas uz otru darbstaciju un japārkopē fails no starpniekdatora. Turpmākās darbības tika veiktas *CentOS* pamatsistēmā:

```
# scp user@10.3.7.162:/home/user/girts/mazais.tar.gz /home/
```

Pēc tam konteinera failsistēmas arhīvs ir jāatver mapē pie citiem *Libvirt* konteineriem:

```
# cd /var/lib/libvirt/lxc
# tar –zxvf /home/mazais.tar.gz
```

Tālāk ir jādefinē konteineris:

```
# virt-install --connect lxc:/// --name jaunais --ram 512 --vcpu 1 --
filesystem /var/lib/libvirt/lxc/rootfs/,/ --noautoconsole
```

Pēc konteinera definēšanas ir nepieciešams pievienot konteinera failam '*/etc/securetty*' papildus konsoli '*pts/0*', lai iespējotu *root* lietotāja sistēmas autentifikāciju.

```
# cat “pts/0” >> /var/lib/libvirt/lxc/rootfs/etc/securetty
```

Pēc tam ir jārestartē konteineris un jāmēģina tam pieslēgties:

```
# virsh –c lxc:/// destroy jaunais
# virsh –c lxc:/// start jaunais
# virsh –c lxc:/// console jaunais
```

Jāautorizējas ar iespējoto *root* lietotāju. Pēc pieslēgšanās konteinerim jāpārbauda, vai tas strādā.

```
# apt-get update
```

Viss strādā, kā nākas, un var pamest konteinera vidi ar taustiņu kombināciju '*ctrl+]*'.

Pēc šī pirmā konteinera migrācijas jāpārceļ arī pārējie *Ubuntu* pamatsistēmas *LXC* konteineri uz *CentOS* pamatsistēmas *Libvirt*. To jādara analoģiski tieši tā pat, kā ar konteineri 'mazais'.

Beigās uz *CentOS* pamatsistēmu izdevās pārcelt visus *Ubuntu* pamatsistēmas konteinerus, un visi darbojās normāli, izņemot *Debian* sistēmas konteineri. Tam bija problēmas ar ievades un izvades attēlošanu uz ekrāna, un tika atrasta informācija, ka tā kā *LXC* konteineri vēlprojām ir attīstības stadijā, vēlprojām pastāv kļūdas, kuras nav novērstas, un viena no tām ir šī. Tomēr tas

netraucētu konteineru izmantošanai, jo šim *Debian* konteinerim var pieslēgties ar *ssh* savienojumu un strādāt normāli, kas arī tika pārbaudīts.

8.4 Libvirt konteineru publiskās IP adreses

Lai piešķirtu *Libvirt* konteineriem publiskās IP adreses, ir nepieciešams izveidot tilta savienojumu, kurš tiks savienots ar pamatsistēmas *eth0* tīkla interfeisu. Piebilde: izmantot jau esošo *Libvirt* tīkla savienojumu nedrīkst. Vispirms ir jāizveido fails '*ifcfg-br0*' mapē '*/etc/sysconfig/network-scripts/*', kurā glabāsies tilta savienojuma konfigurācija, un failam ir jāizskatās sekojoši:

/etc/sysconfig/network-scripts/ifcfg-br0

```
DEVICE=br0
TYPE=Bridge
DELAY=0
ONBOOT=yes
BOOTPROTO=dhcp
NM_CONTROLLED=no
```

Pēc tam ir jāpamaina fails '*ifcfg-eth0*', kurš atrodas tajā pašā mapē, un tam ir jābūt sekojošam:

/etc/sysconfig/network-scripts/ifcfg-eth0

```
DEVICE="eth0"
HWADDR="D8:CB:8A:33:4F:2A"
NM_CONTROLLED=no
ONBOOT="yes"
TYPE="Ethernet"
UUID="129f5a4e-1eb6-47d4-94cc-b240e3c1e30f"
BRIDGE=br0
```

Tālāk ir jāpārstartē tīkla serviss:

```
# service network restart
```

Tālāk, lai piešķirtu jau izveidotam konteinerim publisko IP adresi, ir jāpalabo tā konfigurācijas fails, bet tas ir jādara, izmantojot special rīku, kas atver *vi* teksta redaktoru, nevis, ar roku mainot failu. Tika labota konteinera 'jaunais' konfigurācija.

```
# virsh –c lxc:/// edit jaunais
```

Fails ir rakstīts marķēšanas veidā, un kontekstā svarīga ir tikai sadaļa par tīkla interfeisu. Tai ir jāizskatās sekojoši:

```
<interface type='bridge'>
     <mac address='00:16:3e:11:cb:0e'/>
     <source bridge='br0'/>
```

```
</interface>
```

Pēc sadaļas izmainīšanas jānospiež taustiņš ‘*Esc*’ un pēc tam jāievada ‘ZZ’, lai saglabātu izmaiņas un izietu no teksta redaktora. Tālāk ir jāpārstartē konteineris, lai izmaiņas stātos spēkā un tad var pieslēgties konteinerim.

```
# virsh –c lxc:/// destroy jaunais
# virsh –c lxc:/// start jaunais
# virsh –c lxc:/// console jaunais
```

Jāautentificējas ar iespējoto *root* lietotāju, un tad jāievada komanda ‘*ifconfig*’, kas izvadīs informāciju par tīkla interfeisiem.

```
# ifconfig
```

```
etho  Link encap:Ethernet  HWaddr 00:16:3e:11:cb:0e
      inet addr:10.1.1.49  Bcast:10.1.1.255 Mask 255.255.255.0
```

Svarīga ir tikai augstāk redzamā daļa, kurā var redzēt, ka tīkla interfeisa ‘*eth0*’ IP adrese ir 10.1.1.49, kas ir darbinieku tīkla adrese.

Lai konteinerim tiktu piešķirta publiskā IP adrese jau definēšanas brīdī, pie definēšanas komandas ir jāpievieno papildus arguments. Tas tika izmēģināts, vispirms likvidējot un tad atkal definējot konteineri ‘jaunais’.

```
# virsh –c lxc:/// undefine jaunais
# virt-install --connect lxc:/// --name jaunais --ram 512 --vcpu 1 --w
bridge=br0 --filesystem /var/lib/libvirt/lxc/rootfs/,/ --noautoconsole
```

Pēc šo komandu izpildes tika izveidots konteineris jaunais, un tam tika piešķirta publiskā IP adrese. Konteineris vēlprojām darbojās normāli.

9. OPENSTACK MĀKONIS

9.1 Openstack pamatprincipi

Openstack ir bezmkasas atvērtā pirmkoda operējoša mākoņskaitļošanas sistēma, kas kontrolē lielus skaitļošanas, glabāšanas un tīklošanas resursu apjomus datucentrā un sniedz ērtas pārvaldības iespējas grafiskā vadības panelī, kā arī ar komandrindā. To bieži mēdz dēvēt arī par *IaaS (Infrastructure as a service)* risinājumu, kas nozīmē Infrastruktūra kā pakalpojums. Organizācija, kas nodarbojas ar *Openstack* attīstīšanu darbojas ar 6 mēnešu ciklu.[6]

Openstack nodrošina rīkus, kas ļauj realizēt un pārvaldīt mākoņskaitļošanas platformas gan publiskā, gan privātā līmenī. Tas ļauj lietotājiem darbināt virtuālās mašīnas un citas instances, kuras veic dažādus uzdevumus, kas palīdz pārvaldīt mākoņa vidi.

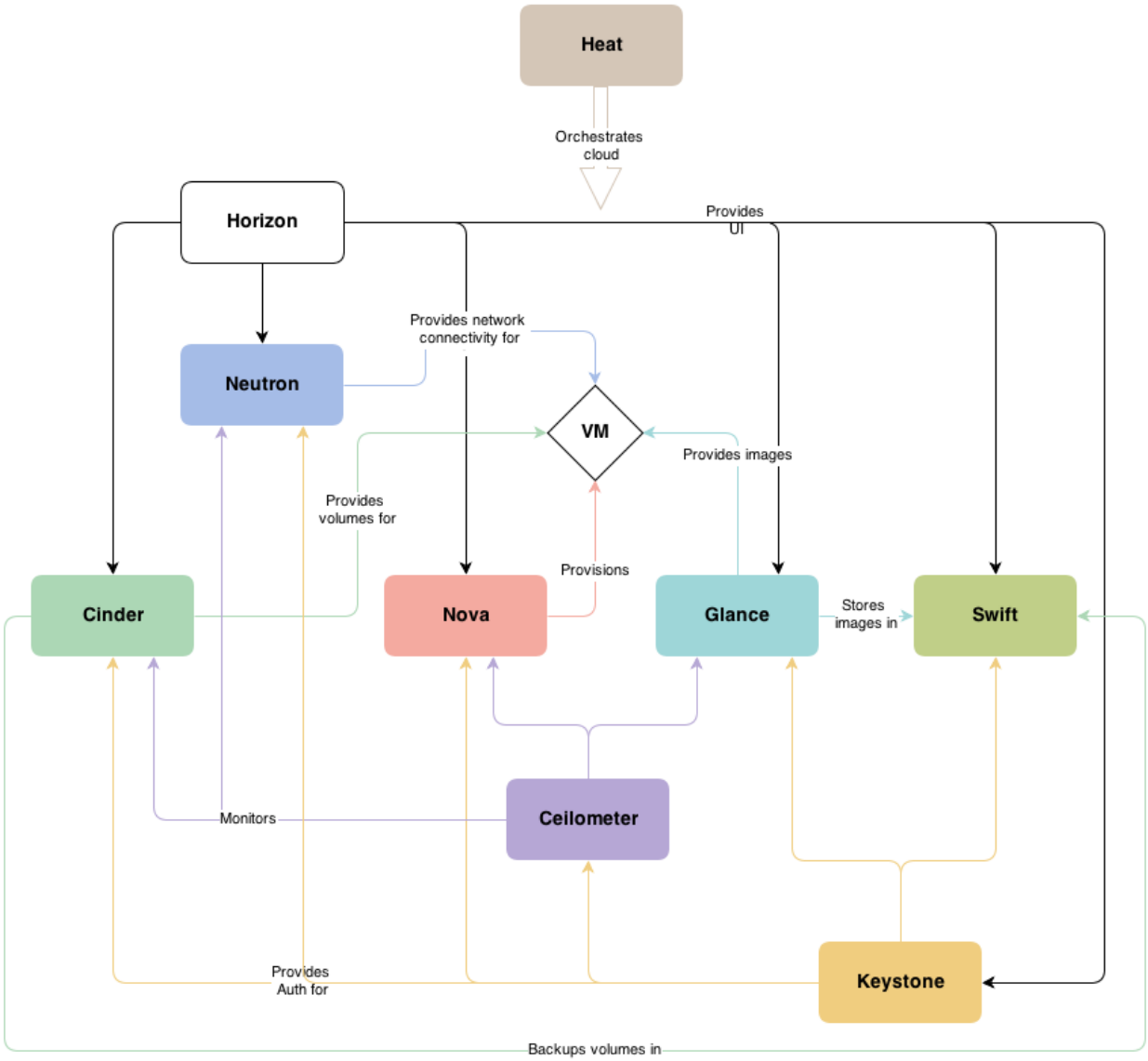
9.2 Openstack komponentes

Openstack ir izstrādāts pēc un tam piemīt servisu tipa arhitektūra ar dažādiem segvārdiem tā komponentēm, kuras katra ir atbildīga par sava servisa uzturēšanu un pārvaldīšanu. Komponentes un arī to skaits ir atšķirīgs dažādām *Openstack* versijām, tāpēc, lai nebūtu pārpratumi, tālāk ir aprakstītas tikai *Openstack Juno* komponentes.

- Skaitļošana (Nova) – Šis ir galvenais *Openstack* pārvaldības rīks, kas atbild par virtuālo mašīnu izvietošanu un pārraudzību lielos apjomos. Tas atbalsta dažādas virtualizācijas tehnoloģijas, kā piemēram *KVM, VMware, XEN* un arī konteineru realizācijas tehnoloģiju *LXC*.
- Objektu glabāšana (Swift) – Swift ir glabāšanas sistēma objektiem un failiem ar atmiņas mērogojamības iespēju. Dati tiek glabāti pa visu datucentru, kas ļauj neraizēties vai kāds no diskiem gadījumā nav pilns, un atvieglo mērogojamību.
- Bloku glabāšana (Cinder) – Šī komponente rūpējas par datu bloku glabāšanu un glabā datus konkrētās vietās, lai nodrošinātu ātrāku piekļuvi tiem.
- Tīklošana (Neutron) – Neutron uztur un pārvalda visu *Openstack* tīkla topoloģiju un nosaka, kas ar ko varēs veikt datu apmaiņu. Tas nodrošina virtuālo lokālo tīklu, peldošo IP adrešu, *DHCP* (automātiskā IP adrešu iegūšana un piešķiršana) un citu lietu lietošanu.
- Vadības panelis (Horizon) – *Openstack* vadības panelis, kas ar grafiskās saskarnes palīdzību ļauj viegli un ērti skatīt un pārvaldīt to, kas notiek mākonī. Tā ir ir vienīgā grafiskā saskarne, taču katrai komponentei ir savs *API,* kas ļauj pārvaldīt attiecīgo komponenti.

- Identitātes serviss (Keystone) – Keystone nodrošina centrālu mapi *Openstack* lietotājiem, kurā ir šiem lietotājiem pieejamie servisi un tas kalpo kā centrālā autentifikācijas sistēma ar dažādiem iespējojamiem autentifikācijas veidiem, piemēram lietotājvārds un parole, tekstvienību vērsta sistēma un citi.
- Attēlu serviss (Glance) – Šī komponente nodrošina virtuālo mašīnu veidošanai nepieciešamo attēlu pārraudzību, pievienošanu, glabāšanu un piegādi. To var izmantot arī rezerves kopiju glabāšanai un tas datu glabāšanai var izmantot dažādas aizmugurnodrošinājumus, ieskaitot *Cinder* komponenti.
- Telemetrija (Ceilometer) – paredzēts, lai monitorētu izmantotos mākoņa resursus un strādā uz vienkāršu skaitītāju pamata. Tas uzrāda informāciju par instancēm, piemēram to patērētās procesorstundas, darbības laiku, kā arī par citiem parametriem.
- Orķestrēšana (Heat) – Šī komponente ir paredzēta ērtai nepieciešamo parametru glabāšanai failā, kas definē, kādi resursi ir nepieciešami mākoņskaitļošanas lietojumprogrammai.

Visas šīs komponentes kopā veido *Openstack* un strādā kā vienota mākoņskaitļošanas sistēma.[7]



8.1.att. **Openstack Havana konceptuālā arhitektūra** *Avots:*

http://docs.openstack.org/juno/install-guide/install/apt/content/figures/1/a/common/figures/openstack_havana_conceptual_arch.png

10. KONTEINERU INTEGRĀCIJA OPENSTACK

Darba pēdējais solis un ceļš uz mērķa sasniegšanu ir konteineru integrācija mākonī. Lai aizstātu *KVM* virtualizācijas rīku ar *Libvirt* hipervizoru, ir nepieciešams uz Mezgls01 servera, uz kura ir uzstādīts *KVM* hipervizors, uzstādīt *nova-compute-lxc* pakotni, kuru uzstādot tiks pieprasīts likvidēt *KVM* pakotni, tādā veidā pilnībā aizstājot to. Šīs pakotnes uzstādīšanas laikā automātiksi tiks izveidots jauns papildus konfigurācijas fails, kurš saturēs informāciju par parametriem *compute-driver* un *virt_type*, un atradīsies */etc/nova/* mapē ar nosaukumu *nova-compute.conf* . [2]

```
apt-get install nova-compute-lxc
```

/etc/nova/nova-compute.conf

```
[Default]
Compute_driver=libvirt.LibvirtDriver
[libvirt]
Virt_type=lxc
```

Ir izdevies uzstādīt *Libvirt* kā hipervizoru mākonī un nākamais sagatavot attēlu (*image (.img)* tipa failu), kuru vajadzēs pievienot *Glance*, lai pēc šī faila varētu varētu veidot virtuālās mašīnas mākonī.

Nepieciešamā attēla veidošanas komandas uz *Ubuntu* hosta [3]:

```
cd /var/lib/lxc
truncate –size 2GB precise.img
# losetup –f precise.img
losetup –a
```

Izvads pēdējai komandai, no kā secinu, ka turpmākās komandās jāizmanto loop0 iekārta:

```
/dev/loop0: [0801]:10354801 (/var/lib/lxc/precise.img)
```

```
# mkfs /dev/loop0
mkdir mnt
# mount /dev/loop0 mnt
cd ubuntu
tar –czvf finale.tar.gz rootfs/
# chmod 666 finale.tar.gz
cd ..
```

```
cd mnt
tar –xvf /var/lib/lxc/ubuntu/finale.tar.gz
mv rootfs/* ./
rm –r rootfs
cd ..
# umount /dev/loop0
# losetup –d /dev/loop0
```

Konteinera attēls ir izveidots, izmantojot paša veidotā konteinera ‘ubuntu’ failsistēmu, un pēc attēla izveides ir nepieciešams sagatavot vidi, lai būtu piekļuve mākoņa *Glance* servisam, un varētu augšupielādēt izveidoto failu.

Komandas vides sagatavošanai un attēla augšupielādēšanai [4]:

```
# apt-get install ubuntu-cloud-keyring
# echo “deb http://ubuntu-cloud.archive.canonical.com/ubuntu” “trusty-updates/juno main” > /etc/apt/sources.list.d/cloudarchive-juno.list
# apt-get update && apt-get dist-upgrade
# apt-get install python-glanceclient
```

Pēc šiem soļiem ir pārlūkprogrammā jāievada IP adrese, ar kuru var nokļūt uz *Openstack* vadības paneli *Horizon*, un jāauterizējas. Pēc tam ir nepieciešam doties uz sadaļu “Project”, tālāk uz sadaļu “Compute” un jāatver iespēja “Access & Security”. Jaunajā logā ir jāizvēlas opcija “API Access”, kur būs iespējams uzspiezt uz pogas “Download Openstack RC File”, kas sāks faila lejupielādi. Pēc tam komandrindā ir jāiekļauj šis fails kā *source*, lai varētu komandrindā izpildīt *Glance* komandas. Kad tas ir izdarīts var pievienot pašizveidoto attēlu *Glance* attēlu krātuvei.

```
Source /home/girts/Downloads/admin-openrc.sh
glance image-create –name full-ubuntu –is-public True –disk-format raw –container-format bare –progress –file /var/lib/lxc/finale.img
```

Pēc faila pievienošanas *Glance* attēlu krātuvei ir iespējams mākonī startēt instanci (virtuālo mašīnu) no šī attēla. Tas tika pamēģināts un instance startējās normāli, kas liecina, ka jebkuru no veidotajiem konteineriem var pievienot *Glance* attēlu krātuvei, taču bija arī dažas problēmas. Jaunajai virtuālajai mašīnai nebija piekļuve internetam, jo tai nebija piešķirta peldošā IP adrese, taču kad izveidoja vēl vienu mašīnu tajā pašā tīklā, abas mašīnas savā starpā varēja sazināties un pēc pārbaudes secināju, ka šajā man piešķirtajā testa mākonī nebija pareizi sakonfigurētas

peldošās IP adreses, tātad viss darbojās no konteinera puses. Nākamā problēma bija tas, ka konteineriem nedarbojās *Openstack VNC* grafiskā komandrinda, bet pieslēgties varēja ar hipervizora iebūvēto tiešu teksta pieslēgumu. Tā ir vaina tajā, ka uz konteineriem nav uzstādīts *VNC* serveris, taču klienti savām virtuālajām mašīnām var pieslēgties ar *SSH* savienojumu, kā tas arī ir paredzēts. Nākamā problēma, kas 'uzpeldēja', bija tas, ka šīs jaunizveidotās virtuālās mašīnas nevarēja pareizi izslēgt, jo konteinera iekšienē darbojās viens process, kuru *Libvirt* serviss nespēja vai nemācēja apstādināt, un šo procesu, apturot manuāli, izmantojot 49erver komandrindu, varēja izslēgt virtuālo mašīnu. Problēmas tika sagaidītas, jo, lasot par konteineru integrāciju mākonī, tika atrasta informācija, ka konteineru turpmāka attīstība ir pārtraukta, un visas esošās kļūdas nav izlabotas, taču kopumā ir izdevies ieintegrēt konteinerus mākonī un galvenais mērķis ir sasniegts.

Ir parādījusies arī jauna konteineru realizācijas tehnoloģija paredzēta tieši integrācijai ar *Openstack* ar nosaukumu *LXD,* kas papildina jau esošā *LXC* pakotni. *LXD* nodrošina drošību starp konteineriem, pilnu virtuālo mašīnu iespējas, dzīvo migrāciju starp konteineriem un iespēju pievienot papildus atmiņu un virtuālās tīkla saskarnes. Papildus tam šī jauna programmatūra nodrošina maksimālu viesu skaitu, samazinot izmaksas mākonī, vieglu konteineru pārvaldību un procesu uzraudzību tieši no pamatsistēmas līmeņa, vieglu saskarni un vienu komandrindu ar pilnvērtīgu dokumentāciju un palīdzību, tūlītēju viesu startēšanu, labu savietojamību ar attēlu pārraudzības programmatūru kā arī drošību ar *Apparmor*, *user Namespaces* un *SECCOMP*. Integrācija *Openstack* tiek nodrošināta ar jauno pakotni *nova-compute-lxd*, kura ir pieejama sākot ar *Ubuntu 15.04,* un tai tiek nodrošināta saziņa ar attēlu pārvaldības rīku *Glance* un tīkla pārvaldības rīku *Neutron* tieši tāpat kā *KVM* virtuālajām mašīnām. [5]

11. KONFIGURĀCIJAS APRAKSTS

Darbā ir aprakstīta visa nepieciešamā konfigurācija katram darba posmam. Ir aprakstīta visa nepieciešamā konfigurācija, kas ir nepieciešama, lai sasniegtu darbā izvirzīto mērķi. Pati konfigurācija ir apskatāma darbā pie attiecīgās nodaļas. Piemēram, konteinera konfigurācija ir apskatām pie konteinera uzstādīšanas. Noklusētā konfigurācija un konfigurācija, kura netika mainīta darbā nav atrodama, jo, lai iekļautu pilnu *LXC, Libvirt, Docker, Openstack* un katra konteinera konfigurāciju, būtu nepieciešamas papildus 30 lapas ar pielikumiem.

12. KVALITĀTES NODROŠINĀŠANAS PASĀKUMU APRAKSTS

Lai nodrošinātu kvalitāti, tika pārbaudīta uzstādītā programmatūra un tās funkcionālā darbība.

LXC programmas kontekstā runāt par kvalitātes nodrošināšanu ir grūti, jo programma vēlprojām ir attīstības stadijā un vēlprojām ir diezgan daudz neatrisinātu kļūdu, taču programma ir viegli izprotama, un tai ir pietiekami daudz un viegli atrodama dokumentācija vai cita veida palīdzība, kas uzlabo pieredzi darbībā ar šo programmu.

Turpmāka šīs programmas darbība nebūtu vitāla, jo mērķis bija ieintegrēt konteinerus kā virtualizācijas rīku mākonī, kas arī tika sasniegts, taču šo programmu varētu izmantot arī turpmāk, lai sagatavotu jaunus virtuālo mašīnu attēlus mākonim. Lai nodrošinātu šīs programmas normālu darbību turpmāk, svarīgi būtu regulāri atjaunināt sistēmu un visas tās pakotnes, jo programmai, kas vēlprojām ir izstrādes stadijā ļoti bieži tiek publicēti atjauninājumi. Papildus tam būtu svarīgi regulāri, piemēram, reizi nedēļā, pārbaudīt programmas darbību izveidojot jaunus konteinerus un pārbaudot to funkcionalitāti. Svarīgi būtu saglabāt pieejamu arī šo dokumentu, jo tajā ir aprakstīts viss nepieciešamais programmas uzstādīšanai un darbam ar to.

Libvirt programmas kontekstā būtu jārīkojas tieši tāpat kā ar *LXC* programmu, jo to uzbūve un darbības principi ir ļoti līdzīgi, un arī šī programmatūra vēlprojām ir attīstības stadijā. Papildus tam būtu svarīgi sekot līdzi jaunākajām ziņām par programmas attīstību.

Tā kā galvenais mērķis ir sasniegts un darba galaprodukts ir mākonī integrējamu konteineru failsistēmu attēli un šī dokumentācija, būtu svarīgi pietiekami regulāri (vēlams katru dienu) pārbaudīt vai izveidotie konteineru attēli vēlprojām mākonī darbojas kā nākas, izveidojot jaunu instance no šiem attēliem un pārbaudot tās darbību. Tas ir svarīgi, jo arī *Openstack* mākonis vēlprojām tiek attīstīts, un reizi sešos mēnešos tiek publicēts tā jaunākais laidiens. Katrā nākamajā versijā var rasties kādas nesakritības ar jauno programmatūru un jau esošo, tāpēc būtu svarīgi sekot līdzi *Openstack* mākoņa attīstības jaunumiem, kā arī jaunu programmatūru sākt izmantot tikai pēc tam, kad tā ir bijusi pārbaudīta testa vidē.

Svarīgi ir minēt, ka *LXC* integrācija mākonī tālāk vairs attīstīta netiks, jo jau tiek strādāts pie *LXD* projekta, kas pēc princpia ir tas pats *LXC,* tikai veidots speciāli integrācijai mākonī. Šī iemesla dēļ, lai nodrošinātu ideālu kvalitāti, būtu jāseko līdzi *LXD* attīstībai, jo to nākotnē plāno integrēt mākoņa laidienā.

13. REZERVES KOPIJU VEIDOŠANAS PLĀNS

Tā kā darba galaprodukts ir dokumentācija, kā pamācība, un konteineru attēli, kā šabloni jaunām instancēm, tieši šīm lietām ir jāveido rezerves kopijas. Lai izveidotu rezerves kopiju dokumentācijai, viss, kas ir jāizdara ir jāpārkopē dokumentācijas fails uz drošu datu krātuvi, vēlams vairākām. To var saglabāt gan mākonī, vieglai piekļūšanai, gan kādā ārējā datu krātuvē, piemērām, *USB* zibatmiņā, gan kādā mākonī, kas piedāvā datu glabāšanu kā pakalpojumu, piemēram, *Dropbox*, lai dokumentācija būtu pieejama visur, kur ir tīkla savienojums.

Līdzīgi ir ar konteineru failsistēmu attēliem. Tā kā tie ir nemainīgi faili, tie vienkārši ir jāglabā drošā vietā gadījumam, ja nobrūk viss mākonis, tajā skaitā attēlu glabāšanas serviss *Glance*. Tāpat kā dokumentāciju šos attēlus var glabāt ārējos datu glabātājos un arī kādā mākonī.

14. DROŠĪBAS PASĀKUMU APRAKSTS

Veicot darbu tika strādāts kabinetā pie datora, kurš bija pieslēgts darbienieku tīklā, kā redzams attēlā 1.1. Šajā tīklā pieslēgtām ierīcēm ir piekļuve visiem apakštīkliem 1.1.attēlā, savukārt darbinieku tīkla ierīcēm piekļūt no internet nevar. Tiek uzturēti tikai esošie savienojumi, kas arī nodrošina darbinieku tīkla drošību. Arī mākonī ir līdzīga situācija, tikai no mākoņa apakštīkla var veidot savienojumus tikai ar internetu, vai arī uzturēt jau esošus savienojumus. Mākonī pārvaldības mezglam piekļūt var tikai no darbinieku tīkla vai īpašas IP adreses, kuras ir iekļautas kā izņēmums, taču klientu mezglam piekļuvi var regulēt. Var veidot virtuālās mašīnas, kurām var piekļūt tikai no vienas IP adreses un var veidot publiskas virtuālās mašīnas. Visi šie piekļuves ierobežojumi ir izveidoti kā ugunsmūra likumi galvenajā tīkla maršrutētājā.

Lai nodrošinātu aizsardzību lietojumprogrammu līmenī, visām sistēmu ietekmējošām darbībām ir nepieciešamas *root* tiesības. Arī ar *LXC* un *Libvirt* programmām līmenī, kas varētu apdraudēt sistēmu, strādāt var tikai ar *root* tiesībām.

SECINĀJUMI

Darbu pabeidzot, tika secināts, ka konteineri ir ļoti labs rīks, kas nodrošina pilnas sistēmas virtualizāciju. Lai arī vēl joprojām pastāv neatrisinātas problēmas konteinerus realizējošā programmatūrā, konteinerus ir iespējams realizēt kā sistēmas virtualizācijas rīku un arī ieviest mākonī kā galveno virtuālo mašīnu hipervizoru. Ar integrāciju mākonī neradās nekādas problēmas, jo *Linux* konteineriem un to realizēšanas rīkiem ir daudz un viegli atrodamas pamācības un dokumentācijas, kā ar tiem strādāt. Tiesa lielākā daļa šo avotu izklāsta, kā veikt vienkāršas darbības, bet sarežģītākām lietām, kā piemēram, konteineru migrēšanai starp operētājsistēmām un programmām un konteineru izveides skriptu veidošanai, pamācību nav, taču beigās arī šīm lietām lielu problēmu nebija.

Pirms darba veikšanas bija strādāts arī ar pilnās virtualizācijas risinājumiem, taču konteineri strādā ātrāk, ir ātrāk izveidojami, elastīgāki un prasa mazāk atmiņas. Tos nav tik viegli pārvaldīt, jo tiem nav grafiskās vides saskarnes, taču tas ir nesvarīgs sīkums, jo galvenais ir kvalitāte, nevis ērtības. Lielākais konteineru pluss ir to vieglā izveidošana un konfigurācijas maiņa, savukārt mīnuss ir tas, ka tie darbojas tikai vienas operētājsistēmas ietvaros.

Autors uzskata, ka konteineriem nākotnē būs ļoti liela nozīme, jo tie ne tikai sniedz risinajumu līdz šim neatrisināmām problēmām, bet arī piedāvā ļoti labu alternatīvu pilnās virtualizācijas risinājumiem. Attīstoties konteineru tehnoloģijai, uzlabosies to pārvaldības programmu kvalitāte. Jau šobrīd tiek strādāts pie jauna konteineru risinājuma speciāli integrēšanai mākonī ar nosaukumu *LXD*. Ļoti iespējams, ka konteineri kļūs par galveno sistēmas virtualizācijas rīku, kā arī par pieprasītāko rīku, procesu un programmu izolēšanai, jo nemitīgi tiek strādāts pie tā, lai konteineri nodrošinātu visas pilnās virtualizācijas funkcijas, bet patērētu daudz mazāk resursus.

IZMANTOTĀ LITERATŪRA

1. **Rouse, Margaret.** What is container-based virtualization (operating system-level virtualization)? - Definition from WhatIs.com. *techtarget.com.* [Tiešsaiste] [Citēts: 2015. gada 20. 04.] http://searchservervirtualization.techtarget.com/definition/container-based-virtualization-operating-system-level-virtualization.

2. LXC (Linux containers) - OpenStack Configuration Reference. *openstack.org.* [Tiešsaiste] [Citēts: 2015. gada 03. 05.] http://docs.openstack.org/kilo/config-reference/content/lxc.html.

3. ubuntu image for libvirt_type=lxc - Ask OpenStack: Q&A Site for OpenStack Users and Developers. *openstack.org.* [Tiešsaiste] [Citēts: 2015. gada 03. 05.] https://ask.openstack.org/en/question/14401/ubuntu-image-for-libvirt_typelxc/.

4. Chapter 2. Basic environment - OpenStack Installation Guide for Ubuntu 14.04 - juno. *openstack.org.* [Tiešsaiste] [Citēts: 2015. gada 03. 05.] http://docs.openstack.org/juno/install-guide/install/apt/content/ch_basic_environment.html#basics-packages.

5. LXD: the next-generation container hypervisor for Linux | Cloud | Ubuntu. *ubuntu.com.* [Tiešsaiste] [Citēts: 2015. gada 10. 05.] http://www.ubuntu.com/cloud/tools/lxd.

6. Openstack - Wikipedia, the free encyclopedia. *wikipedia.org.* [Tiešsaiste] 2015. gada 15. 05. [Citēts: 2015. gada 16. 05.] http://en.wikipedia.org/wiki/OpenStack.

7. What is OpenStack? | Opensource.com. *opensource.com.* [Tiešsaiste] [Citēts: 2015. gada 10. 05.] http://opensource.com/resources/what-is-openstack.

8. KVM. *linux-kvm.org.* [Tiešsaiste] [Citēts: 2015. gada 16. 05.] http://www.linux-kvm.org/page/Main_Page.

9. LXC. *ubuntu.com.* [Tiešsaiste] [Citēts: 2015. gada 28. 02.] https://help.ubuntu.com/lts/serverguide/lxc.html.

10. **Lezcano, Daniel.** lxc-console. *sourceforge.net.* [Tiešsaiste] [Citēts: 2015. gada 28. 02.] lxc.sourceforge.net/man/lxc-console.html.

11. lxc - Getting data from the outside into my linux container? - Ask Ubuntu. *askubuntu.com.* [Tiešsaiste] 2014. gada 09. 01. [Citēts: 2015. gada 08. 03.] http://askubuntu.com/questions/402425/getting-data-from-the-outside-into-my-linux-container.

12. **Pasztor, Janos.** LXC tutorial | Janos Pasztor. *janoszen.com.* [Tiešsaiste] 2013. gada 14. 05. [Citēts: 2015. gada 15. 03.] http://www.janoszen.com/wp-content/uploads/2013/05/lxc-minimal.sh.

13. Using lxc LinuX Containers on Debian 'Squeeze' - by Sebastiaan Giebels [Creating a LXC virtual machine template (from scratch)]. *pcprobleemloos.nl.* [Tiešsaiste] [Citēts: 2015. gada 15. 03.] http://wiki.pcprobleemloos.nl/using_lxc_linux_containers_on_debian_squeeze/creating_a_lxc_virtual_machine_template.

14. LXC. *ubuntu-it.org.* [Tiešsaiste] [Citēts: 2015. gada 16. 03.] http://help.ubuntu-it.org/12.04/server/serverguide/it/lxc.html.

15. Linux Containers - LXC - News. *linuxcontainers.org.* [Tiešsaiste] [Citēts: 2015. gada 16. 03.] https://linuxcontainers.org/lxc/news/.

16. **Graber, Stephane.** [lxc-devel] [PATCH] lxc-ubuntu: Remove trim option. *linuxcontainers.org.* [Tiešsaiste] 2013. gada 04. 10. [Citēts: 2015. gada 17. 03.] https://lists.linuxcontainers.org/pipermail/lxc-devel/2013-October/005650.html.

17. **Worrall, Daz.** Little script to create a minimal ubuntu using debootstrap and bring it up to date. *github.com.* [Tiešsaiste] 2012. gada 13. 12. [Citēts: 2015. gada 20. 03.] https://gist.github.com/DazWorrall/4277257.

18. debtree - Package dependency graphs. *debian.org.* [Tiešsaiste] [Citēts: 2015. gada 22. 03.] http://collab-maint.alioth.debian.org/debtree/.

19. Debootstrap - Debian Wiki. *debian.org.* [Tiešsaiste] 2014. gada 27. 10. [Citēts: 2015. gada 28. 03.] https://wiki.debian.org/Debootstrap.

20. 3.1. Overview of the Installation Process. *debian.org.* [Tiešsaiste] [Citēts: 2015. gada 28. 03.] https://www.debian.org/releases/stable/amd64/ch03s01.html.en.

21. linux - mount dev,proc,sys in a chroot environment? - Super User. *superuser.com.* [Tiešsaiste] 2010. gada 18. 07. [Citēts: 2015. gada 01. 04.] superuser.com/questions/165116/mount-dev-proc-sys-in-a-chroot-environment.

22. 10. Using Chroots - Ubuntu Packaging Guide. *ubuntu.com.* [Tiešsaiste] [Citēts: 2015. gada 03. 04.] http://packaging.ubuntu.com/html/chroots.html.

23. **Madjoudj, Athmane.** HowTos/LXC-on-CentOS6 - Centos Wiki. *centos.org.* [Tiešsaiste] 2014. gada 28. 02. [Citēts: 2015. gada 10. 04.] http://wiki.centos.org/HowTos/LXC-on-CentOS6.

24. libvirt: LXC container driver. *libvirt.org.* [Tiešsaiste] [Citēts: 2015. gada 20. 04.] https://libvirt.org/drvlxc.html#devnodes.

25. What Is Docker? An open platform for distributed apps. *docker.com.* [Tiešsaiste] [Citēts: 2015. gada 20. 03.] http://www.docker.com/whatisdocker/.

26. Docker basics - Docker Documentation. *docker.com.* [Tiešsaiste] [Citēts: 2015. gada 21. 03.] http://docs.docker.com/articles/basics/.

27. **Graber, Stephane.** LXC 1.0: Unprivileged containers [7/10] Stephane Graber`s website. *stgraber.org.* [Tiešsaiste] 2014. gada 17. 01. [Citēts: 2015. gada 05. 05.] https://www.stgraber.org/2014/01/17/lxc-1-0-unprivileged-containers/.

PIELIKUMI

1. pielikums [12]

lxc-minimal

```
#!/bin/bash

# Crash on any error
set -e

# Load LXC defaults
if [ -r /etc/default/lxc ]; then
    . /etc/default/lxc
fi

# The install function installs the minimal operating system in the
LXC
install() {
    DLDIR=/tmp/download
    DPKGDIR=/tmp/dpkg
    ROOTDIR=$1

    # Create package download directory
    rm -rf $DLDIR
    mkdir -p $DLDIR
    # Create fake dpkg status directory
    rm -rf $DPKGDIR
    mkdir -p $DPKGDIR
    mkdir -p $DPKGDIR/updates
    mkdir -p $DPKGDIR/info
    echo  -n "" >$DPKGDIR/status
    echo  -n "" >$DPKGDIR/available

    # Create VE root directory
    mkdir -p $ROOTDIR

    # Download packages
    pushd $DLDIR >/dev/null
    apt-get download busybox-static
    popd >/dev/null

    # Install the packages in the VE root
    dpkg -i --instdir=$ROOTDIR --admindir=$DPKGDIR $DLDIR/busybox-
static*.deb

    # Create /dev, /proc and /sys
    mkdir -p $ROOTDIR/etc
```

```
    mkdir -p $ROOTDIR/etc/init.d
    mkdir -p $ROOTDIR/dev
    mkdir -p $ROOTDIR/dev/shm
    mkdir -p $ROOTDIR/dev/pts
    mkdir -p $ROOTDIR/proc
    mkdir -p $ROOTDIR/sys

    # Busybox can run as an init binary, use it
    mkdir -p $ROOTDIR/sbin
    ln $ROOTDIR/bin/busybox $ROOTDIR/sbin/init
    ln $ROOTDIR/bin/busybox $ROOTDIR/sbin/getty

    chmod +x $ROOTDIR/sbin/init

    # Create a symlink so the console actually works
    ln $ROOTDIR/bin/static-sh $ROOTDIR/bin/sh

    # Create minimum required device nodes in /dev
    mknod $ROOTDIR/dev/null c 1 3
    mknod $ROOTDIR/dev/zero c 1 5
    mknod $ROOTDIR/dev/console c 5 0
    mknod $ROOTDIR/dev/tty c 5 0
    mknod $ROOTDIR/dev/tty0 c 4 0
    mknod $ROOTDIR/dev/tty1 c 4 0
    mknod $ROOTDIR/dev/tty5 c 4 0
    mknod $ROOTDIR/dev/random c 1 8
    mknod $ROOTDIR/dev/urandom c 1 9
    mknod $ROOTDIR/dev/ram0 b 1 0
    chmod 666 $ROOTDIR/dev/tty $ROOTDIR/dev/console
$ROOTDIR/dev/ram0 $ROOTDIR/dev/tty0 $ROOTDIR/dev/null

    # Alternatively we could install udev in the VE

    # Set up /etc/passwd so we have users
    cat <<EOF >> $ROOTDIR/etc/passwd
root:x:0:0:root:/root:/bin/sh
EOF
    # Set up /etc/group so we have users
    cat <<EOF >> $ROOTDIR/etc/group
root:x:0:root
EOF

    # Create the rcS file which would be the startup script
    cat <<EOF >> $ROOTDIR/etc/init.d/rcS
#!/bin/sh
# Do nothing
EOF
```

```
    chmod 744 $ROOTDIR/etc/init.d/rcS

    # Create the fstab file
    cat <<EOF >> $ROOTDIR/etc/fstab
proc  /proc      proc    defaults     0      0
shm   /dev/shm   tmpfs   defaults     0      0
EOF

    # Create inittab which busybox can run
    cat <<EOF >> $ROOTDIR/etc/inittab
::sysinit:/etc/init.d/rcS
tty1::respawn:/sbin/getty -i -L -n -l /bin/sh tty1 38400 vt102
console::askfirst:/bin/sh
EOF
    chmod 644 $ROOTDIR/etc/inittab
}

# Create the configuration file. Shamelessly copied from the Ubuntu
template
configure() {
    ROOTDIR=$1
    NAME=$2
    PATH=/var/lib/lxc/$NAME
    TTYDIR=

    # if there is exactly one veth network entry, make sure it has
an
    # associated hwaddr.
    nics=`/bin/grep -e '^lxc\.network\.type[ \t]*=[ \t]*veth'
$PATH/config | /usr/bin/wc -l`
    if [ $nics -eq 1 ]; then
        /bin/grep -q "^lxc.network.hwaddr" $PATH/config || /bin/cat
<<EOF >> $PATH/config
lxc.network.hwaddr = 00:16:3e:$(/usr/bin/openssl rand -hex 3|
/bin/sed 's/\(..\)/\1:/g; s/.$//')
EOF
    fi

    /bin/cat <<EOF >> $PATH/config
lxc.utsname = $NAME

lxc.devttydir =
lxc.tty = 1
lxc.pts = 1024
lxc.rootfs = $ROOTDIR
lxc.mount  = $PATH/fstab
lxc.cap.drop = sys_module mac_admin
```

```
lxc.pivotdir = lxc_putold

# uncomment the next line to run the container unconfined:
#lxc.aa_profile = unconfined

lxc.cgroup.devices.deny = a
# Allow any mknod (but not using the node)
lxc.cgroup.devices.allow = c *:* m
lxc.cgroup.devices.allow = b *:* m
# /dev/null and zero
lxc.cgroup.devices.allow = c 1:3 rwm
lxc.cgroup.devices.allow = c 1:5 rwm
# consoles
lxc.cgroup.devices.allow = c 5:1 rwm
lxc.cgroup.devices.allow = c 5:0 rwm
#lxc.cgroup.devices.allow = c 4:0 rwm
#lxc.cgroup.devices.allow = c 4:1 rwm
# /dev/{,u}random
lxc.cgroup.devices.allow = c 1:9 rwm
lxc.cgroup.devices.allow = c 1:8 rwm
lxc.cgroup.devices.allow = c 136:* rwm
lxc.cgroup.devices.allow = c 5:2 rwm
# rtc
lxc.cgroup.devices.allow = c 254:0 rwm
#fuse
lxc.cgroup.devices.allow = c 10:229 rwm
#tun
lxc.cgroup.devices.allow = c 10:200 rwm
#full
lxc.cgroup.devices.allow = c 1:7 rwm
#hpet
lxc.cgroup.devices.allow = c 10:228 rwm
#kvm
lxc.cgroup.devices.allow = c 10:232 rwm
EOF

    /bin/cat <<EOF > $path/fstab
proc            proc         proc    nodev,noexec,nosuid 0 0
sysfs           sys          sysfs defaults  0 0
EOF

    if [ $? -ne 0 ]; then
        /bin/echo "Failed to add configuration"
        return 1
    fi
}
```

```
# Set up the path option
options=$(getopt -o n:p: -l path:,name: -- "$@")
if [ $? -ne 0 ]; then
    exit 1
fi
eval set -- "$options"

# Fetch command line parameters
while true; do
    case "$1" in
        -p|--path)      path=$2; shift 2;;
        -n|--name)      name=$2; shift 2;;
        --)             shift 1; break ;;
        *)              break ;;
    esac
done

if [ -z "$path" ]; then
    echo "'path' parameter is required"
    exit 1
fi

if [ "$(id -u)" != "0" ]; then
    echo "This script should be run as 'root'"
    exit 1
fi

rootfs=$path/rootfs

install $rootfs

configure $rootfs $name
```

2. pielikums [17]

little.sh

```
#!/bin/bash
set -e
DEFAULT_PACKAGES=ssh,language-pack-en-base
DEFAULT_COMPONENTS=main,universe
DEBOOTSTRAP=/usr/sbin/debootstrap
DEFAULT_MIRROR=http://archive.ubuntu.com/ubuntu
DEFAULT_VARIANT=minbase

MIRROR=${STRAP_MIRROR:-$DEFAULT_MIRROR}
ROOTFS=$1
PACKAGES=${STRAP_PACKAGES:-$DEFAULT_PACKAGES}
```

```
COMPONENTS=${STRAP_COMPONENTS:-$DEFAULT_COMPONENTS}
SUITE=${STRAP_SUITE:-precise}
VARIANT=${STRAP_VARIANT:-$DEFAULT_VARIANT}

type $DEBOOTSTRAP >/dev/null
if [ $? -ne 0 ]; then
    echo "debootstrap not installed"
    exit 1
fi

if [ "x$ROOTFS" == "x" ]; then
    echo "Usage: $0 <path_to_root>"
    exit 1
fi

$DEBOOTSTRAP --include $PACKAGES --components $COMPONENTS --variant
$VARIANT $SUITE $ROOTFS $MIRROR

cat > $ROOTFS/etc/apt/sources.list << EOF
deb $MIRROR $SUITE ${COMPONENTS//,/ }
deb $MIRROR $SUITE-updates ${COMPONENTS//,/ }
deb $MIRROR $SUITE-security ${COMPONENTS//,/ }
EOF

chroot $ROOTFS mount -t proc /proc /proc
chroot $ROOTFS apt-get update
chroot $ROOTFS apt-get dist-upgrade -y
umount $ROOTFS/proc
```

3. pielikums

Komandas ‘dpkg –l’ izvads

```
Desired=Unknown/Install/Remove/Purge/Hold
| Status=Not/Inst/Conf-files/Unpacked/halF-conf/Half-inst/trig-aWait/Trig-pend
|/ Err?=(none)/Reinst-required (Status,Err: uppercase=bad)
||/ Name                          Version            Architecture       Description
+++-=============================-==================-==================-
================================================================
ii  adduser                       3.113+nmu3ubuntu3  all                add and
remove users and groups
ii  apt                           1.0.1ubuntu2.6     amd64              commandline
package manager
ii  apt-utils                     1.0.1ubuntu2.6     amd64              package
management related utility programs
ii  base-files                    7.2ubuntu5.1       amd64              Debian base
system miscellaneous files
ii  base-passwd                   3.5.33             amd64              Debian base
system master password and group files
ii  bash                          4.3-7ubuntu1.5     amd64              GNU Bourne
Again SHell
```

```
ii  bsdutils                     1:2.20.1-5.1ubuntu amd64              Basic
utilities from 4.4BSD-Lite
ii  busybox-initramfs            1:1.21.0-1ubuntu1  amd64              Standalone
shell setup for initramfs
ii  bzip2                        1.0.6-5            amd64              high-
quality block-sorting file compressor - utilities
ii  console-setup                1.70ubuntu8        all                console
font and keymap setup program
ii  coreutils                    8.21-1ubuntu5.1    amd64              GNU core
utilities
ii  cpio                         2.11+dfsg-1ubuntu1 amd64              GNU cpio --
a program to manage archives of files
ii  cron                         3.0pl1-124ubuntu2  amd64              process
scheduling daemon
ii  dash                         0.5.7-4ubuntu1     amd64              POSIX-
compliant shell
ii  debconf                      1.5.51ubuntu2      all                Debian
configuration management system
ii  debconf-i18n                 1.5.51ubuntu2      all                full
internationalization support for debconf
ii  debianutils                  4.4                amd64
Miscellaneous utilities specific to Debian
ii  dh-python                    1.20140128-1ubuntu all                Debian
helper tools for packaging Python libraries and appli
ii  diffutils                    1:3.3-1            amd64              File
comparison utilities
ii  dmsetup                      2:1.02.77-6ubuntu2 amd64              Linux
Kernel Device Mapper userspace library
ii  dpkg                         1.17.5ubuntu5.3    amd64              Debian
package management system
ii  e2fslibs:amd64               1.42.9-3ubuntu1    amd64
ext2/ext3/ext4 file system libraries
ii  e2fsprogs                    1.42.9-3ubuntu1    amd64
ext2/ext3/ext4 file system utilities
ii  eject                        2.1.5+deb1+cvs2008 amd64              ejects CDs
and operates CD-Changers under Linux
ii  file                         1:5.14-2ubuntu3.3  amd64              Determines
file type using "magic" numbers
ii  findutils                    4.4.2-7            amd64              utilities
for finding files--find, xargs
ii  gcc-4.8-base:amd64           4.8.2-19ubuntu1    amd64              GCC, the
GNU Compiler Collection (base package)
ii  gcc-4.9-base:amd64           4.9.1-0ubuntu1     amd64              GCC, the
GNU Compiler Collection (base package)
ii  gnupg                        1.4.16-1ubuntu2.1  amd64              GNU privacy
guard - a free PGP replacement
ii  gpgv                         1.4.16-1ubuntu2.1  amd64              GNU privacy
guard - signature verification tool
ii  grep                         2.16-1             amd64              GNU grep,
egrep and fgrep
ii  gzip                         1.6-3ubuntu1       amd64              GNU
compression utilities
ii  hostname                     3.15ubuntu1        amd64              utility to
set/show the host name or domain name
ii  ifupdown                     0.7.47.2ubuntu4.1  amd64              high level
tools to configure network interfaces
```

```
ii  init-system-helpers         1.14               all              helper
tools for all init systems
ii  initramfs-tools             0.103ubuntu4.2     all              tools for
generating an initramfs
ii  initramfs-tools-bin         0.103ubuntu4.2     amd64            binaries
used by initramfs-tools
ii  initscripts                 2.88dsf-41ubuntu6  amd64            scripts for
initializing and shutting down the system
ii  insserv                     1.14.0-5ubuntu2    amd64            boot
sequence organizer using LSB init.d script dependency i
ii  iproute2                    3.12.0-2           amd64            networking
and traffic control tools
ii  iputils-ping                3:20121221-4ubuntu amd64            Tools to
test the reachability of network hosts
ii  isc-dhcp-client             4.2.4-7ubuntu12    amd64            ISC DHCP
client
ii  isc-dhcp-common             4.2.4-7ubuntu12    amd64            common
files used by all the isc-dhcp* packages
ii  kbd                         1.15.5-1ubuntu1    amd64            Linux
console font and keytable utilities
ii  keyboard-configuration      1.70ubuntu8        all              system-wide
keyboard preferences
ii  klibc-utils                 2.0.3-0ubuntu1     amd64            small
utilities built with klibc for early boot
ii  kmod                        15-0ubuntu6        amd64            tools for
managing Linux kernel modules
ii  language-pack-en            1:14.04+20141110   all              translation
updates for language English
ii  language-pack-en-base       1:14.04+20140707   all
translations for language English
ii  less                        458-2              amd64            pager
program similar to more
ii  libacl1:amd64               2.2.52-1           amd64            Access
control list shared library
ii  libapt-inst1.5:amd64        1.0.1ubuntu2.6     amd64            deb package
format runtime library
ii  libapt-pkg4.12:amd64        1.0.1ubuntu2.6     amd64            package
management runtime library
ii  libarchive-extract-perl     0.70-1             all              generic
archive extracting module
ii  libattr1:amd64              1:2.4.47-1ubuntu1  amd64            Extended
attribute shared library
ii  libaudit-common             1:2.3.2-2ubuntu1   all              Dynamic
library for security auditing - common files
ii  libaudit1:amd64             1:2.3.2-2ubuntu1   amd64            Dynamic
library for security auditing
ii  libblkid1:amd64             2.20.1-5.1ubuntu20 amd64            block
device id library
ii  libbsd0:amd64               0.6.0-2ubuntu1     amd64            utility
functions from BSD systems - shared library
ii  libbz2-1.0:amd64            1.0.6-5            amd64            high-
quality block-sorting file compressor library - runtime
ii  libc-bin                    2.19-0ubuntu6.5    amd64            Embedded
GNU C Library: Binaries
ii  libc6:amd64                 2.19-0ubuntu6.5    amd64            Embedded
GNU C Library: Shared libraries
```

```
ii  libcap2:amd64               1:2.24-0ubuntu2    amd64              support for
getting/setting POSIX.1e capabilities
ii  libcap2-bin                 1:2.24-0ubuntu2    amd64              basic
utility programs for using capabilities
ii  libcgmanager0:amd64         0.24-0ubuntu7.2    amd64              Central
cgroup manager daemon (client library)
ii  libck-connector0:amd64      0.4.5-3.1ubuntu2   amd64              ConsoleKit
libraries
ii  libcomerr2:amd64            1.42.9-3ubuntu1    amd64              common
error description library
ii  libdb5.3:amd64              5.3.28-3ubuntu3    amd64              Berkeley
v5.3 Database Libraries [runtime]
ii  libdbus-1-3:amd64           1.6.18-0ubuntu4.3  amd64              simple
interprocess messaging system (library)
ii  libdebconfclient0:amd64     0.187ubuntu1       amd64              Debian
Configuration Management System (C-implementation lib
ii  libdevmapper1.02.1:amd64    2:1.02.77-6ubuntu2 amd64              Linux
Kernel Device Mapper userspace library
ii  libdrm2:amd64               2.4.56-1~ubuntu1   amd64              Userspace
interface to kernel DRM services -- runtime
ii  libedit2:amd64              3.1-20130712-2     amd64              BSD
editline and history libraries
ii  libestr0                    0.1.9-0ubuntu2     amd64              Helper
functions for handling strings (lib)
ii  libexpat1:amd64             2.1.0-4ubuntu1     amd64              XML parsing
C library - runtime library
ii  libffi6:amd64               3.1~rc1+r3.0.13-12 amd64              Foreign
Function Interface library runtime
ii  libfribidi0:amd64           0.19.6-1           amd64              Free
Implementation of the Unicode BiDi algorithm
ii  libgcc1:amd64               1:4.9.1-0ubuntu1   amd64              GCC support
library
ii  libgcrypt11:amd64           1.5.3-2ubuntu4.1   amd64              LGPL Crypto
library - runtime library
ii  libgdbm3:amd64              1.8.3-12build1     amd64              GNU dbm
database routines (runtime version)
ii  libgnutls-openssl27:amd64   2.12.23-12ubuntu2. amd64              GNU TLS
library - OpenSSL wrapper
ii  libgnutls26:amd64           2.12.23-12ubuntu2. amd64              GNU TLS
library - runtime library
ii  libgpg-error0:amd64         1.12-0.2ubuntu1    amd64              library for
common error values and messages in GnuPG compon
ii  libgpm2:amd64               1.20.4-6.1         amd64              General
Purpose Mouse - shared library
ii  libgssapi-krb5-2:amd64      1.12+dfsg-2ubuntu5 amd64              MIT
Kerberos runtime libraries - krb5 GSS-API Mechanism
ii  libjson-c2:amd64            0.11-3ubuntu1.2    amd64              JSON
manipulation library - shared library
ii  libjson0:amd64              0.11-3ubuntu1.2    amd64              JSON
manipulation library (transitional package)
ii  libk5crypto3:amd64          1.12+dfsg-2ubuntu5 amd64              MIT
Kerberos runtime libraries - Crypto Library
ii  libkeyutils1:amd64          1.5.6-1            amd64              Linux Key
Management Utilities (library)
ii  libklibc                    2.0.3-0ubuntu1     amd64              minimal
libc subset for use with initramfs
```

```
ii  libkmod2:amd64              15-0ubuntu6        amd64              libkmod
shared library
ii  libkrb5-3:amd64             1.12+dfsg-2ubuntu5 amd64              MIT
Kerberos runtime libraries
ii  libkrb5support0:amd64       1.12+dfsg-2ubuntu5 amd64              MIT
Kerberos runtime libraries - Support library
ii  liblocale-gettext-perl      1.05-7build3       amd64              module
using libc functions for internationalization in Perl
ii  liblockfile-bin             1.09-6ubuntu1      amd64              support
binaries for and cli utilities based on liblockfile
ii  liblockfile1:amd64          1.09-6ubuntu1      amd64              NFS-safe
locking library
ii  liblog-message-simple-perl  0.10-1             all                simplified
interface to Log::Message
ii  liblzma5:amd64              5.1.1alpha+2012061 amd64              XZ-format
compression library
ii  libmagic1:amd64             1:5.14-2ubuntu3.3  amd64              File type
determination library using "magic" numbers
ii  libmodule-pluggable-perl    5.1-1              all                module for
giving  modules the ability to have plugins
ii  libmount1:amd64             2.20.1-5.1ubuntu20 amd64              block
device id library
ii  libmpdec2:amd64             2.4.0-6            amd64              library for
decimal floating point arithmetic (runtime libra
ii  libncurses5:amd64           5.9+20140118-1ubun amd64              shared
libraries for terminal handling
ii  libncursesw5:amd64          5.9+20140118-1ubun amd64              shared
libraries for terminal handling (wide character suppo
ii  libnewt0.52:amd64           0.52.15-2ubuntu5   amd64              Not Erik's
Windowing Toolkit - text mode windowing with slan
ii  libnih-dbus1:amd64          1.0.3-4ubuntu25    amd64              NIH D-Bus
Bindings Library
ii  libnih1:amd64               1.0.3-4ubuntu25    amd64              NIH Utility
Library
ii  libp11-kit0:amd64           0.20.2-2ubuntu2    amd64              Library for
loading and coordinating access to PKCS#11 modul
ii  libpam-cap:amd64            1:2.24-0ubuntu2    amd64              PAM module
for implementing capabilities
ii  libpam-modules:amd64        1.1.8-1ubuntu2     amd64              Pluggable
Authentication Modules for PAM
ii  libpam-modules-bin          1.1.8-1ubuntu2     amd64              Pluggable
Authentication Modules for PAM - helper binaries
ii  libpam-runtime              1.1.8-1ubuntu2     all                Runtime
support for the PAM library
ii  libpam0g:amd64              1.1.8-1ubuntu2     amd64              Pluggable
Authentication Modules library
ii  libpcre3:amd64              1:8.31-2ubuntu2    amd64              Perl 5
Compatible Regular Expression Library - runtime files
ii  libplymouth2:amd64          0.8.8-0ubuntu17.1  amd64              graphical
boot animation and logger - shared libraries
ii  libpng12-0:amd64            1.2.50-1ubuntu2    amd64              PNG library
- runtime
ii  libpod-latex-perl           0.61-1             all                module to
convert Pod data to formatted LaTeX
ii  libpopt0:amd64              1.16-8ubuntu1      amd64              lib for
parsing cmdline parameters
```

```
ii  libprocps3:amd64            1:3.3.9-1ubuntu2.2 amd64              library for
accessing process information from /proc
ii  libpython2.7:amd64          2.7.6-8            amd64              Shared
Python runtime library (version 2.7)
ii  libpython2.7-minimal:amd64  2.7.6-8            amd64              Minimal
subset of the Python language (version 2.7)
ii  libpython2.7-stdlib:amd64   2.7.6-8            amd64              Interactive
high-level object-oriented language (standard li
ii  libpython3-stdlib:amd64     3.4.0-0ubuntu2     amd64              interactive
high-level object-oriented language (default pyt
ii  libpython3.4-minimal:amd64  3.4.0-2ubuntu1     amd64              Minimal
subset of the Python language (version 3.4)
ii  libpython3.4-stdlib:amd64   3.4.0-2ubuntu1     amd64              Interactive
high-level object-oriented language (standard li
ii  libreadline6:amd64          6.3-4ubuntu2       amd64              GNU
readline and history libraries, run-time libraries
ii  libselinux1:amd64           2.2.2-1ubuntu0.1   amd64              SELinux
runtime shared libraries
ii  libsemanage-common          2.2-1              all                Common
files for SELinux policy management libraries
ii  libsemanage1:amd64          2.2-1              amd64              SELinux
policy management library
ii  libsepol1:amd64             2.2-1ubuntu0.1     amd64              SELinux
library for manipulating binary security policies
ii  libslang2:amd64             2.2.4-15ubuntu1    amd64              S-Lang
programming library - runtime version
ii  libsqlite3-0:amd64          3.8.2-1ubuntu2     amd64              SQLite 3
shared library
ii  libss2:amd64                1.42.9-3ubuntu1    amd64              command-
line interface parsing library
ii  libssl1.0.0:amd64           1.0.1f-1ubuntu2.8  amd64              Secure
Sockets Layer toolkit - shared libraries
ii  libstdc++6:amd64            4.8.2-19ubuntu1    amd64              GNU
Standard C++ Library v3
ii  libtasn1-6:amd64            3.4-3ubuntu0.1     amd64              Manage
ASN.1 structures (runtime)
ii  libterm-ui-perl             0.42-1             all
Term::ReadLine UI made easy
ii  libtext-charwidth-perl      0.04-7build3       amd64              get display
widths of characters on the terminal
ii  libtext-iconv-perl          1.7-5build2        amd64              converts
between character sets in Perl
ii  libtext-soundex-perl        3.4-1build1        amd64
implementation of the soundex algorithm
ii  libtext-wrapi18n-perl       0.06-7             all
internationalized substitute of Text::Wrap
ii  libtinfo5:amd64             5.9+20140118-1ubun amd64              shared low-
level terminfo library for terminal handling
ii  libudev1:amd64              204-5ubuntu20.10   amd64              libudev
shared library
ii  libusb-0.1-4:amd64          2:0.1.12-23.3ubunt amd64              userspace
USB programming library
ii  libustr-1.0-1:amd64         1.0.4-3ubuntu2     amd64              Micro
string library: shared library
ii  libuuid1:amd64              2.20.1-5.1ubuntu20 amd64              Universally
Unique ID library
```

```
ii  libwrap0:amd64              7.6.q-25           amd64              Wietse
Venema's TCP wrappers library
ii  locales                     2.13+git20120306-1 all                common
files for locale support
ii  lockfile-progs              0.1.17             amd64              Programs
for locking and unlocking files and mailboxes
ii  login                       1:4.1.5.1-1ubuntu9 amd64              system
login tools
ii  logrotate                   3.8.7-1ubuntu1     amd64              Log
rotation utility
ii  lsb-base                    4.1+Debian11ubuntu all                Linux
Standard Base 4.1 init script functionality
ii  lsb-release                 4.1+Debian11ubuntu all                Linux
Standard Base version reporting utility
ii  makedev                     2.3.1-93ubuntu1    all                creates
device files in /dev
ii  mawk                        1.3.3-17ubuntu2    amd64              a pattern
scanning and text processing language
ii  mime-support                3.54ubuntu1.1      all                MIME files
'mime.types' & 'mailcap', and support programs
ii  module-init-tools           15-0ubuntu6        all
transitional dummy package (module-init-tools to kmod)
ii  mount                       2.20.1-5.1ubuntu20 amd64              Tools for
mounting and manipulating filesystems
ii  mountall                    2.53               amd64              filesystem
mounting tool
ii  multiarch-support           2.19-0ubuntu6.5    amd64
Transitional package to ensure multiarch compatibility
ii  ncurses-base                5.9+20140118-1ubun all                basic
terminal type definitions
ii  ncurses-bin                 5.9+20140118-1ubun amd64              terminal-
related programs and man pages
ii  net-tools                   1.60-25ubuntu2.1   amd64              The NET-3
networking toolkit
ii  netbase                     5.2                all                Basic
TCP/IP networking system
ii  netcat-openbsd              1.105-7ubuntu1     amd64              TCP/IP
swiss army knife
ii  ntpdate                     1:4.2.6.p5+dfsg-3u amd64              client for
setting system time from NTP servers
ii  openssh-client              1:6.6p1-2ubuntu2   amd64              secure
shell (SSH) client, for secure access to remote machi
ii  openssh-server              1:6.6p1-2ubuntu2   amd64              secure
shell (SSH) server, for secure access from remote mac
ii  openssh-sftp-server         1:6.6p1-2ubuntu2   amd64              secure
shell (SSH) sftp server module, for SFTP access from
ii  passwd                      1:4.1.5.1-1ubuntu9 amd64              change and
administer password and group data
ii  perl                        5.18.2-2ubuntu1    amd64              Larry
Wall's Practical Extraction and Report Language
ii  perl-base                   5.18.2-2ubuntu1    amd64              minimal
Perl system
ii  perl-modules                5.18.2-2ubuntu1    all                Core Perl
modules
ii  plymouth                    0.8.8-0ubuntu17.1  amd64              graphical
boot animation and logger - main package
```

```
ii  procps                       1:3.3.9-1ubuntu2.2 amd64              /proc file
system utilities
ii  python3                      3.4.0-0ubuntu2     amd64              interactive
high-level object-oriented language (default pyt
ii  python3-minimal              3.4.0-0ubuntu2     amd64              minimal
subset of the Python language (default python3 versi
ii  python3.4                    3.4.0-2ubuntu1     amd64              Interactive
high-level object-oriented language (version 3.4
ii  python3.4-minimal            3.4.0-2ubuntu1     amd64              Minimal
subset of the Python language (version 3.4)
ii  readline-common              6.3-4ubuntu2       all                GNU
readline and history libraries, common files
ii  resolvconf                   1.69ubuntu1.1      all                name server
information handler
ii  rsyslog                      7.4.4-1ubuntu2.5   amd64              reliable
system and kernel logging daemon
ii  sed                          4.2.2-4ubuntu1     amd64              The GNU sed
stream editor
ii  sensible-utils               0.0.9              all                Utilities
for sensible alternative selection
ii  ssh                          1:6.6p1-2ubuntu2   all                secure
shell client and server (metapackage)
ii  sudo                         1.8.9p5-1ubuntu1   amd64              Provide
limited super user privileges to specific users
ii  sysv-rc                      2.88dsf-41ubuntu6  all                System-V-
like runlevel change mechanism
ii  sysvinit-utils               2.88dsf-41ubuntu6  amd64              System-V-
like utilities
ii  tar                          1.27.1-1           amd64              GNU version
of the tar archiving utility
ii  tzdata                       2015a-0ubuntu0.14. all                time zone
and daylight-saving time data
ii  ubuntu-keyring               2012.05.19         all                GnuPG keys
of the Ubuntu archive
ii  ubuntu-minimal               1.325              amd64              Minimal
core of Ubuntu
ii  ucf                          3.0027+nmu1        all                Update
Configuration File(s): preserve user changes to confi
ii  udev                         204-5ubuntu20.10   amd64              /dev/ and
hotplug management daemon
ii  upstart                      1.12.1-0ubuntu4.2  amd64              event-based
init daemon
ii  ureadahead                   0.100.0-16         amd64              Read
required files in advance
ii  util-linux                   2.20.1-5.1ubuntu20 amd64
Miscellaneous system utilities
ii  vim                          2:7.4.052-1ubuntu3 amd64              Vi IMproved
- enhanced vi editor
ii  vim-common                   2:7.4.052-1ubuntu3 amd64              Vi IMproved
- Common files
ii  vim-runtime                  2:7.4.052-1ubuntu3 all                Vi IMproved
- Runtime files
ii  vim-tiny                     2:7.4.052-1ubuntu3 amd64              Vi IMproved
- enhanced vi editor - compact version
ii  whiptail                     0.52.15-2ubuntu5   amd64              Displays
user-friendly dialog boxes from shell scripts
```

```
ii  xkb-data                     2.10.1-1ubuntu1    all              X Keyboard
Extension (XKB) configuration data
ii  zlib1g:amd64                1:1.2.8.dfsg-1ubun amd64            compression
library - runtime
```

4. pielikums

'ubuntu' konteinera pakotņu apraksti

- adduser – šī pakotne iekļauj komandas lietotāju izveidošanai un likvidēšanai
- apt (drošības)– priekša '*dpkg*' pakotnei, kas nodrošina '*apt-get*' rīku un vieglāku un drošāku veidu kā pārvaldīt pakotnes
- apt-utils (drošības)– satur dažas '*apt*' rīka programmas
- base-files (būtiska, drošības)– šī pakotne satur *Debian* sistēmas pamata failsistēmas hierarhiju un dažādus svarīgus failus
- bash (būtiska, drošības)– komandu valodas interpretators, kas izpilda no ievadierīcēm vai failiem ievadītas komandas
- bsdutils (būtiska, drošības)– satur minimālo *BSD* rīku skaitu, kas ir nepieciešami, lai startētu *Debian* sistēmu
- busybox-initramfs – apvieno daudzu *UNIX* rīku mazas versijas vienā izpildāmā failā
- bzip2 (drošības) – Datu saspiedējs
- console-setup – nodrošina *Linux* komandrindu un tās konfigurāciju
- coreutils (būtiska, drošības) – satur būtiskus pamata sistēmas rīkus
- cpio (drošības) – rīks arhīvu veidošanai un izjaukšanai, kā arī failu kopēšanai
- cron – šis dēmons ir fona process, kas nodrošina programmu darbību konkrētos laikos, piemēram, katru nedēļu izveidot rezerves kopiju
- dash – komandrindas valodas interpretators, kas izpilda skriptus ātrāk kā '*bash*', un tāpēc ir noklusētā sistēmas čaula *Debian* sistēmās
- debconf – konfigurācijas pārvaldības sistēma *Debian* pakotnēm
- debconf-i18n – nodrošina '*debconf*' internacionalizāciju
- debianutils (būtiska) – nodrošina mazus rīkus, ko primāri izmanto *Debian* pakotņu instalāciju skripti
- dh-python – *Debian* palīgrīki *Python* biblioteku un lietojumprogrammu pakošanai
- diffutils (būtiska) – nodrošina *diff, diff3, sdiff, cmp* programmas. Rāda atšķirības starp salīdzināmiem failiem
- dmsetup (drošības) – minimāla kodola dziņa interpretācija, ka pārvalda sējumus

- dpkg (būtiska, drošības) – *Debian* pakotņu pārvaldības sistēma
- e2fslibs:amd64 – nodrošina bibliotēkas lietotāja programmām, kuras tieši piekļūst paplašinātām failsistēmām
- e2fsprogs (būtiska) – satur programmas *ext* tipa failsistēmu pārvaldīšanai
- eject – izgrūž kompaktdiskus un pārvalda kompaktdisku mainītājus *Linux*
- file (drošības) – pārbauda argumentus, lai tos klasificētu
- findutils (būtiska) – rīki failu atrašanai
- gcc-4.8-base:amd64 – *GNU* kompilatoru kolekcija
- gcc-4.9-base:amd64 – *GNU* kompilatoru kolekcija
- gnupg (drošības) – *GNU* rīks drošai komunikācijai un failu glabāšanai
- gpgv (drošības) – samazināta '*gnupg*' versija, kas tikai pārbauda parakstus
- grep (būtiska) – rīks teksta meklēšanai
- gzip (būtiska) – nodrošina noklusētos *Debian* saspiešanas rīkus
- hostname (būtiska) – nodrošina sistēmas *DNS* vārda, sistēmas vārda un *NIS* domēna vārda pārvaldīšanu
- ifupdown (drošības) – nodrošina rīkus interfeisu iestatīšanai
- init-system-helpers – palīgrīki visām '*init*' sistēmām
- initramfs-tools – nodrošina rīkus pakotu *Linux* kodolu *initramfs* veidošanu un startēšanu
- initramfs-tools-bin – '*initramfs-tools'* izmantotie *binaries*
- initscripts – šīs pakotnes skripti iesāknē sistēmas startēšanu un izslēgšanu
- insserv – programma, ko izmanto lai atjauninātu simbolisko saišu secību
- iproute2 – tīklošanas pārraudzības rīki
- iputils-ping – nodrošina ehotestēšanas komandu
- isc-dhcp-client – *DHCP* klients automātiskai IP adreses iegūšanai
- isc-dhcp-common – visu '*isc-dhcp**' pakotņu kopīgie faili
- kbd – atļauj *Linux* komandrindas ustādīšanu, šrifta mainīšanu un citas klaviatūras iespējas
- keyboard-configuration – klaviatūras priekšrocības sistēmas plašumā
- klibc-utils – satur pret '*klibc´*saitētu programmu apkopojumu
- kmod – kodola moduļu pārvaldības rīki
- language-pack-en – angļu valodas tulkojumu atjauninājumi
- language-pack-en-base – angļu valodas tulkojumi

- less – nodrošina *less* rīku, kas paredzēts vieglākai teksta pārskatīšanai. Failu lapotājs

Visas *lib** pakotnes ir izlaiztas, jo tās parasti nav patstāvīgas pakotnes, bet gan uzstādītas, kā citas pakotnes atkarība.

- locales – nodrošina atbalstu lokalizētām vidēm
- lockfile-progs – iekļauj dažas programmas drošai failu un pastkastu saslēgšanai ar komandrindu.
- login (būtiska, drošības) – nepieciešama, lai varētu pieslēgties un strādāt ar sistēmu
- logrotate (drošības) – nodrošina žurnalēšanas failu vienkāršāku pārvaldību
- lsb-base (būtiska) – standarta pamata sistēma, uz kuru var paļauties trešo pušu lietojumprogrammas
- lsb-release (būtiska) – vienkāršs rīks, kas palīdz noteikt, kāda distributīva tiek izmantota
- makedev – tiek izmantots, lai veidotu ierīču failus, bet modernās sistēmās šī pakotne nav nepieciešama, jo to aizstāj '*udev*'
- mawk – *AWK* programmēšanas valodas interpretators
- mime-support (drošības) – *MIME* faili un atbalsta programmas
- module-init-tools – programmu kopa, kas pārvalda kodola moduļus
- mount (būtiska, drošības) – nodrošina *mount, umount, swapon, swapoff, losetup* komandas
- mountall (drošības) – montē failsistēmas
- multiarch-support (drošības) – nodrošina vairāku arhitektūru savietojamību
- ncurses-base (būtiska) – satur failus galiekārtas atbalstam
- ncurses-bin (būtiska) – satur programmas galiekārtas atbalstam
- net-tools – satur svarīgus rīkus kodola tīkla pārvaldīšanai, piemēram, *ifconfig*
- netbase – nodrošina nepieciešamo infrastruktūru *TCP/IP* tīklošanai
- netcat-openbsd – vienkāršs rīks, kas nodrošina datu lasīšanu un rakstīšanu tīkla savienojumos
- ntpdate (drošības) – sinhronizē pulksteni, lai uzturētu precīzu laiku
- openssh-client – *ssh* savienojuma klients
- openssh-server – *ssh* savienojuma serveris
- openssh-sftp-server – failu sūtīšanas serveris
- passwd – satur programmas paroļu un grupu uzturēšanai un pārvaldīšanai

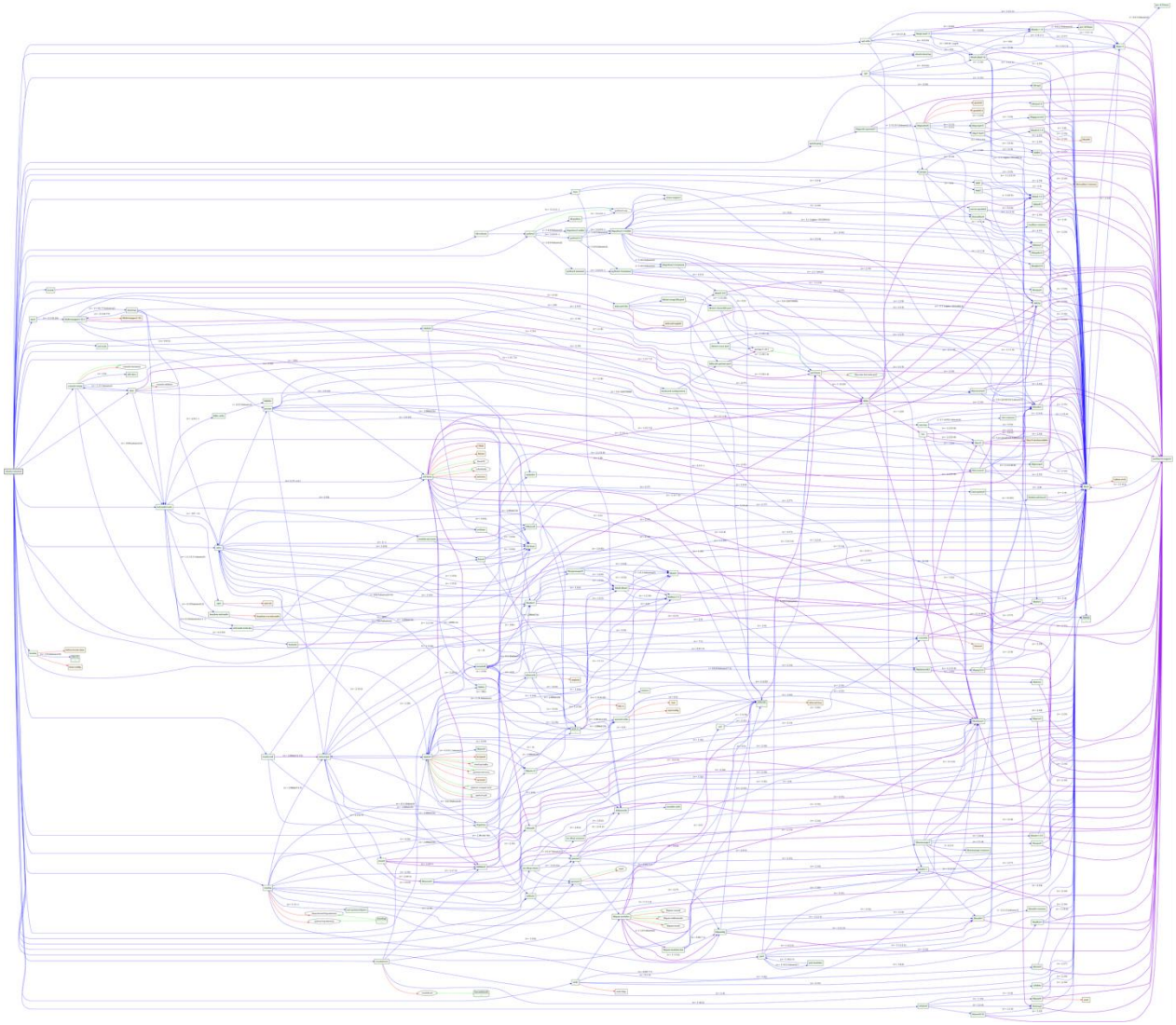
- perl – optimizēts patvaļīgu teksta failu lasīšanai un sistēmas administrācijai
- perl-base (būtiska) – nodrošina *Perl* interpretatoru un vienkāršu uzdevumu veikšanai nepieciešamās darbības bibliotekas
- perl-modules – *Perl* moduļi
- plymouth – nodrošina grafisku startēšanās animāciju, kamēr startējas sistēma
- procps – rīki, lai pārskatītu */proc* failsistēmu
- python3 – programmēšanas valoda
- python3-minimal – *Python* interpretators un daži būtiski moduļi
- python3.4 - programmēšanas valoda
- python3.4-minimal - *Python* interpretators un daži būtiski moduļi
- readline-common – bibliotēka, kas nodrošina atblastu programmām, kurām ir nepieciešama komandrindas saskarne
- resolvconf – nodrošina sistēmas informācijas par vārdu serveriem atjaunināšanu
- rsyslog (drošības) – veido žurnālus
- sed (būtiska) – ievaddatu apstrāde un izvade
- sensible-utils – nodrošina mazus rīkus, kas nosaka, kādus pārlūkus, redaktorus vai lapotājus izmantot
- ssh – metapaka, kas nodrošina *Openssh* servera un klienta uzstādīšanu
- sudo – ļauj sistēmas administrātoram piešķirt *sudo* tiesības lietotājiem un žurnalē *root* darbības
- sysv-rc – nodrošina startēšanas, izslēgšanas un cita veida atbalstu
- sysvinit-utils – satur svarīgus *System-V* rīkus
- tar (būtiska) – nodrošina failu arhivāciju
- tzdata (drošības) – satur nepieciešamus datus lokālās laika zonas implementācijai
- ubuntu-keyring – satur laidiena failu digitālos parakstus
- ubuntu-minimal – metapaka, kas nodrošina un ir akarīga no visām pakām minimālā *Ubuntu* sistēmā
- ucf – nodrošina konfigurācijas failu pārvaldīšanu failiem, kuri nav atzīmēti kā konfigurācijas faili
- udev – rīku un dēmona kopa no kodola saņemtu notikumu pārvaldīšanā lietotāja telpā
- upstart – pārvalda uzdevumu un servisu startēšanu, uzraudzīšanu un apturēšanu

- ureadhead – izmanto startēšanas laikā, lai ielasītu failus vēl pirms tie ir vajadzīgi
- util-linux (būtiska) – satur svarīgus sistēmas uzturēšanas rīkus
- vim – *UNIX* redaktora *Vi* versija
- vim-common – Satur visu *vim* versiju bez grafiskās vides kopīgos failus
- vim-runtime – satur *vim* darbības failus
- vim-tiny – satur minimālu *vim* versiju
- whiptail – nodrošina dažādu dialogu logu attēlošanu čaulas skriptos
- xkb-data – nodrošina dažādu klaviatūras izklāju izvēles iespēju, izmantojo grafisku saskarni
- zlib1g:amd64 – failu saspiedēju atbalsts

Visu šo pakotņu apraksti tika iegūti un tulkoti no 3 avotiem. Pakotnēm, kuras ir pieejamas ar *Ubuntu Lucid* laidienā apraksts ir pieejams mājas lapā http://packages.ubuntu.com/lucid/'pakotnes_nosaukums' un pakotnēm, kas ir piejamas tikai laidienos sākot ar *Ubuntu Trusty* apraksti ir pieejami mājas lapā http://packages.ubuntu.com/trusty/'pakotnes_nosaukums', un pāris pakotnņu apraksti tika atrasti mājas lapā https://launchpad.net/ubuntu/trusty/+package/'pakotnes_nosaukums'.

5. pielikums

‘ubuntu-minimal’ pakotnes atkarību grafa bilde



Kvalifikācijas darbs „***Linux un Docker konteineri kā sistēmas virtualizācijas rīki***” izstrādāts Latvijas Universitātes Datorikas fakultātē.

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

Autors: ***Ģirts Dālbergs*** **_____ __.05.2015**.

Rekomendēju darbu aizstāvēšanai

Darba vadītājs: ***Dr.sc.comp. Leo Trukšāns*** **_____ __.05.2015.**

Recenzents: _____ **__.05.2015**

Darbs iesniegts 01.06.2015.

Kvalifikācijas darbu pārbaudījumu komisijas sekretārs: *Darja Solodovņikova*_____

Darbs aizstāvēts kvalifikācijas darbu pārbaudījuma komisijas sēdē

___.06.2015. prot. Nr. _____

Komisijas sekretārs(-e): _____