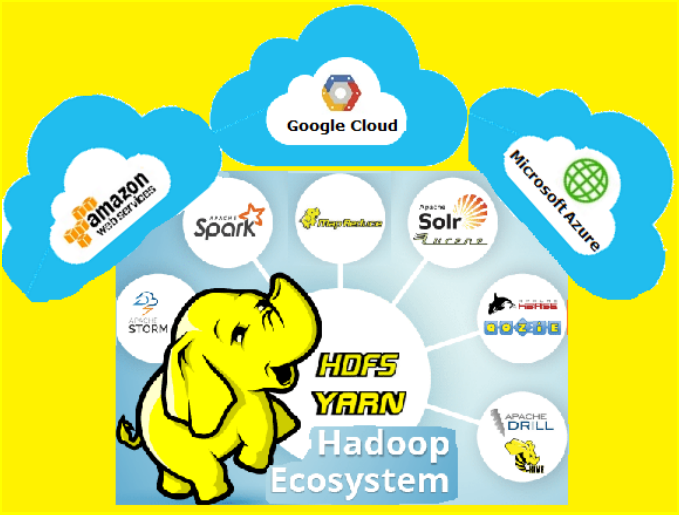
გია სურგულაძე,
დავით გულუა,
ბექა კახელი

პროგრამული აპლიკაციების აგება ვირტუალიზაციის პირობებში



„სტუ-ს IT კონსალტინგის ცენტრი”

გია სურგულაძე, დავით გულუა,
ბექა კახელი

პროგრამული აპლიკაციების აგება ვირტუალიზაციის პირობებში

დამტკიცებულია:

სტუ-ს „IT კონსალტინგის სამეცნიერო ცენტრის“ სარედაქციო კოლეგიის მიერ

თბილისი,
2019

უაკ 004.5

განხილულია კომპიუტერული სისტემების სფეროში პროგრამული და აპარატურული ვირტუალიზაციის თანამედროვე ტექნოლოგიები, მათი კლასიფიკაცია და პრაქტიკული გამოყენების ძირითადი მიმართულებები, პროგრამული აპლიკაციების. დეველოპმენტის საბაზო პრინციპები, მეთოდები და შესაბამისი ინსტრუმენტული საშუალებები. თეორიული კვლევის და პრაქტიკული ექსპერიმენტის საილუსტრაციო მაგალითები შესრულებულია Windows და Linux ოპერაციული სისტემების, VMWare Server ვირტუალური ინფრასტრუქტურის, მაიკროსოფთის Hyper-V ვირტუალური მანქანის, Apache Hadoop და MongoDb დიდი განაწილებული სისტემების, ღრუბლოვანი ინფრასტრუქტურისთვის Raspberry Pi ინსტრუმენტის გამოყენებით და სხვ. *მონოგრაფია* გამიზნულია ინფორმატიკის სფეროს საგანმანათლებლო პროგრამის სამივე საფეხურის სტუდენტებისა და სპეციალისტებისათვის, რომლებიც დაინტერესებულნი არიან კომპიუტერულ სისტემებში პროგრამული და აპარატურული ვირტუალიზაციის საკითხების შესწავლით, დამუშავებითა და გამოყენებით.

რეცენზენტები:

- პროფ. ნ. ამილახვარი (საქართველოს ტექნიკური უნივერსიტეტი)
- ასოც.პროფ. თ. კივილაძე (საპატრიარქოს ქართული უნივერსიტეტი)

რედკოლეგია:

ა. ფრანგიშვილი (თავმჯდომარე), მ. ახობაძე, გ. გოგიჩაიშვილი, ზ. ბოსიკაშვილი, ე. თურქია, რ. კაკუბავა, ნ. ლომინაძე, ჰ. მელაძე, თ. ობგაძე, გ. სურგულაძე (რედაქტორი), გ. ჩაჩანიძე, ა. ცინცაძე, ზ. წვერაიძე

ISBN 978-9941-8-0627-8

Gia Surguladze, David Gulua,
Beka Kakheli

DEVELOPMENT OF SOFTWARE APPLICATIONS USING VIRTUALIZATION

We consider modern technologies of software and hardware virtualization in the field of computer systems, their classification and the main directions of practical use, the basic principles and methods of developing application software and related tools. Illustrative examples of theoretical studies and practical experiments were performed on the basis of Windows and Linux operating systems, VMWare server virtual infrastructure, Hyper-V virtual machine, Apache Hadoop and MongoDb for large distributed ecosystems, Raspberry Pi tool for cloud infrastructure, etc. This book is intended for bachelor, master, doctoral students of Software Engineering of Management Information Systems as well as for readers interested in the topics of software development in a virtualization environment.

ISBN 978-9941-8-0627-8

ავტორთა შესახებ:

გია სურგულაძე - ტექნიკის მეცნიერებათა დოქტორი, ინფორმატიზაციის საერთაშორისო აკადემიის ნამდვილი წევრი (1994 წლიდან, გაეროსთან არსებული IIA), სტუ-ს ინფორმატიკის ფაკულტეტის პროფესორი, „მართვის ავტომატიზებული სისტემების (პროგრამული ინჟინერიის)“ დეპარტამენტის უფროსი. 77 წიგნის (16 მონოგრაფია, 61 სახელმძღვანელო), 55 ელ-სახელმძღვანელოს და 300-მდე სამეცნიერო ნაშრომის ავტორი მართვის საინფორმაციო სისტემების, პროგრამული ინჟინერიის, მონაცემთა მენეჯმენტის, დაპროგრამების ჰიბრიდული ტექნოლოგიების, იმიტაციური მოდელირების, პეტრის ფერადი ქსელების, რიგების თეორიის და სხვ. სფეროში.

დავით გულუა - ტექნიკის მეცნიერებათა კანდიდატი (სტუ, 2004), დ. აღმაშენებლის სახ. ეროვნული თავდაცვის აკადემიის ასოც. პროფესორი. გერმანიის DAAD-ის სტიპენდიანტი (2002-2003), ბერლინის ჰუმბოლდტის უნივერსიტეტის სერვერების მენეჯერი (2004-2008). 4 წიგნის და 30 სამეცნიერო ნაშრომის ავტორი მართვის საინფორმაციო სისტემების, პეტრის ქსელების, ვებ-ტექნოლოგიების, კომპიუტ_სისტემების და ქსელების ადმინისტრირების სფეროში. კითხულობს ლექციებს თსუ, სტუ, ბიზნეს-ტექნოლოგიებისა და შავი ზღვის უნივერსიტეტებში.

ბექა კახელი - ინფორმატიკის აკად. დოქტორი (სტუ, 02.2019). გადაუდებელი დახმარების ოპერატიული მართვის ცენტრის (112) მონაცემთა ბაზების და VOIP სისტემის სერტიფიცირებული ადმინისტრატორი, Linux/Unix ინჟინერი 7+ წლიანი გამოცდილებით ინფორმაციული ტექნოლოგიების სფეროში. უნარ-ჩვევები - მონცემთა ბაზები: DB2, Oracle, MongoDB, MsSQL MariaDB PostgreSQL; სისტემა VoIP: Asterisk, Media gateway, SIP; ოპერაციული სისტემები: CentOS, Red hat, SUSE, AIX, Windows; ვირტუალიზაცია: VMware, Docker, FreeNAS; პროგრამირება: Python, Bash/Shell, OOP, SQL, Git; მონიტორინგი: WhatsUP Gold, TCP dump, Wireshark და სხვ.

შინაარსი

შესავალი

ინფორმატიკის (კომპიუტინგის) სფეროში ტერმინი „ვირტუალიზაცია“ (Virtualization) ნიშნავს რაიმე ვირტუალურის (ფაქტობრივად არარეალურის) შექმნას, მათ შორის ვირტუალური კომპიუტერების აპარატურული პლატფორმების, მონაცემთა შენახვისა და გადაცემის მოწყობილობებისა და გამოთვლითი ქსელების რესურსების ჩათვლით [1-4].

ვირტუალიზაციის კონცეფცია საკმაოდ ძველია (XIII საუკუნე - საფრანგეთი, ინგლისი) [5,6]. გამოთვლითი ტექნიკის გამოჩენამ ვირტუალობის ტერმინს დამატებითი თვისება შესძინა, კერძოდ, „სინამდვილეში არარსებული, მაგრამ შექმნილი პროგრამული უზრუნველყოფის საფუძველზე“ [6].

განსაკუთრები დიდი მნიშვნელობა მან ინფორმაციული ტექნოლოგიების განვითარების ბოლო ოცწლეულში შეიძინა, როცა აქტიურად დაიწყო განვითარება ღრუბლოვანმა ტექნოლოგიებმა ვირტუალიზაციისა და კლასტერიზაციის თანამედროვე საშუალებების გამოყენებით, დიდ მონაცემთა ეკოსისტემებმა და მონაცემთა დამუშავების ცენტრების ეფექტური არქიტექტურების კვლევამ.

წინამდებარე ნაშრომში შემოთავაზებულია ავტორების მიერ წლების განმავლ;ობაში საქართველოს ტექნიკურ უნივერსიტეტსა და გერმანიის ჰუმბოლდტის უნივერსიტეტში მოღვაწეობის პერიოდში შესრულებული სამუშაოების შედეგები, აგრეთვე მათი დოქტორანტურაში სწავლისა და საგანგებო სიტუაციების მართვის ცენტრში „112“ პრაქტიკული მუშაობის და ახალი ეკოსისტემების დანერგვის შედეგები [7-11].

წიგნის ***პირველ თავში*** წარმოდგენილია ვირტუალიზაციის პროცესის არსი და ძირითადი ცნებები, მათი კლასიფიკაცია სახეების მიხედვით და დანიშნულება. საწყისი მაგალითის სახით განხილულია VMWare Server-ის ვირტუალური ინფრასტრუქტურა, მისი ინსტალირების და მუშაობის დეტალები, აგრეთვე ვირტუალური მანქანა Hyper-V და სერვერთა კლასტერი.

მეორე თავში აღწერილია გამოყენებული თანამედროვე ტექნოლოგიები. მიმოხილულია GPL ლიცენზიაზე დამყარებული და თავისუფლად გავრცელებადი Linux-ის ოპერაციული სისტემა [12]. ასევე მიმოხილულია მონაცემების შენახვის ტექნიკური და ლოგიკური მეთოდოლოგიები, დისკური მასივები და მათი კომბინირებული ტექნოლოგიები. დისკური მასივების დახმარებით შესაძლებელია ინფორმაციის ფიზიკური დისკური მატარებლების მართვა, გაერთიანება, ვირტუალური დისკური მასივების შექმნა და მონაცემების ჩაწერა-წაკითხვის, საიმედოობის მართვა. ამ თავშივე მიმოხილულია მონაცემების შენახვის სისტემები, რომლებიც იყენებს დისკური მასივის პრინციპებს და ინფორმაციის შენახვის საიმედოობისა და წარმადობის გარანტიას იძლევა.

წიგნის **მესამე თავში** აღწერილია კვლევის მეთოდოლოგია და კვლევის მოსალოდნელი შედეგები. ასევე მიმოხილულია ინფორმაციის შენახვა-დამუშავების პრობლემური სფერო, ის რომ მონაცემები ექსპონენციალურად იზრდება და მისი შენახვა-დამუშავების ამოცანა დღითიდღე უფრო რთულ ამოცანად ფორმირდება. ტრადიციული მეთოდებით ამ ინფორმაციის რეალურ დროში დამუშავება კი წარმოუდგენლად დიდ რესურსებს უკავშირდება. პრობლემის გადაჭრის გზად ნაშრომში შემოთავაზებულია ჰორიზონტალური ზრდის ტექნოლოგიები, ინფორმაციის განაწილებულ გარემოში დამუშავების ტექნოლოგიები და მათი უპირატესობები. ასევე ნაშრომის ფარგლებში მოძიებულია დაბალი ღირებულების ტექნიკა, რომლის დახმარებითაც შესაძლებელია განაწილებული სისტემების პროექტირება და, საჭიროების შემთხვევაში, მისი მასშტაბირება. განხილულია ოპერაციული სისტემის მახასიათებლები და რეკომენდაციების სახით ჩამოყალიბებულია ის პარამეტრები, რასაც უნდა აკმაყოფილებდეს როგორც აპარატურა, ასევე ოპერაციული სისტემა, პროგრამული კოდი და მონაცემთა ბაზები. განხილული და აღწერილია მიკროსერვისების

მართვის პლატფორმები და შემუშავებულია განაწილებული გარემოს ეკოსისტემის არქიტექტურა.

ნაშრომის **მეოთხე თავში** გადმოცემულია ოპერაციული სისტემის გამართვის დეტალები, შემუშავებულია რეკომენდაციები მიზნად დასახული ამოცანის შესასრულებლად. დეტალურად არის აღწერილი აღნიშნული ეკოსისტემის არქიტექტურის ყველა ეტაპი - დაწყებული ვირტუალური მანქანის შექმნითა და მისი მახასია-თებლების ფორმირებით, ოპერაციული სისტემის პარამეტრების გამართვა, საჭირო პაკეტების მოძიება ინსტალაცია, ქსელისა და უსაფრთხოების პარამეტრების რეკომენდაციების მიხედვით გასწორება. ასევე აღწერილია გამზადებულ პლატფორმაზე მონაცემთა ბაზების ინსტალაცია, კონფიგურაცია და გაშვება. ნაშრომის შემაჯამებელ ნაწილში აღწერილია წიგნის ფარგლებში შემუშავებული პროგრამული უზრუნველყოფა, რომელიც შესრუ-ლებულია დაპროგრამების Python ენის საფუძველზე და იყენებს ყველა იმ თანამედროვე ტექნოლოგიებსა და მიდგომებს, რომლებიც განხილულია წიგნის წინა თავებში.

მეხუთე თავში განხილულია ღრუბლოვანი ტექნოლოგიების სერვისებისა და ვირტუალიზაციის პრინციპების გამოყენების პრაქტიკები ისეთი ცნობილი ფირმების საფუძველზე, როგორებიცაა Amazon, Micrisoft Azure და Google Cloud. წარმოდგენილია ვირტუალიზაციის მაგალითები სერვისების, ვირტუალური მანქანებისა და ვირტუალური სერვერების სახით.

მონოგრაფიაში ავტორთა ყურადღება გამახვილებულია განაწილებული კომპიუტერული სისტემების დაპროექტებისა და პროგრამული დეველოპმენტის საკითხებზე ვირტუალიზაციის კონცეფციის პირობებში, რათა გამოყენებითი პროგრამული აპლიკაციების შექმნისა და ტესტირების პროცესები იყოს ბევრად ეფექტური და ხარისხიანი.

მადლობელი ვიქნებით ჩვენი მკითხველის საქმიანი შენიშვ-ნებისა და რეკომენდაციებისათვის.

I თავი

ვირტუალიზაცია, კლასიფიკაცია და საფუძვლები

1.1. ვირტუალიზაციის არსი და ძირითადი ცნებები

ვირტუალიზაციის თეორიული და პრაქტიკული ასპექტები ჯერ კიდევ გასული საუკუნის 70-იან წლებში იქნა დამუშავებული **IBM** ფირმის სუპერმანქანებისთვის (მაგალითად, IBM/360 მაინფრეიმი და სხვ.). სისტემის ყოველ მომხმარებელს მთლიანი სერვერიდან გამოეყოფოდა რესურსების საკუთარი ნაკრები და უქმნიდა გამოყოფილ გამოთვლით სისტემასთან მუშაობის ილუზიას [3].

2000-იან წლებში იდეა ხელახლა გახდა აქტუალური, განსაკუთრებით სერვერულ სისტემებში, თუმცა ზოგადად ტერმინი „ვირტუალიზაცია“ ბევრად ფართო შინაარს ატარებს.

ვირტუალურად შეიძლება ვაქციოთ გამოთვლითი სისტემის პრაქტიკულად ყველა კომპონენტი, როგორც აპარატული, ასევე პროგრამული (მაგალითად, ოპტიკური დისკამძრავები, ქსელური ინტერფეისები, ოპერატიული მეხსიერება, ოპერაციული სისტემები და ა.შ.).

სერვერის ვირტუალიზაცია ერთ ფიზიკურ სერვერზე რამდენიმე ვირტუალური სერვერის გაშვებას გულისხმობს.

ვირტუალური მანქანები ან სერვერები წარმოადგენს პროგრამებს, რომლებიც ეშვება ფიზიკური, ე.წ. „მასპინძელი" ოპერაციული სისტემის ფარგლებში (ჰოსტ-სერვერი). თავის მხრივ, ყოველი ვირტუალური მანქანა დამოუკიდებელი ოპერაციული სისტემაა, საკუთარი პროგრამებით და სერვისებით.

ვირტულიზაციის შედეგად მიღებული ეკონომიკური ეფექტი უდავოა. დიდი სერვერული სისტემა (მონაცემთა დამუშავების ცენტრი) ვრცელ ფართზეა განთავსებული და ელექტროენერგიის დიდ რაოდენობას მოიხმარს, განსაკუთრებით მაშინ, როცა ცენტრს

გაცივების სპეციალური სისტემები და დამატებითი ინფრა-სტრუქტურა ემსახურება. ვირტუალიზაციის გამოყენება რამდენიმე ფიზიკური სერვერის ერთ მძლავრ სერვერზე „შეფუთვის“ საშუალებას იძლევა, რითაც იზოგება ადგილი და მცირდება ელექტროენერგიის ხარჯი. ამასთან მცირდება აპარატურაზე საერთო დანახარჯებიც ნაკლები რაოდენობის სერვერთა არსებობის გამო.

შეიძლიება ჩამოვთვალოთ ვირტუალური სისტემების სხვა უპირატესობებიც ფიზიკურ სისტემებთან შედარებით, განსაკუთ-რებით კომპიუტერული ქსელების სერვერული ინფრასტრუქტუ-რის აგებისა და მართვის თვალსაზრისით:

- ერთ გამოთვლით სისტემაში ერთზე მეტი ოპერაციული სისტემის პარალელური მუშაობა;
- კრიტიკული რესურსების (პროცესორის დრო, ოპერატიული და გარე მეხსიერება) ოპტიმალური განაწილება ვირტუალურ მანქანებს შორის;
- ოპერაციული სისტემების სწრაფი გადატანა ფიზიკურ ჰოსტებს შორის. ოპერაციული სისტემების ინსტალაციისა და კონფიგურირების დროითი დანახარჯების შემცირება წინასწარ მომზადებული ვირტუალური სისტემების (იმიჯების) ხარჯზე.

გვერდს ვერ ავუვლით ძირითად ნაკლოვანებებსაც, რაც ვირტუალურ სისტემებს ახასიათებს:

- სერვერულ რესურსებზე მაღალი მოთხოვნები - ვირტუალუ-რი მანქანა, ფაქტობრივად, დამოუკიდებელი ოპერაციული სისტემაა და გამოთვლითი რესურსებიც პრაქტიკულად ფიზიკური მანქანის მასშტაბებით სჭირდება, ანუ ბევრად მეტი, ვიდრე ამას სტანდარტული პროგრამები მოითხოვს;
- დამოკიდებულება ერთ ფიზიკურ სერვერზე, რომლის მწყობრიდან გამოსვლაც მასზე არსებული ყველა ვირტუალური მანქანის მწყობრიდან გამოსვლის ტოლფასია.

ბოლო პრობლემის აღმოფხვრა ვირტუალიზაციისა და კლასტერული არქიტექტურის ურთიერთშერწყმით ხდება შესაძლებელი, რაც მოცემული ნაშრომის ერთერთ ძირითად საგანს წარმოადგენს.

ვირტუალიზაციის პროცესი საკმაოდ კომპლექსურია და რამდენიმე ამოცანის გადაჭრას ითხოვს:

იზოლირება ვირტუალიზაციის მნიშვნელოვანი კომპონენტია და ვირტუალური სისტემების სრულ ურთიერთდამოუკიდებლობაში მდგომარეობს. ერთი ვირტუალური მანქანის მწყობრიდან გამოსვლას მეორის მუშაობაზე არავითარი გავლენის მოხდენა არ შეუძლია. კერძოდ, ფიზიკურ სერვერზე განთავსებულ ვირტუალურ მანქანებს შორის მონაცემთა არანაირი ერთიანი სივრცე არ არსებობს. მათ შორის ინფორმაციის ნებისმიერი ტრანსფერი ჩვეულებრივი ქსელური ინტერფეისების გავლით ხორციელდება;

ინკაპსულაცია ნიშნავს მთლიანი ვირტუალური სისტემის ერთ (ან მეტ) ჩვეულებრივ კომპიუტერულ ფაილში ინტეგრირებას. ამგვარი სისტემები მეტად მოქნილია ფიზიკურ სისტემებს შორის გადატანის, ასლის შექმნის (კოპირების) თუ არქივირების ოპერაციების შესრულებისას.

გამოთვლითი მანქანების ვირტუალიზაციისას განიხილავენ 2 მთავარ კომპონენტს: მასპინძელ (Host) და სტუმარ (Guest) სისტემებს, ამასთან მასპინძელი ერთია და როგორც წესი, ფიზიკურ გამოთვლით სისტემას წარმოადგენს, ხოლო სტუმარ-სისტემები მასზე განთავსებული 1 ან მეტი ვირტუალური მანქანაა.

ცხადია, ამგვარი მარტივი სტრუქტურა გამოსადეგია მხოლოდ დესკტოპ-სისტემებისთვის, სადაც მომხმარებელს რამდენიმე ოპერაციული სისტემის ერთდროული გამოყენება სჭირდება. იგი ასაგებად საკმაოდ ადვილია, თუმცა რამდენიმე ამოცანის გადაჭრას მაინც საჭიროებს. ეს ზოგადი ამოცანები ნებისმიერი ვირტუალური ინფრასტრუქტურის ფარგლებში წარმოიშობა და მათი განხილვა აუცილებელია:

- *სტუმარ-სისტემათა შაბლონების აგება* - არის პროცედურა, რომელსაც ვირტუალური სისტემის ასაგებად საჭირო დროითი დანახარჯების მნიშვნელოვნად შემცირება ძალუძს. ყოველი სტუმარ-ოპერაციული სისტემისათვის (*Windows, LINUX, Mac OS X* და სხვ.) იქმნება საბაზისო კონფიგურაცია, რომელიც შემდეგ ყოველი კონკრეტული მოთხოვნის მიხედვით ფართოვდება;
- *ვირტუალური სისტემების კოპირება ან კლონირება* – კოპირების ოპერაცია ვირტუალური სისტემის ჩვეულებრივი ასლის აგებას გულისხმობს და ნაკლებად ეფექტურია, რადგან მასპინძელ-სისტემის გარე მეხსიერების სწრაფ გავსებას იწვევს. ბევრად უფრო მოქნილი კლონირების მექანიზმი მომხმარებელს არჩევანს უტოვებს: სრული კლონირებისას ვირტუალური მანქანის სრული და დამოუკიდებელი ასლი იქმნება, ხოლო ბმული (ლინკირებული) კლონირება საწყის ვირტუალურ მანქანას ეყრდნობა (მისი მოდიფიკაციაა) და მეხსიერების მინიმალურ ხარჯვას სჭერდება;
- *სკრინშოტების შექმნა* – სკრინშოტი ვირტუალური სისტემის მიმდინარე მდგომარეობას ეწოდება და საჭიროების მიხედვით სისტემის მოცემული მდგომარეობის აღდგენას განაპირობებს;
- *ფიზიკური მანქანების ვირტუალიზაცია* – გულისხმობს არსებული ფიზიკური სისტემების ვირტუალურ ინფრასტრუქტურაში მიგრაციას ვირტუალური მანქანების სახით;
- *რესურსების განაწილება მასპინძელ და სტუმარ სისტემებს შორის* – გულისხმობს მასპინძელ-სისტემის რესურსების (პროცესორული სიმძლავრეები, ოპერატიული და გარე მეხსიერება, დისკამძრავები, ქსელური ინტერფეისები) სტუმარ-სისტემებზე გადანაწილებას. ეხება პირველ რიგში ქსელურ რესურსებს (სისტემის უნიკალური ქსელური *MAC*-მისამართი, *IP*-მისამართი).

ზემოთ აღწერილი ამოცანების პრაქტიკული რეალიზაციის პროცესი სადღეისოდ საკმაოდ შორს არის წასული. ვირტუალიზაციის პროგრამული პროდუქტები (ფირმა VMWare-ს ESX Server, vSphere და Workstation; ფირმა Citrix-ის XenApp, ფირმა

Microsoft-ის Hyper-V, Virtual Server, Virtual PC, ფირმა Oracle-ის VirtualBox) მოიცავს ინსტრუმენტების ფართო ნაკრებს სხვადასხვა (მათ შორის კორპორაციული) მასშტაბის კომპიუტერული ქსელების სერვერულ სისტემათა ვირტუალიზაციისთვის.

საყურადღებოა, რომ ამ ინსტრუმენტების ნაწილი არსებული ფიზიკური ინფრასტრუქტურის უმტკივნეულოდ „გავირტუალების" საქმეს ემსახურება, რაც ორგანიზაციებს საშუალებას აძლევს თავიანთი სერვერული ინფრასტრუქტურის მოდერნიზაცია და ვირტუალურ რელსებზე გადაყვანა ძირითადი საწარმოო პროცესის შეწყვეტის გარეშე შეასრულოს.

საილუსტრაციოდ განვიხილოთ ვირტუალიზაციის ერთერთი სრული პროგრამული პაკეტი (ფირმა VMWare) და მისი მთავარი კომპონენტები:

- *ჰაიპერვაიზორი (Hypervisor) VMware ESX (ESXi) Server* - ჰოსტ-სისტემის მმართველი პროგრამა (იხილეთ ქვემოთ);
- *VMWare Converter* - ფიზიკური გამოთვლითი სისტემების ვირტუალიზაციის ან ვირტუალური მანქანების სხვადასხვა ფორმატებში კონვერტირების ინსტრუმენტი;
- *VMotion* – ვირტუალური მანქანების მიგრაცია სერვერებს შორის მათი გამორთვის გარეშე;
- *Virtual SMP* - სტუმარ-სისტემის 4-ზე მეტ პროცესორთან მუშაობის უზრუნველყოფა.

ბოლო წლებში ვირტუალიზაცია სერვერულ აპარატურასაც შეეხო. x86-არქიტექტურაზე მომუშავე პროცესორების ახალი თაობა (Intel, AMD) ვირტუალიზაციის აპარატურული მხარდაჭერითაა აღჭურვილი, რაც ერთიორად ამაღლებს ვირტუალური ოპერაციული სისტემების მუშაობის ეფექტურობას. სწორედ ამგვარი პროცესორებით აღჭურვილ სერვერულ სისტემებშია ყველაზე ეფექტური „აბსოლუტური ვირტუალიზაციის" განხორციელება, რომელიც ჰოსტ-მანქანად არა სტანდარტული ოპერაციული სისტემების, არამედ ე.წ. ჰაიპერვაიზორების ()

გამოყენებას გულისხმობს. ჰაიპერვაიზორი სპეციალურად ვირტუალური მანქანების მომსახურებისთვის შექმნილი მინი ოპერაციული სისტემაა. ვირტუალურ "მასპინძლად" სტანდარტული ოპერაციული სისტემების ჰაიპერვაიზორებით ჩანაცვლება მნიშვნელოვნად ზოგავს სერვერის რესურსებს. მიმდინარე ეტაპზე ყველაზე გავრცელებული ჰაიპერვაიზორებია VMware ESX Server, Microsoft Hyper-V და უფასო პროგრამა Xen.

1.2. ვირტუალური ინფრასტრუქტურა *MWare Server*

1.2.1. ინსტალაცია

მთავარი პროგრამის ინსტალაციის შემდგომ სტუმარ სისტემის მუშაობის პროცესში უნდა დაყენდეს VMTools სტუმარ სისტემაში თაგვის და მონიტორის მუშაობის გასაუმჯობესებლად. აღნიშნული პროგრამა ინსტალაციისას გვთავაზობს Windows-ის სისტემური Hardware Acceleration ფუნქციის აქტივაციას და მაქსიმუმზე დაყენებას: Display -> Settings -> Advanced -> troubleshooting -> Hardware Accelleration.

ვირტუალურ მანქანების მართვა ხდება სპეციალური მონიტორ-პროგრამით, სახელად VMM (Virtual Machine Monitor), რომელიც განაგებს მანქანათშორის რესურსების განაწილებას.

1.2.2. მუშაობის დეტალები: გასტ-სისტემების კოპირება, კლონირება, იმპორტი

სისტემის ერთი ჰოსტ-მანქანიდან მეორეზე უბრალოდ კოპირებისას პროგრამა სვამს კითხვას: შეინარჩუნოს გასტ-სისტემის უნიკალური იდენტიფიკატორი (**SID**) თუ შექმნას ახალი? რეკომენდებულია: სისტემის კოპირებისას ახალი იდენტიფი-კატორის შექმნა, ხოლო გადატანისას კი – ძველის შენარჩუნება.

კლონირებისას არ იცვლება ჰოსტ-სისტემის **IP**-მისამართი და ქსელური სახელი, რაც ვირტუალურ მანქანების ერთდროული

ჩართვისას IP- და ქსელურ კონფლიქტებს აღძრავს. ორივე მანუალურ კონფიგურირებას მოითხოვს.

პროგრამა ახერხებს შემდეგ პროდუქტებში შექმნილი იმიჯების იმპორტირებას: MVMWare Workstation (Server), Microsoft Virtual PC (Server), Norton Ghost, Symantec Live State Recovery

1.2.3. კონფიგურაციული ცვლილებები (Edit Virtual Machine Settings):

Hardware – „პერიფერიულ მოწყობილობათა" (Removable Devices) სიაში შედიან Audio, Floppy, CD-ROM, Ethernet, USB-Devices. ყოველი საკონფიგურაციო ცვლილებისას ვირტუალური მანქანა გათიშული უნდა იყოს, ხოლო შემდეგ საჭიროა მოწყობილობის "მიერთება" ვირტუალურ მანქანასთან (მენიუ VM -> <პერიფერიული მოწყობილობა> -> Connect)

Memory – გასტ-სისტემის ოპერატიული მეხსიერების ზომის ცვლა Hard Disk – "ხისტი დისკი" დეფრაგმენტაციის პროგრამითა და SCSI-კონფიგურაციით.

CD ROM Device – „კომპაქტ-დისკი". Host-Client გადამრთველით შეიძლება ორიდან ერთერთი ფიზიკური კომპაქტ-დისკის გამოტანა: ან ჰოსტ-სისტემის (ubtest01), ან დაშორებულ კლიენტისა, რომლიდანაც ჰოსტ-სისტემა და ვირტუალური მანქანები იმართება (Citrix4).

- ოფცია "Connected" საჭიროა გასტ-სისტემაზე დისკამძრავის მანუალურ გააქტიურებისთვის;
- ოფცია "Connect at powered on" – მოწყობილობას ვირტუალურ მანქანის ჩართვისტანავე გააქტიურებს;
- ოფცია "Connect for Exclusive Use" ნიშნავს, რომ დისკამძრავს იყენებს ერთი და მხოლოდ ერთი გასტ-სისტემა, რომელიც პირველი დაეპატრონება. ოფციის გასააქტიურებლად წინასწარ საჭიროა კომპაქტ-დისკამძრავის გამოერთება ვირტუალურ მანქანიდან;
- ოფცია "ISO Image" ფიზიკურ დისკს იმიჯით ჩაანაცვლებს;

Floppy Disk – ფლოპი-დისკი ან მისი იმიჯი;

Ethernet – ქსელის პარამეტრები. ქსელში გასვლის 4 ვარიანტი;

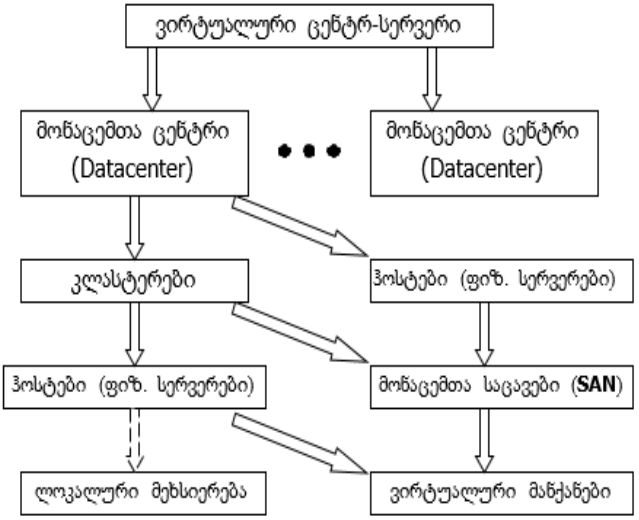
USB Controller – მიკროსქემა (მოწყობილობა) მონაცემების გაცვლისათვის კომპიტერსა და USB შორის (ინტერფეისი);

Processors – პროცესორთა რაოდენობის არჩევა (ერთზე მეტ-პროცესორიან სისტემებში).

სხვა საკონფიგურაციო ცვლილებები სრულდება Options განყოფილებაში. მაგალითად, ჰოსტ-მანქანის ამუშავებისას ავტომატურად გასტ-მანქანა რომ ამუშავდეს. ვირჩევთ StartUp/Shut Down -> On Host startUp -> Power On Virtual Machine

1.2.4. VSphere (VMWare Infrastructure)

მართავს ვირტუალურ მანქანების ინფრასტრუქტურას. ცენტრალიზებულად იმართება პროცესორები, ოპერატიული მეხსიერება და ა.შ. ინფრასტრუქტურის იერარქიული აგებულება. მოცემულია 1.1 ნახაზზე



ნახ.1.1.

შენიშვნა: ვირტუალური მანქანის გადატანა შესაძლებელია ერთი მონაცემთა ცენტრის ფარგლებში ჰოსტებს შორის (ტექნოლოგია VMotion), მაგრამ არა მონაცემთა ცენტრებს შორის.

ვირტუალური მანქანა ვირტუალური მოწყობილობის (Virtual Appliance) ნაირსახეობაა, ხოლო ეს უკანასკნელი წინასწარ აგებულ და კონფიგურირებული, შესასრულებლად გამზადებული პროგრამაა, რომელიც თავის შესაფერის ოპერაციულ სისტემიანა შეიძლება ვირტუალურ ინფრასტრუქტურაში ჩაინერგოს. ვირტუალური ალიანსი შეილება იყოს თვით ოპერაციული სისტემა, თამაში და სხვა.

ვირტუალური მანქანების და ჰოსტების გარკვეული სიმრავლისგან მიიღება ვირტუალური ქსელები (VM Network).

სქემები (Maps) ინფრასტრუქტურის პროგრამის გრაფიკული კომპონენტია, რომელიც სქემატურად გამოსახავს დამოკიდებულებებს ინფრასტრუქტურის კომპონენტებს შორის. დამოკიდებულებათა ტიპებია:

ჰოსტი -> ვირტუალური მანქანა (ყოველი ვირტუალური მანქანა ფიზიკურად ერთ ჰოსტზეა განთავსებული, ყოველი ჰოსტი შეიცავს 0, 1 ან მეტ ვირტუალურ მანქანას. – ლურჯი კავშირის ხაზი;

ჰოსტი -> ვირტუალური ქსელი – მწვანე კავშირის ხაზი;

ვირტუალური მანქანა -> ვირტუალური ქსელი (?) - მწვანე კავშირის ხაზი;

ჰოსტი -> მონაცემთა საცავი (ყოველი ჰოსტს გააჩნია მიმართვის უფლება მონაცემთა საცავის ყოველ ელემენტზე, თუ ფილტრი არ აყენია – შავი კავშირის ხაზი;

ვირტუალური მანქანა -> მონაცემთა საცავი. ყოველი ვირტუალური მანქანა განთავსებულია ჰოსტის ლოკალური მეხსიერების ან მონაცემთა საცავის ერთ-ერთ ელემენტზე - შავი კავშირის ხაზი;

ინფრასტრუქტურის კიდევ ერთი ელემენტია რესურსთა ნაკრები (Resource Pool), რომელიც პროცესორის სიმძლავრეებს და

ოპერატიული მეხსიერების მოცულობებს ოპერატიულს ანაწილებს ვირტუალურ მანქანებზე და ჰოსტებისგან დამოუკიდებელია (?).

კონვერტირება. კონვერტირებისთვის ინფრასტრუქტურის პროგრამაში შესაბამისი პლაგინი უნდა დაინსტალირდეს და გააქტიურდეს (ბრძანება Plugins -> Manage PlugIns), რის შემდეგაც ინფრასტრუქტურული ერთეულის (კლასტერი) გააქტივებისას მთავარ მენიუს პუნქტში Inventory -> Cluster ან კონტექსტურ მენიუში გამოანათებს ბრძანება Import Machine. ეთითება წყარო (ფიზიკური მანქანა ქსელში – სახელი ან IP-მისამართი, ვირტუალური მანქანა – VMWare ან MS Virtual Machine, სხვადასხვა იმიჯები – Norton Ghost და სხვ.). დანიშნულების ადგილად ეთითება რომელიმე ESX-სერვერი (ჰოსტი) და დისკი – ლოკალური ან SAN-იდან. კონვერტირებული მანქანა ESX-სერვერის ფორმატისაა (გაფართოება?) და ჩვეულებრივი VMWare Server-ის გარემოში უკვე ვეღარ აღიქმება.

მიგრაცია. ჩვეულებრივ, კლასტერის ერთი კვანძიდან მეორეში ვირტუალური მანქანა ინფრასტრუქტურის ფარგლებშივეა შესაძლებელი, ოღონდ გამორთულ მდგომარეობაში (მანქანაზე Rmouse -> Migrate). ინსტრუმენტ Vmotion-ის ინსტალაციის შემდგომ (გრაფიკული ინტერფეისი არა აქვს) მიგრაციის იგივე ბრძანება ჩართულ ვირტუალურ მანქანებზეც ვრცელდება. მაგრამ ინფრასტრუქტურის პროგრამის ფასია 3000 ევრო, ხოლო VMotion-უტილიტის დამატებით – 5000.

1.3. Microsoft-ის ვირტუალური მანქანა Hyper-V

წარმოადგენს Windows 2008 Server-ის გაფართოებას და ინტეგრირდება მასში შესაბამისი, ე.წ. როლის (Role) გააქტიურებით (ბრძანება Server Manager -> Roles).

აპლეტი: %ProgramFiles%\Hyper-V\virtmgmt.msc

VMWare to MS VM

ფიზიკური ან ვირტუალური VMWare მანქანების კონვერტირების პროცედურა მაიკროსოფტის ვირტუალურ მანქანებად 3

ეტაპად შეიძლება დაიყოს. პირველზე ფიზიკური ან VMWare-მანქანა გარდაიქმნება VMWare-ვირტუალურ ხისტ დისკად ზედ ოპერაციული სისტემით (მოდიფიცირდება გარე მეხსიერების, ოპერატიული მეხსიერების, პროცესორთ რიცხვის მნიშვნელობები ეტ ცეტერა), ხოლო შემდეგ VMWare-დისკი გარდაიქმნება MS VM-დისკად. მესამე ეტაპზე MS VM-დისკის ბაზაზე მაიკროსოფტის ვირტუალური მანქანა აიგება.

პირველ პროცედურას ემსახურება უტილიტა VMWare Converter, რომელიც შეიძლება იყოს ნაწილი VMWare Infrastructure-სი აპლეტის სახით ან ცალკე პაკეტად მოიხმარებოდეს.

მეორე პროცედურის მსახურია პაწია უტილიტა vmdk2vhd (vmdk – VMWare-დისკის ფორმატი, vhd – MS VM-დისკისა).

ხოლო MS VM ვირტუალურ დისკიდან ვირტულურ მანქანის ასაგებად Windows 2008 Server-ის გარემოდან ვირტუალურ მანქანების მართვის Hyper-V-პროგრამა გამოიძახება. ბრძანებით New -> Build Virtual Computer იწყება ახალ ვირტუალურ კომპის აგების პროცედურა, სადაც ერთ ეტაპზე ვირტუალური დისკის არჩევისას უნდა გააქტიურდეს ოფცია Use existed Virtual Hard Drive და მანდ შესაბამისი vhd-ფაილი უნდა მიეთითოს.

1.4. სერვერთა კლასტერი

სერვერკლასტერი ერთმანეთში ერთი სისტემის სახით დაკავშირებული კვანძების (კლასტერების) ერთობლიობას ქვია, რომლებსაც მათ ერთიან სისტემად კლასტერის სერვისი (Clusterdienst) აქცევს (იხ.სერვისების სიაში). კლასტერი შედგება კვანძებისგან, რომლებიც ერთმანეთს რეგულარულ ტაქტებს უგზავნიან და აკონტროლებენ. თუ ერთი კვანძი წყობიდან გამოვიდა, მის ამოცანას და რესურსებს მეორე გადიბარებს (FailOver). Windows 2003-ში კვანძების მაქსიმალური ოდენობა – 8.

ყოველ კვანძს მინიმუმ 1 ქსელური ადაპტერი ჭირდება, ბევრად ჯობია თუ ორი ექნება. კლასტერის ქსელურ ადაპტერებს

ფუნქციების მიხედვით გამოარჩევენ:

1. მხოლოდ კვანძთაშორის კომუნიკაციისთვის. ამ დროს მეორე ადაპტერი სრულიად აუცილებელია კლიენტებთან კავშირისთვის. კვანძთა შორის კავშირები საკუთარი, შიდა ლოკალური ქსელის სახით აიგება (192.168 ტიპის IP-მისამართებით)

2. მხოლოდ კლიენტისა და კლასტერის კავშირისთვის. აქაც მეორე ადაპტერი მოითხოვება კვანძებს შორის კავშირისთვის.

3. ორთავე ზემორე ტიპი ერთად. ამ დროს კვანძი ერთი ადაპტერითაც გავა იოლას, მაგრამ ორი მაინც ჯობია: ყოველი ადაპტერი ორთავე ფუნქციას შეითავსებდა და ერთმანეთს დააზღვევდნენ.

კლასტერის კვანძს ყველა ქსელის კომპიუტერივით უნდა IP-მისამართი თითოეული ადაპტერისთვის (ჯობს სტატიკური, ვიდრე DHCP-სერვერიდან განაწილებული) და DNS-სახელი, რომელიც არ უნდა თანხვდებოდეს დომენის და მასში შემავალ კომპიუტერების სახელებს, აგრეთვე სხვა კვანძების სახელებს.

კლასტერს ორი ტიპის მეხსიერება შეიძლება ქონდეს: ჰოსტების (კვანძების) ლოკალური და კლასტერული (iSCSI ან FibreChannel ტექნოლოგიით). პირველი კვანძებშივე არსებული (შეიძლება დამოუკიდებელ გარე მოწყობილობის სახითაც) ლოკალურ ხისტი დისკ(ებ)ია და ძირითადად პროგრამებს შეიცავს, მეორე – ხისტი დისკების საცავი ან მისი ნაწილი და მონაცემებს ინახავს, ამასთან ყოველი საცავის ყოველი დისკი (და საერთოდ კლასტერის ყველა რესურსი: დისკები, IP-მისამართები, ქსელური სახელები, სისტემური და სხვა სერვისები, განაწილებული საქაღალდეები, გამოყენებითი პროგრამები და სხვა) ფიზიკურად მხოლოდ ერთ კვანძს ეკუთვნის (სურათი). მიმდინარე კვანძზე ავარიის შემთხვევას კლასტერის სერვისი დააფიქსირებს და რესურსს ავტომატურად სხვა კვანძზე მიამაგრებს.

საჭირო შენიშვნები: კლასტერის აგებისას ოპერაციული სისტემის სისტემური (მაგ. C:) დისკი და კლასტერის დისკები ერთ

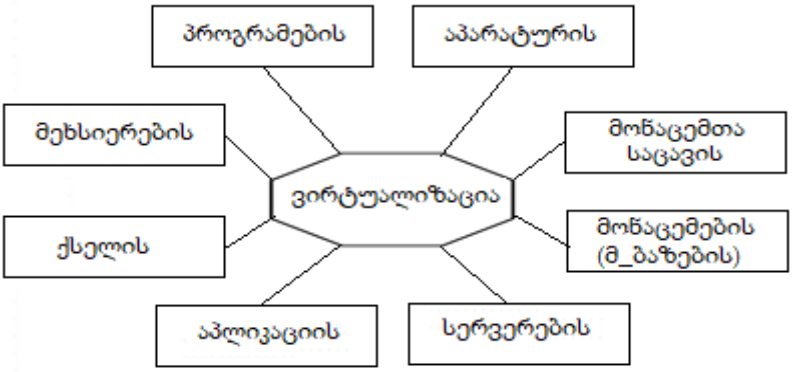
SCSI-არხზე (SCSI Path ID) არ უნდა განთავსდნენ, თუმცა საერთო ადაპტერის ქონა შეუძლიათ (SCSI Port Number).

კლასტერის მეხსიერება (Cluster Memory, Clusterspeicher) შიდა ან გარე მეხსიერებაა (დისკების მასივი). მეხსიერების რომელიმე მოწყობილობა დროის მოცემულ მომენტში ფიზიკურად 1 და მხოლოდ 1 კვანძს ეკუთვნის.

1.5. ვირტუალიზაციის სახეების კლასიფიკაცია

დასკვნის სახით შეიძლება აღვნიშნოთ, რომ *ვირტუალიზაცია* კომპიუტერული რესურსების ერთობლიობა ან მათი ლოგიკური გაერთიანებაა, რომელიც აბსტრაგირებულია ერთი აპარატურის ფიზიკურ რესურსებზე და რეალიზებულია გამოთვლით პროცესებს შორის სრული ლოგიკური იზოლაცია. მაგალითად, ერთ კომპიუტერზე ორი ოპერაციული სისტემა, ან რამდენიმე პროგრამული აპლიკლაცია და ა.შ.

ვირტუალიზაციის სახეები, რომლებიც გამოიყენება დღეს, წარმოდგენილია 1.2 ნახაზზე.



ნახ.1.2. ვირტუალიზაციის სახეები

მომდევნო თავებში ჩვენ განვიხილავთ ვირტუალიზაციის სახეების პრაქტიკული გამოყენების მაგალითებს, განსაკუთრებით პროგრამული აპლიკაციების შექმნის თვალსაზრისით.

II თავი

Linux ოპერაციული სისტემის გამოყენება

2.1. კვლევის მიზნები და ამოცანები

ნაშრომის მიზანად დასახულ იქნა მონაცემთა შენახვა-დამუშავების განაწილებული ეკოსისტემის შექმნა დაბალი ღირებულების კომპიუტერული ტექნოლოგიებისა და აპარატურული საშუალებების გამოყენებით, რაც ხელმისაწვდომი იქნება სასწავლო და რეალურ გარემოში სამუშაოდ.

მონაცემთა შენახვა-დამუშავების განაწილებულ ეკოსისტემაში იგულისხმება რიგი აპარატურული და პროგრამული ელემენტების ურთიერთდაკავშირებით მიღებული ერთიანი სისტემა, რომელიც სხვადასხვა ტექნოლოგიების გაერთიანებაა.

ეს ელემენტებია ინფორმაციის მატარებლები, ქსელური მარშრუტიზატორები და გამოთვლითი რესურსი. აღნიშნული ტექნოლოგიები გამოიყენება ერთდროულად მონაცემების შესანახად და დასამუშავებლად. ნაშრომში დასახული მიზნის მისაღწევად არსებული აპარატურული და პროგრამული ელემენტებიდან ავარჩიოთ ამოცანაზე მორგებული ტექნოლოგიები, რომლებიც გამოყენებულ იქნა პროექტის ფარგლებში [86].

პროექტის ფარგლებში ჩატარდება შემდეგი სახის სამუშაოები:

- *აპარატურის მოძიება:* პროექტის ფარგლებში მოძიებული აპარატურა იყოფა რამდენიმე სახეობად: ქსელური ინფრასტრუქტურა, კომპიუტერული ტექნიკა, გაგრილების სისტემა და კვების წყარო. აღნიშნული ტექნიკა უნდა იყოს ერთმანეთთან თავსებადი;
- *ოპერაციული სისტემების მოძიება:* ოპერაციული სისტემა ერთ-ერთი ყველაზე მნიშვნელოვანი კომპონენტია, რადგან ის პლატფორმაა, რომელზეც უნდა დაშენდეს ეკოსისტემის ძირითადი ნაწილი. ის უნდა იყოს მოქნილი, სტაბილური და ხელმისაწვდომი;
- *ინფორმაციის შენახვის განაწილებული სისტემების მოძიება;*

- *განაწილებული გამოთვლითი სისტემების მოძიება;*
- *მოძიებული ტექნოლოგიების ერთმანეთთან დაკავშირება და მონაცემთა შენახვა-დამუშავების განაწილებული ეკოსისტემის დაპროექტება*;

ეტაპობრივი სამუშაოების შედეგად გამართულ იქნა სისტემა, რომლის გამოყენებითაც შესაძლებელი ხდება ინფორმაციის შენახვა-დამუშავება განაწილებულ გარემოში.

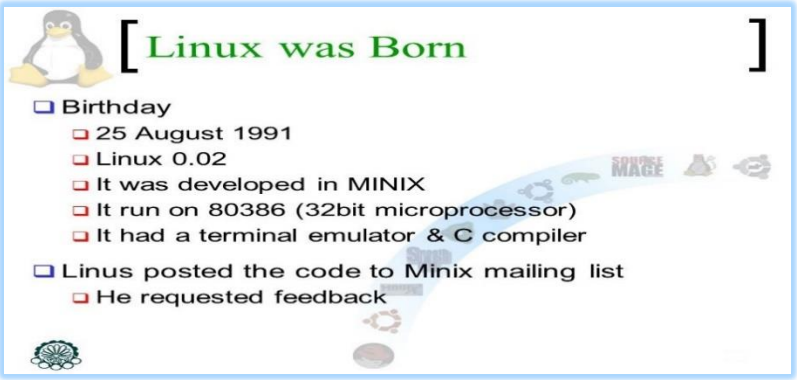
2.2. Linux ოპერაციული სისტემა

ოპერაციული სისტემა ინფორმაციის შენახვა-დამუშავების განაწილებული სისტემის ერთ-ერთი ყველაზე მნიშვნელოვანი კომპონენტია. Unix-ის ტიპის ოპერაციული სისტემებიდან ამოვარჩიოთ ისეთი, რომელსაც გამოვიყენებთ პროექტის ფარგლებში.

Linux არის GPL ლიცენზიაზე დამყარებული თავისუფლად გავრცელებადი ოპერაციული სისტემა, რომელიც UNIX ოპერაციული სისტემის მსგავსად შეიქმნა. მისი თავდაპირველი ვერსია შეიქმნა ჰელსინკის უნივერსიტეტის სტუდენტის ლინუს ტორვალდსის მიერ (Linus Torvalds) [87].

UNIX არის მრავალ მომხმარებლიანი და მულტი-ფუნქციური ოპერაციული სისტემა რომელიც დაპროექტდა და შეიქმნა "Bell Labs"-ის მიერ 1970 წელს. UNIX ის შექმნის მთავარი მიზანი იყო, ყოფილიყო რაც შეიძლება პატარა და მოქნილი სისტემა რომელსაც გამოიყენებდნენ პროგრამისტები. UNIX იყო ერთ-ერთი პირველი ოპერაციული სისტემა, რომელიც იყო დაწერილი მაღალი დონის პროგრამირების "C" ენაზე, რაც საშუალებას გვაძლევს აღნიშნული ოპერაციული სისტემა დავაყენოთ ისეთ აპარატურაზე რომელსაც „C“-ის კომპილატორი აქვს. რაც ბევრად ზრდის ჩვენთვის მისაღები აპარატურის რაოდენობას [12,88].

Linux ოპერაციული სისტემის შექმნაში MINIX, UNIX პროგრამისტთა და ინტერნეტის ქსელში მომუშავე ასობით ენთუზიასტს მიუძღვის წვლილი (ნახ.2.1) [27, 88, 89].



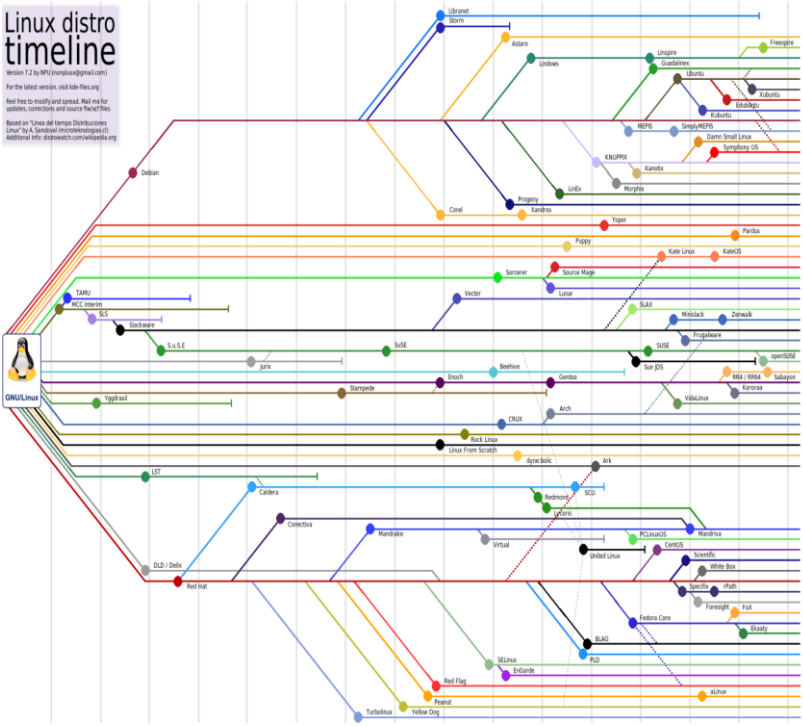
ნახ. 2.1.. Linux

მასში არაა გამოყენებული AT&T UNIX-ის ან არც ერთი სხვა UNIX სისტემის პროგრამული კოდი. Linux-ის პროგრამების უმეტესობა შექმნილია კემბრიჯში, მასაჩუსეტში, GNU Free Software Foundation პროექტის ჩარჩოში, თუმცა მის შექმნაში მონაწილეობა მიღებული აქვთ პროგრამისტებს მთელი მსოფლიოდან. ასევე აღსანიშნავია, რომ Linux-ის ბაზაზე შექმნილი პროგრამების უმეტესობას ღია პროგრამული კოდი აქვს (open source) და უფასოა [26].

Linux ოპერაციული სისტემის გამოყენება წარმატებით შეიძლება როგორც სამუშაო სადგურად, ასევე სერვერის პლატფორმად. დღესდღეობით Linux არის UNIX-ის ოჯახის საოპერაციო სისტემების ღირსეული გაგრძელება. Linux სრულყოფილ, მრავალ ფუნქციონალურ და მრავალ ამოცანიან ოპერაციულ სისტემას წარმოადგენს, რომელიც თანამედროვე ინფორმაციული ტექნოლოგიების მზარდ მოთხოვნილებებს წარმატებით ართმევს თავს.

Linux სისტემის ეფექტურობამ და მოქნილობამ ბევრი გულშემატკივარი გაიჩინა, ტექნოლოგიების განვითარებასთან ერთად, გაჩნდა ლინუქსის სხვადასხვა ვარიაციები. მსხვილმა კომპანიებმა დაიწყეს Linux ოპერაციული სისტემის გადაწერა და

თავიანთი დისტრიბუციების შექმნა; საბოლოო ჯამში, ოპერაციული სისტემა დაიყო Debian-based და RPM-based ორი ძირითად მიმართულებად, რომლებიც თავის მხრივ იყოფიან სხვადასხვა დისტრიბუციებად [13].



ნახ.2.2. ლინუქსის ოპერაციული სისტემის დისტრიბუციების სქემა

როგორც 2.2 ნახაზზეა ნაჩვენები „RPM-Based“ მიმართულება შეიცავს შემდეგ დისტრიბუციებს: Red Hat Linux, openSUSE, Fedora, Mandriva Linux და ა.შ., რომლებიც კიდევ იყოფა და მომხმარებლისთვის მზა პროდუქტს წარმოადგენენ. მაგალითად:

Red Hat Based

CentOS

Oracle Linux

openSUSE-based
SUSE Linux Enterprise Desktop
SUSE Linux Enterprise Server
Fedora-based
Berry Linux
EduLinux
Mandriva Linux-based
Annvix
TinyMe

2.2.1. Linux ოპერაციული სისტემის ფაილური სტრუქტურა

ფაილური სისტემა განსაზღვრავს, თუ როგორ მოხდეს ოპერაციულ სისტემაში ინფორმაციის შენახვა-დამუშავება. შესაბამისად, აუცილებელია კარგად გვქონდეს გაცნობიერებული, თუ რა გვჭირდება და რისი მოცემა შეუძლია ამა თუ იმ ფაილურ სისტემას.

ლინუქსს აქვს როგორც საკუთარი, ასევე სხვა ოპერაციული სისტემების ფაილური სისტემების მხარდაჭერა. თუმცა სხვა ოპერაციული სისტემების ფაილურ სისტემებს გარკვეული შეზღუდვები ახასიათებს, მაგალითად, როდესაც მათი გამოყენება ლინუქსის სისტემური დირექტორიებისთვის ხდება. ამიტომ, როგორც წესი, სასურველია გამოვიყენოთ ლინუქსის ფაილური სისტემები, რომლებიც კონკრეტულად ლინუქსისთვის არის შექმნილი და გათვალისწინებულია Linux ოპერაციული სისტემის სპეციფიკაციები [14].

Linux ფაილური სისტემებიდან უნდა გამოვყოთ შემდეგი გაფართოებული ფაილური სისტემები: Ext2FS (Ext2) - ლინუქსის ტრადიციული ფაილური სისტემა, რომელიც სპეციალურად ლინუქსისთვის 90-იანი წლების ბოლოს შეიქმნა. Ext2FS სტაბილურ ფაილურ სისტემად ითვლებოდა და მიუხედავად იმისა, რომ დღეს ის სხვა, გაუმჯობესებული ფაილური სისტემებით არის

ჩანაცვლებული, ის მაინც გამოიყენება ზოგ მანქანაზე. მაგალითად, Ext2FS-ის გამოყენება შესაძლებელია /boot სისტემური დირექტორი-ისათვის. მცირე მოცულობის დანაყოფისთვის ისეთი ფაილური სისტემების გამოყენება, როგორიცაა Ext3 ან Ext4 ნაკლებად მოხერხებულია, რადგან აღნიშნულ ფაილურ სისტემებს დიდი ზომის ჟურნალი გააჩნიათ. ამიტომ, მცირე ზომის დირექტორიე-ბისთვის უპირატესობის მინიჭება შეგვიძლია არაჟურნალირებადი ფაილური სისტემისთვის (Ext2).

Ext3FS (Ext3) მესამე გაფართოებული ფაილური სისტემა, ფაქტობრივად, იგივე Ext2-ია, მხოლოდ ჟურნალის მხარდაჭერით. შედეგად ვიღებთ Ext2-ის მსგავს სტაბილურ ფაილურ სისტემას, მაგრამ კვების გათიშვის ან სისტემის გაუთვალისწინებელი გადატვირთვის შემთხვევაში სისტემა უფრო სწრაფად აღდგება.

Ext4fs (Ext4) მეოთხე გაფართოებულ ფაილურ სისტემას გააჩნია ძალიან დიდი ზომის მყარ დისკებთან ან დიდი ზომის ფაილებთან მუშაობის შესაძლებლობა. Ext4 შემდეგი თაობის ფაილურ სისტემას წარმოადგენს [14,15].

XFS (Extents File System - XFS) ფაილური სისტემა Silicon გრაფიკის (CGI) მიერ არის შექმნილი, IRIX OS ოპერაციული სისტემისთვის. მოგვიანებით პროგრამული კოდი ლინუქსს გადაეცა. XFS ტექნიკურად საკმაოდ ინტელექტუალური ფაილური სისტემაა. IRIX ოპერაციულ სისტემაზე XFS ფაილური სისტემა საკმაოდ კარგ, ჩქარ და მოქნილ ფაილურ სისტემად ითვლებოდა. ოპერაციული სისტემიის დირექტორიები აღნიშნულ ფაილურ სისტემებზე ხის სტრუქტურით არის მოწყობილი. „ / “ - ძირეული დირექტორიით, რომელშიც მოთავსებულია დანარჩენი დირექტო-რიები: /swap, /boot, /root , /home, /usr, /var, /proc და ა.შ [16,17].

ლინუქსის ოპერაციული სისტემა საშუალებას გვაძლევს სისტემის დაყენებისას განვსაზღვროთ ამა თუ იმ დირექტორიის ზომა, ჩაწერის ადგილი და ა.შ. იმისათვის რომ სწორად დავაპროექტოთ სისტემა, მნიშვნელოვანია, ვიცოდეთ, თუ რომელი

დირექტორიების გამოყოფა ხდება. გასათვალისწინებელია ის ფაქტი, რომ მოცემული დირექტორიებისთვის გამოყოფილი მოცულობის შერჩევა დამოკიდებულია იმაზე თუ რა ტიპის მოხმარებისთვის იქნება გამოყენებული (სამომხმარებლო კომპიუტერი, ვებ-სერვერი, მონაცემთა ბაზის სერვერი და სხვ.).

შესაბამისად, რთულია კონკრეტული რეკომენდაციების მიცემა იმის თაობაზე, თუ რა მოცულობის უნდა იყოს თითოეული მათგანი. ლინუქსის სისტემაში ერთ-ერთი ყველაზე მნიშვნელოვანია swap სივრცე, რომელსაც ოპერაციული სისტემა იყენებს იმ შემთხვევაში როდესაც არ ყოფნის ოპერატიული მეხსიერება. მსგავს მომენტებში სისტემა იწყებს swap-ის გამოყენებას, ამიტომაც, აუცილებელია მოხდეს swap-ის სწორი კონფიგურირება (ცხრ.2.1). მაგალითად: მოთავსდეს შედარებით სწრაფ დისკზე (SSD დისკზე) და მისი ზომა ოპერაციული სისტემის მეხსიერების პროპორციულად უნდა იცვლებოდეს [18].

ოპერატიული მეხსიერება რეკომენდებული

swap-ის ზომები **ცხრ.2.1**

4GB ზე ნაკლები	მიწიმუმ 2 GB
4GB --- 16GB	მიწიმუმ 4 GB
16GB --- 64GB	მიწიმუმ 8 GB
64GB --- 256GB	მიწიმუმ 16 GB
256GB --- 512GB	მიწიმუმ 32 GB

"/" დირექტორია სისტემის ფუნქციონირებისათვის საჭირო ფაილებს შეიცავს. მისი მოცულობა უმეტეს შემთხვევაში მყარი დისკის მოცულობაზე არის დამოკიდებული.

/home დირექტორია ყველა იმ დირექტორიას და ფაილს შეიცავს, რომლებსაც სისტემის მომხმარებელი ქმნის. რამდენიმე მომხმარებლიან სისტემაში (მაგალითად. user1, user2), თითოეულ მომხმარებელს საკუთარი home დირექტორია ექნება (მაგალითად,

/ home / user1, / home / user2). მრავალმომხმარებლიან სისტემას, რომელზეც მომხმარებლები საკუთარ ინფორმაციას ინახავენ, სასურველია /home დირექტორიისთვის ცალკე დანაყოფი ჰქონდეს, რაც უზრუნველყოფს მომხმარებლების ინფორმაციის საიმედოდ შენახვას სისტემის ხელახალი ინსტალაციის ან მისი განახლების დროსაც კი, რადგან ინსტალაციის დროს შესაძლებელია ამ დირექტორიის უცვლელად, ფორმატირების გარეშე დატოვება. იმ შემთხვევაში, თუ /home დირექტორიისთვის არ არის გამოყოფილი სპეციალური დანაყოფი, აღნიშნული დირექტორია განთავსდება "/"-ზე და სისტემის ხელახალი ინსტალაციის და დანაყოფების ფორმატირების დროს /home დირექტორიაში არსებული ინფორმაცია წაიშლება. ინფორმაციის შესანარჩუნებლად საჭირო იქნება წინასწარ მისი გადატანა სხვა რომელიმე შემნახველ მოწყობილობაზე.

/boot დირექტორია ასევე ავტომატურად იქმნება სისტემის ინსტალაციის დროს და მასში ლინუქს სისტემის ჩამტვირთავი პროგრამის (მაგალითად, GRUB) და თავად ბირთვის ფაილები ინახება. რეკომენდებულია /boot დირექტორიის ცალკე გამოყოფა, მინიმალური ზომით 250 Mb [14].

/usr დირექტორიაში ინახება მომხმარებლის მიერ დაინსტალირებული ლინუქსის ძირითადი პროგრამები და მონაცემები. ეს, შესაძლოა, სისტემის ერთ-ერთი ყველაზე დიდი მოცულობის დირექტორია იყოს, რაც დამოკიდებულია იმაზე თუ რა რაოდენობის და მოცულობის პროგრამებს ვაყენებთ სისტემაზე.

/usr/local დირექტორიაში ლინუქსის ისეთი პროგრამები და მონაცემები ინახება, რომლებსაც მომხმარებელი თავად აკომპილირებს.

/opt წარმოადგენს ქვედირექტორიას, რომელშიც ძირითადად კერძო პროგრამები ყენდება, მათ შორის კომერციული პროგრამებიც. აღნიშნული დირექტორია ისეთი პროგრამებისთვის

გამოიყენება, რომლებიც ლინუქსის სტანდარტულ პაკეტებთან არაა ინტეგრირებული.

/var დირექტორია სისტემის ყოველდღიური ფუნქციონირებისთვის საჭირო ფაილებისა და სისტემური ლოგებისთვის არის განსაზღვრული.

/tmp დირექტორიაში დროებითი ფაილები ინახება. ისეთი დირექტორიები კი, როგორიცაა: / etc, / dev, / bin, / sbin, / lib სისტემურ დირექტორიებად ითვლება და მათში სისტემის ფუნქციონირებისათვის საჭირო პარამეტრები ინახება [16].

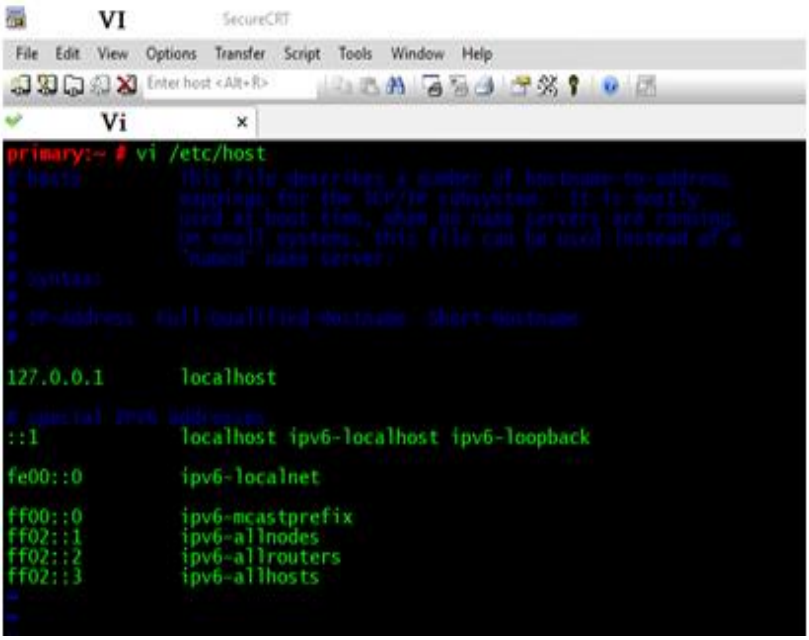
2.2.2. Linux ოპერაციული სისტემის შესაძლებლობები

Linux ოპერაციული სისტემის ბირთვი საკმაოდ მცირე რესურსს მოითხოვს და მხოლოდ 22 მილიონი ხაზის მოცულობის კოდისგან შედგება. ოპერაციული სისტემის გამართული ფუნქციონირებისათვის საკმარისია 400 Mhz პროცესორი, 256 MB ოპერატიული მეხსიერება და 10 MB მეხსიერების ზომა მყარ დისკზე. ეს ოპერაციულ სისტემას ბევრად უფრო მარტივს და მოსახერხებელს ხდის.

ოპერაციული სისტემის დაყენებისას მომხმარებელს შეუძლია მხოლოდ მინიმალური პაკეტის დაყენებაც, რის შემდეგაც დააყენებს მხოლოდ იმ პროგრამებსა და სისტემებს, რომლებიც მას სჭირდება. შესაბამისად ოპერაციული სისტემის ზომა მინიმალური გამოდის, რაც რისკებს მნიშვნელოვნად ამცირებს და ზრდის სისტემის საიმედოობას. ამ მახასიათებლების გამო ლინუქსს ხშირად იყენებენ შემდეგი მიზნებისთვის: firewall gateway, database server, web server, ftp server, http/https server, proxy server, distribution systems და ა.შ [19, 20].

ლინუქსის ოპერაციული სისტემა ფუნქციონირებისთვის საჭირო პარამერტებს ინახავს ტექსტურ ფაილებში. კონფიგურაციის

შესაცვლელად საკმარისია საჭირო ტექსტური ფაილის რედაქტირება, რისთვისაც გამოგვადგება „Vi“ რედაქდორი (ნახ.2.3).



ნახ.2.3. Vi რედაქტორი

„ Vi “ ტექსტური რედაქტორია, რომელიც საშუალებას გვაძლევს, ვიმუშაოთ არამარტო ტექსტურ ფაილებთან, არამედ პროგრამულ კოდთანაც. მას აქვს C, C++, Python და Java პროგრამირების ენების მხარდაჭერა.

Net-tools ხელსაწყოები გვეხმარება ქსელური პარამერტების გაწერასა და ქსელთან დაკავშირებულ ოპერაციებში. Net-tools ხელსაწყოები გვაძლევს ისეთი ბრძანებების გაშვების საშუალებას როგორიცაა: ifconfig, ifup, ipdown [20].

Tcpdump ქსელიდან შემომავალი და გამავალი პაკეტების ანალიზისთვის გამოიყენება. შესაძლებელია მიმდინარე პაკეტების და ასევე დაარქივებული პაკეტების ნახვაც [21].

Top ბრძანებით შესაძლებელია ოპერაციულ სისტემაში მიმდინარე პროცესების და აპარატურის დატვირთვის დონის გაგება.

ლინუქსის სისტემაში ერთ-ერთ ყველაზე მნიშვნელოვან ადგილს იკავებს „Bash-shell". იგი არის UNIX ოპერაციული სისტემისთვის შექმნილი ბრძანებების ენა, რომელიც დაიწერა 1989 წელს, როგორც უფასო პროგრამული პროდუქტი, რომლის საშუალებითაც ხდება ლინუქს ოპერაციულ სისტემაში ბრძანებების გაშვება [22].

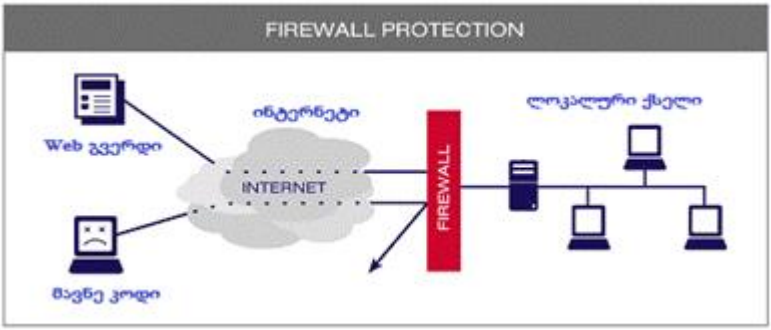
2.2.3. Linux ოპერაციული სისტემის უსაფრთხოება

ინფორმაციული ტექნოლოგიების როლი ჩვენ ყოველდღიურ ცხოვრებაში თანდათან იზრდება. აუცილებელი ხდება მათი უსაფრთხოების უზრუნველყოფა. განვიხილოთ Linux-ის ერთ-ერთი ყველაზე მნიშვნელოვანი თავდაცვის საშუალება „Firewall".

Firewall არის ყველაზე ეფექტური დაცვის საშუალება, რომელიც იცავს მომხმარებლებს სხვადასხვა შეტევებისაგან. Firewall მართავს და აკონტროლებს ქსელის ტრაფიკს და აღმოფხვრის არასანქცინირებულ წვდომას.

Firewall იყენებს სხვადასხვა მეთოდს იმის განსასაზღვრად, თუ რომელი წვდომაა ქსელთან ნებადართული ან შეზღუდული.

- *პაკეტის ფილტრაცია* - IP და MAC მისამრთებზე დაყრდნობით წყვეტს, მისცეს თუ არა კონკრეტულ მომხმარებლებს წვდომის უფლება ამა თუ იმ რესურსზე (ნახ.2.4) [23];



ნახ.2.4. Firewall

- *პროგრამის და Web Site-ის ფილტრაცია.* Firewall სპეციალური ცხრილების მეშვეობით საზღვრავს დაშვებულია თუ არა კონკრეტული ვებ-გვერდი თუ პროგრამული უზრუნველყოფა;
- *პაკეტის მდგომარეობის შემოწმება* - (SPI - Stateful Packet Inspection) - შემომავალი პაკეტები შიდა ჰოსტების მოთხოვნებზე გასცემენ პასუხებს. არასასურველი პაკეტები სპეციალური ნებართვის გარეშე იბლოკება. SPI-ის აგრეთვე შეუძლია ამოიცნოს და გაფილტროს DOS შეტევებიც [24];
- Firewall პროდუქტები სხვადასხვა სახის ფილტრაციის საშუალებებს უზრუნველყოფს. აგრეთვე, Firewall-ის გამოყენებით შესაძლებელია ქსელური მისამართის გარდაქმნაც (Network Address Translation -NAT). NAT შიგა მისამართს ან მისამრთების ჯგუფს გარე მისამრთად გარდაქმნის და ქსელში გზავნის. ამის შედეგად შიგა IP მისამრთები გარე მომხმარებლებისათვის მიუწვდომელია [25].

გამოცდილი მომხმარებლის ხელში Firewall-ის მექანიზმი Linux ოპერაციულ სისტემას მეტად დაცულს და საიმედოს ხდის. აღსანიშნავია ისიც, რომ ტექნოლოგიების განვითარებასთან ერთად ლინუქსის დაცვის მექანიზმებიც ვითარდება. ყოველივე ზემოთ აღწერილი Linux ოპერაციულ სისტემას იდეალურ კანდიდატად ხდის მონაცემთა შენახვა-დამუშავების განაწილებული ეკოსისტემის პლატფორმად გამოსაყენებლად.

2.3. ინფორმაციის შენახვა დამუშავების განაწილებული სისტემები და RAID მასივი

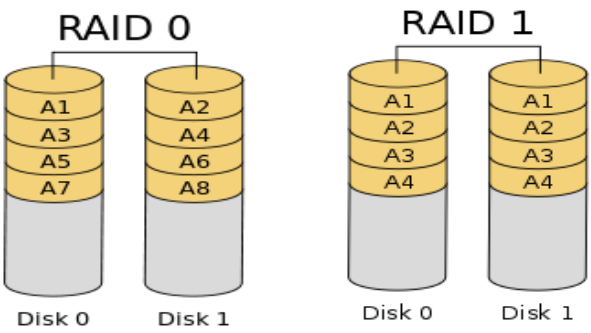
თანამედროვე სამყაროში ინფორმაცია ერთ-ერთი ყველაზე ძვირადღირებული და აუცილებელი პროდუქტი გახდა, შესაბამისად გაჩნდა ინფორმაციასთან წვდომის და შენახვის საიმედოობის ამაღლების მოთხოვნა. ამან წინა პლანზე წამოწია ისეთი საკითხები, როგორიცაა მონაცემების დუბლირება RAID მასივის გამოყენებით, ინფორმაციის შენახვაზე ორიენტირებული სისტემების პროექტირება: NAS, SAN, VSAN ტექნოლოგიების გამოყენებით [8].

ინფორმაციის რაოდენობის ექსპონენციალურმა ზრდამ საჭირო გახადა დიდი მოცულობის მონაცემების შენახვა-დამუშავება რეალურ დროში. რამაც თანამედროვე ტექნოლოგიები ახალი გამოწვევების წინაშე დააყენა. აღნიშნული მოთხოვნების დასაკმაყოფილებლად შეიქმნა განაწილებული სისტემები როგორიცაა: MongoDB, Hadoop, Spark [9].

განვიხილოთ დღეისათვის უკვე არსებული ტექნოლოგიები, რომლებსაც იყენებენ თანამედროვე სამყაროში. იმისათვის, რომ დავაპროექტოთ ინფორმაციის შენახვის სისტემა, საჭიროა წინასწარ ვიცოდეთ თუ რა ინფორმაციის შენახვას ვაპირებთ და რა მოთხოვნებია შენახულ ინფორმაციაზე. ეს მოთხოვნები შეიძლება იყოს: დიდი რაოდენობის ინფორმაციის ჩაწერა, წაკითხვა, სისწრაფე, საიმედოობა. აღნიშნული მოთხოვნების დასაკმაყოფილებლად შეიქმნა RAID მასივი.

2.3.1. RAID მასივები

RAID (Redundant Array of Independent/inexpensive Disks) – მასივი შედგება რამდენიმე დისკისგან, რომელიც იმართება კონტროლერის საშუალებით, ურთიერთდაკავშირებული „ჩქაროსნული მაგისტრალებით“ და წარმოდგენილი შიგა სისტემით, როგორც ერთი მთლიანობა (ნახ.2.5).



ნახ.2.5. დისკური მასივები RAID 0, RAID 1

მასივი ძირითადად მონაცემთა დაცვისა და ჩაწერა//წაკითხვის სიჩქარის გაზრდისათვის (RAID_0) გამოიყენება. თავდაპირველად RAID აბრევიატურა იშიფრებოდა როგორც იაფფასიანი დისკების რეზერვული მასივი („Redundant Arrays of Inexpensive Disks“), რადგანაც ისინი გაცილებით იაფი იყო, ვიდრე RAM). სწორედ ამ სახელწოდებით წარმოადგინეს ავტორებმა: David A. Patterson, Garth A. Gibson და Randy H. Katz 1987 წელს აღნიშნული მასივი.

შემდგომში RAID-ის განმარტება შეიცვალა და მას დამოუკიდებელი დისკების რეზერვული მასივი („Redundant Array of Independent Disks“) დაერქვა. მასივებში ამ დროს უკვე ძვირადღირებული მოწყობილობებიც გამოიყენებოდა [26].

თავდაპირველად RAID-ის იდეა დისკოების მასივში რამდენიმე მყარი დისკოს გაერთიანება და ამით წარმადობის გაუმჯობესება იყო. შემდგომში, საჭიროებიდან გამომდინარე RAID მასივის სხვადასხვა არქიტექტურა შეიქმნა. კერძოდ, კალიფორნიის „ბერკლი“-ის უნივერსიტეტმა RAID-ის სპეციფიკაციის შემდეგი დონეები წარმოადგინა [27]:

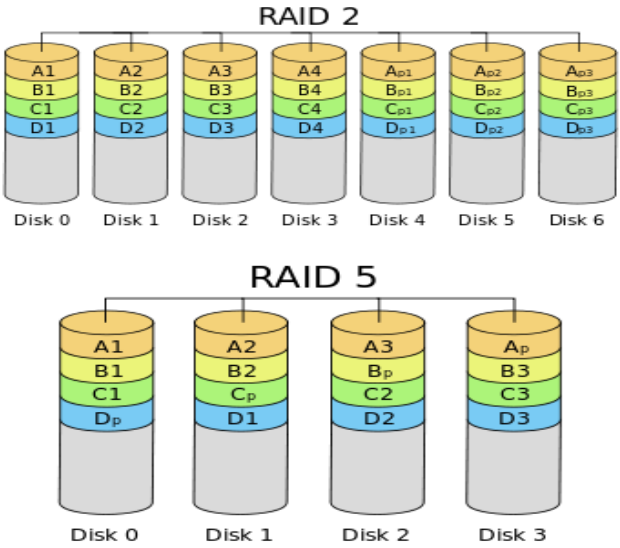
- RAID_0 – მონაცემები ნაწილდება სხვადასხვა დისკზე, რაც ნებისმიერ მომენტში სიჩქარის მატების საშუალებას გვაძლევს. ჩაწერის დროს მონაცემები ბლოკებად იყოფა და სათითაოდ ნაწილდება მასივში შემავალ დისკებზე. დისკების რაოდენობის

ზრდასთან ერთად უფრო და უფრო სწრაფად ხდება ინფორმაციის დისკებზე ჩაწერა, რადგან ინფორმაცია პარალელურად ყველა დისკზე იწერება. მონაცემების ამოღების დროს კი ყველა დისკიდან ერთდროულად შესრულდება წაკითხვის პროცესი.

აღნიშნული ტოპოლოგიის გამოყენებით მასივში დისკების მატებასთან ერთად იზრდება მასივის სიჩქარე და მოცულობა, მაგრამ რომელიმე დისკის დაზიანების შემთხვევაში დავკარგავთ მთელ ინფორმაციას, აღდგენა კი სარეზერვო ასლის არარსებობის გამო შეუძლებელი იქნება.

- RAID_1 – სარკისებური დისკების მასივი, მონაცემების ჩაწერისას ხდება 1:1–ზე კოპირება იგივე მონაცემების სხვა მასივის შემადგენელ დისკოებზე იმავდროულად. ანუ მასივის ყოველივე დისკო ერთსადაიმავე მონაცემებს ინახავს. შესაბამისად, მონაცემები ბევრად უფრო დაცულია, რადგან ერთი დისკოს მწყობრიდან გამოსვლის შემთხვევაში ინფორმაცია მაინც არ დაიკარგება. აღნიშნული ტოპოლოგია გვაძლევს მაღალ საიმედოობას, დისკოს დაზიანების შემთხვევაში ინფორმაციას არ დავკარგავთ, რაც უფრო მეტი დისკო იქნება მასივში, მით უფრო გაიზრდება საიმედოობა, მაგრამ შემცირდება დისკზე ჩაწერის დრო, რადგან თითოეული ჩაწერის დროს ინფორმაცია უნდა ჩაიწეროს რამდენიმე დისკზე, ამასთან არაეფექტურად ვიყენებთ მასივის ზომას;
- RAID_2 – გამოიყენება სარეზერვო მასივისათვის და იყენებს „ჰემინგის“ კოდს (ნახ. 2.6);
- RAID_5 – მასივის შექმნისათვის გამოიყენება სამი ან მეტი დისკო ისე, რომ დაცულ იქნას მონაცემები რომელიმე დისკოს მწყობრიდან გამოსვლის შემთხვევაში. RAID_5 მასივის ეს არქიტექტურა უფრო ოპტიმალურია, ვიდრე RAID_0-4. აღნიშნული ტოპოლოგია იძლევა ერთი დისკოს დაკარგვის საშუალებას ისე რომ ინფორმაციის დაკარგვისგან მაინც დაცულები ვიყოთ. მასივში არსებული ინფორმაცია დისკებზე ნაწილდება და თითოეულ დისკზე შენახულია P საკონტროლო ინფორმაცია რომელიც ერთ-

ერთი დისკოს დაზიანების შემთხვევაში საშუალებას გვაძლევს, ინფორმაცია ყოველგვარი დანაკარგის გარეშე აღვადგინოთ. იმ შემთხვევაში, თუ აპარატურას აქვს „Hot Swap“ რეჟიმის მხარდაჭერა, მაშინ აღნიშნული მასივის გამოყენება შესაძლებელია 24/7-ზე მუშაობისთვის. დისკის დაზიანების შემთხვევაში მისი გამოცვლა შესაძლებელია მუშაობის რეჟიმში „Down Time“-ის გარეშე) [28].



ნახ.2.6. დისკური მასივები RAID_2 და RAID_5

- RAID_6 – მასივის მონაცემები დაცულია ორი დისკოს მწყობრიდან გამოსვლის შემთხვევაშიც;
- RAID_10 – მასივი შეიძლება იყოს ორი 1+0 ან 0+1 ტიპის. მასივის ეს დიზაინი RAID-1-ისა და RAID_0-ის გაერთანებას წარმოადგენს (ნახ.2.7). პროექტირებისათვის საჭიროა 4 დისკო მაინც. აღნიშნული მასივი იძლევა გაორმაგებულ სისწრაფეს და საიმედოობას. აღნიშნული სისტემის გამოყენებისას საჭიროა

დისკოების ორმაგი რაოდენობა, რაც დამატებით ხარჯებთანაა დაკავშირებული. RAID_10-ის გამოყენება ძირითადად მონაცემთა ბაზებისთვის არის რეკომენდებული [29,30].

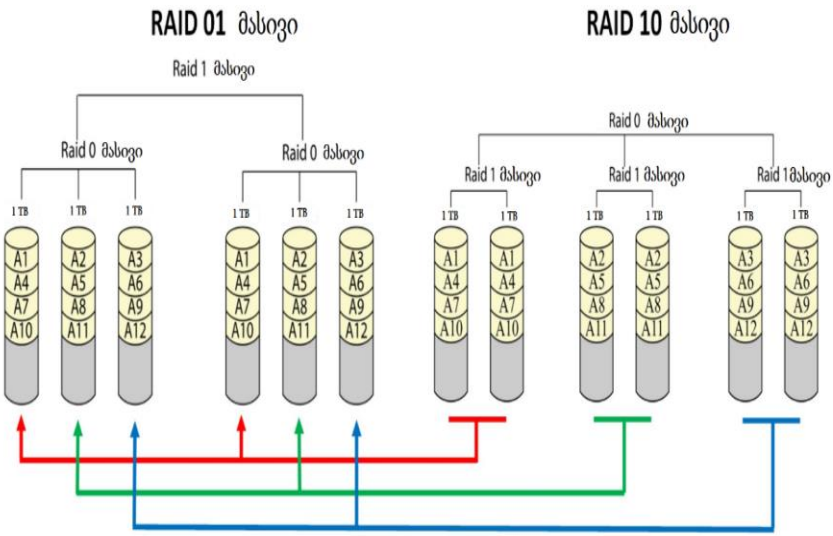
RAID მასივის კონფიგურაცია შესაძლებელია როგორც პროგრამულ, ასევე ფიზიკურ დონეზე. როდესაც დისკოებში შემავალი ინფორმაციის ნაკადი ოპერაციული სისტემიდან იმართება, დატვირთვა მოდის სისტემურ რესურსზე. დატვირთვის არიდების მიზნით შეიქმნა RAID კონტროლერები. აღნიშნული კონტროლერი პროცესორის ნაცვლად ასრულებს ყველანაირ გამოთვლას და თავად მართავს ინფორმაციის განაწილებას დისკოების მასივზე. ოპერაციული სისტემა მას ერთ დისკად აღიქვამს.

2.3.2. RAID მასივის კომბინირებული დონეები

RAID_0 – RAID_5 ბაზური მასივების გარდა არსებობს კომბინირებული მასივებიც: RAID_1+0, RAID_3+0, RAID_5+0, RAID_1+5, რომლებსაც ყველა მწარმოებელი სხვადასხვანაირად განსაზღვრავს. RAID_1+0 – ერთდროულად გულისხმობს RAID_1–სა და RAID_0–ს.

თანამედროვე კონტროლერები RAID_1-ს საწყის პარამეტრად იყენებს. პირველი მთავარი დისკოა, მეორე კი „სარკე“, სადაც დუბლირდება იგივე მონაცემები, რაც პირველ დისკოში. როგორც RAID_0-ის შემთხვევაში, წაკითხვის პროცესი ერთდროულად ორივე დისკოდან სრულდება.

RAID_1 და RAID_1+0 — ერთიდაიმავე მეთოდის, ოღონდ განსხვავებული დასახელებაა (დისკების აპარატული „სარკირება“). აღსანიშნავია ისიც, რომ სრულფასოვანი RAID_1+0-ისთვის მინიმუმ 4 მყარი დისკოა საჭირო. RAID_5+0 — არის მე-5 დონის ალტერნატივა. RAID 1+5 —RAID 5-ის სარკირებული წყვილი და ა.შ (ნახ.2.7).



ნახ.2.7. დისკური მასივები RAID_01 და RAID_10

კომბინირებულ დონეებსაც გააჩნია თავისი დადებითი და უარყოფითი მხარეები, რომლებიც მემკვიდრეობით ერგოთ „მშობლებისაგან“: ალტერნატივის გამოჩენა RAID_5+0 დონისთვის არაფერს მატებს მის სანდოობას, მაგრამ ამის საწინააღმდეგოდ, უარყოფითად აისახება მის წარმადობაზე. დონე RAID_1+5, ბევრად უფრო სანდოა, მაგრამ არც ისე სწრაფი და ამასთან, სავსებით არაეკონომიური: დისკოების საერთო მოცულობის ნახევარზე მეტი არ არის ხელმისაწვდომი.

საჭიროა აღვნიშნოთ ის ფაქტი, რომ მყარი დისკოების შესაძლო რაოდენობა კომბინირებულ მასივებში სხვადასხვაა. მაგალითად RAID_5+0 –ისთვის 6 ან 8 მყარი დისკო გამოიყენება, RAID_1+0 -სთვის – 4, 6 ან 8. RAID-ის მეშვეობით საგრძნობლად იზრდება მონაცემების ჩაწერა/წაკითხვის სისწრაფე. ტრადიციული აპარატული RAID-ის შემთხვევაში, გამოთვლებს კონტროლერი

ასრულებს. ხოლო პროგრამული RAID-ის შემთხვევაში მონაცემების გამოთვლა პროცესორს ევალება. ცხადია, ეს უკანასკნელი ამცირებს პროცესორის წარმადობას ისეთ ოპერაციებზე, რომელიც პროცე-სორზე არის დამოკიდებული. მარტივ კონტროლერებს შეუძლია მხოლოდ 0 და 1 –ის გამოყენება და ითხოვს ნაკლებ გამოთვლებს.

არსებობს ისეთი მასივებიც, რომლებიც აუცილებელია, რომ გამოვრთოთ, თუ გვჭირდება RAID-ში დისკოს დამატება ან ამოკლება. მაგრამ არსებობს ისეთებიც, სადაც შეგვიძლია პირდაპირ შევცვალოთ დისკო ჩართულ სისტემაზე. ამ პროცესს „ცხელი გამოცვლა“ („hot swapping”) ეწოდება. ასეთი ტიპის RAID მასივები ხშირად გამოიყენება მაღალი წვდომადობის („High Availability”) მქონე სისტემებში, სადაც უმნიშვნელოვანესია, რომ სისტემამ მაქსიმალურად უწყვეტად იმუშაოს [31].

RAID არ წარმოადგენს იდეალურ ალტერნატივას მონაცემების რეზერვული კოპირებისათვის. მონაცემები შეიძლება იქნას დაზიანებული ან განადგურებული იმ დისკოს დაზიანების გარეშეც, რომელზეც ეს მონაცემები ინახება. მაგალითად, სისტემური გაუმართაობის დროს შესაძლოა ზოგიერთ მონაცემს სხვა ინფორმაცია გადაეწეროს, ფაილი დაზიანდეს ან წაიშალოს მომხმარებლის შეცდომის, ან „ბოროტი განზრახვის“ გამო და ეს შეუმჩნეველი დარჩეს რამდენიმე დღის, ან კვირის განმავლობაში.

არც ფიზიკური დაზიანებისგანაა დაცული. RAID ორ ან მეტ მყარ დისკოს ერთ ლოგიკურ ბლოკში აერთიანებს, გამოიყენებს რა სპეციალურ აპარატურას ან პროგრამულ უზრუნველყოფას. აპარატურული უზრუნველყოფის შემთხვევაში სისტემა RAID-ს ერთ დისკად აღიქვამს. მაგალითად, სამი განსხვავებული 500GB -იანი დისკის გაერთიანება RAID_5-ში აპარატურულ რეჟიმში გვაძლევს 1TB მოცულობას და სისტემა აღიქვამს მას როგორც 1TB ზომის მყარ დისკოს. მსგავსი პრინციპი მოქმედებს პროგრამული მასივის შემთხვევაშიც. RAID-ის გაგებაში არსებობს სამი „საკვანძო“ სიტყვა:

- MIRRORING – მონაცემების კოპირება 1–ზე მეტ დისკოზე;
- STRIPING – მონაცემების გაყოფა (განაწილება) 1–ზე მეტ დისკოზე;
- ERROR CORRECTION- შეცდომების გამოსწორება.

2.3.3. მონაცემთა შენახვის სისტემები

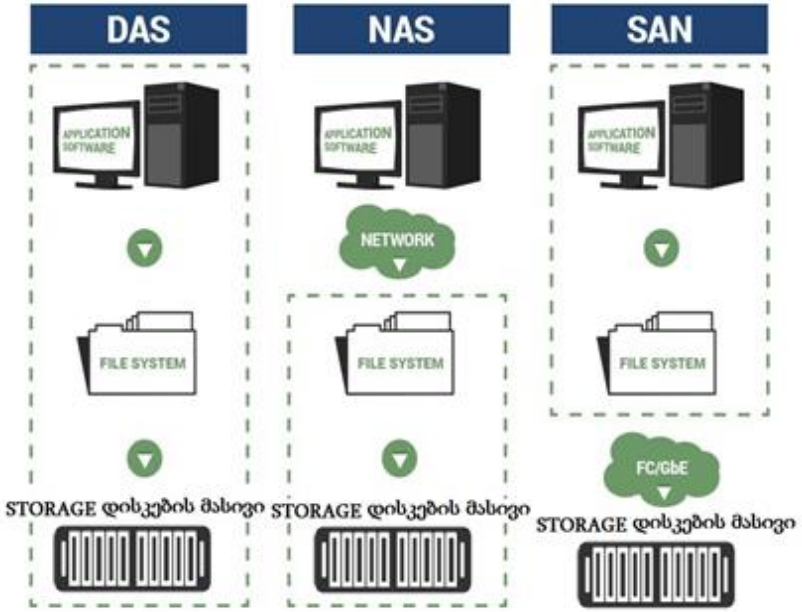
დღევანდელ ბიზნეს გარემოში, ინფორმაციის მოცულობისა და მნიშვნელობის სწრაფი ზრდის პირობებში, სულ უფრო მეტი ყურადღება ექცევა მონაცემთა რეზერვაციის, არქივირებისა და ქსელური შენახვის სისტემებს. ეს ლოგიკურიცაა - მონაცემთა ზრდასთან ერთად, იქმნება პრობლემები მათ უსაფრთხოებასა და ხანგრძლივად შენახვასთან დაკავშირებით. ამ მიმართულებით ახალი ტექნოლოგიების განვითარება და ფართოდ დანერგვა მრავალი კომპანიის წარმატებული საქმიანობის საწინდარი გახდა [32].

დღეისათვის აქტიურად გამოიყენება მონაცემთა შენახვის სისტემების სამი ძირითად მიმართულება [33]:

DAS (Direct Attached Storage) – არის იაფი და მარტივად სამართავი მონაცემთა შენახვის სისტემა, რომელიც მიერთებულია უშუალოდ სერვერებზე. იგი შეზღუდულია სერვერის რესურსის შესაძლებლობით. მოწყობილება შიგა მეხსიერებით, პირდაპირ უერთდება ძირითად კომპიუტერს და ლოკალური სარეზერვო კოპირების ამოცანებს ასრულებს. DAS არის ჩვეულებრივი ხისტი დისკო, რომელიც სერვერთან დაკავშირებისათვის ფართოდ გავრცელებულ SCSI – ტექნოლოგიას (Small Computer System Interface) იყენებს. DAS-სისტემა ვერ უზრუნველყოფს რამდენიმე სერვერიდან მონაცემთა სარეზერვო კოპირებას და მონაცემთა განაწილებას. ამგვარი მოწყობილობა სარეზერვო კოპირების იაფფასიანი ვარიანტია, თუმცა დიდი ორგანიზაციებისთვის გამოუსადეგარი.

NAS (Network Attached Storage) – IP ქსელში ჩართული შენახვის სისტემა. DAS-თან შედარებით, NAS-ში გარანტირებულია მეტი საიმედოობა და სისწრაფე. NAS - "მონაცემთა ქსელურ საცავს" გააჩნია საკუთარი MAC და IP-მისამართები და კომპიუტერული ქსელის სრულუფლებიანი წევრია [33,34].

NAS-სერვერის ძირითადი ფიზიკური კომპონენტი, როგორც წესი, მეხსიერების 1 ან მეტი ხისტი დისკოა, თუმცა ხანდახან მის კონფიგურაციაში გათვალისწინებულია ოპტიკური ან ლენტური მოწყობილობებიც. NAS-მოწყობილობა (NAS Appliance) ირთვება ქსელში და წარმოადგენს ფაილ-სერვერს საკუთარი მცირე ოპერაციული სისტემით და მართვის ვებ-ინტერფეისით. NAS-სერვერები გარეგნულად DAS-მოწყობილობებს ჰგავს, მაგრამ პრინციპულად განსხვავდება მისგან. მას დამატებული აქვს ფაილებთან წვდომის ეფექტური, ქსელური საშუალებები (ნახ.2.8).



ნახ.2.8. მონაცემთა შენახვის სისტემები NAS DAS SAN

აღნიშნული სისტემების გასამართად ძირითადად იყენებენ Linux-ის მსგავს ოპერაციულ სისტემებს რომელზეც ხდება აღნიშნული სისტემის გამართვა განვიხილოთ რამდენიმე სისტემის მაგალითი:

FreeNas – „მონაცემთა ქსელური საცავი“ ინფორმაციის შენახვის სისტემა, რომელიც არის უფასო და „OpenSource“ (NAS). იგი იყენებს FreeBSD ოპერაციულ სისტემას რომელიც ორიენტირებულია OpenZFS ფაილური სისტემის გამოყენებაზე. BSD ლიცენზიის პირობებით ლიცენზირებულია ლიცენზია და მუშაობს x86-64 ტექნიკის სასაქონლო პოზიციაში. FreeNAS მხარს უჭერს Windows, OS X და Unix კლიენტებს და სხვადასხვა ვირტუალიზაციის სისტემებს, როგორიცაა XenServer და Vmware. იგი იყენებს SMB, AFP, NFS, iSCSI, SSH, rsync და FTP / TFTP პროტოკოლებს. FreeNAS_ის დამატებითი შესაძლებლობებია მყარი დისკის სრული შიფრაცია და „Plug-in“ არქიტექტურა რომელიც საშუალებას აძლევს მოერგოს ყველანაირი ტიპის პროგრამულ უზრუნველყოფებს [35].

მისი კონკურენტია NAS4Free ოპერაციული სისტემა, რომელიც შეიძლება დამონტაჟდეს პრაქტიკულად ნებისმიერი აპარატურის პლატფორმაზე. იგი კომპიუტერულ ქსელზე მონაცემების შენახვის საშუალებას იძლევა. NAS4Free არის FreeNas-ის მსგავსად უფარო და ღია. NAS4Free არის მარტივი და სწრაფი გზა, რათა შეიქმნას ცენტრალიზებული და ადვილად ხელმისაწვდომი სერვერი ყველა სახის მონაცემებისთვის, ქსელის, პროტოკოლებისა და ნებისმიერი დანართებებისგან [36].

NAS4Free-ს მხარს უჭერს გაზიარების Windows, Apple და UNIX მსგავსი სისტემები. იგი მოიცავს ZFS v5000, პროგრამული RAID (0,1,5), დისკოს დაშიფვრის, SMART / ელექტრონული მეილ რეპორტინგი და ა.შ. შემდეგი პროტოკოლებით: CIFS/SMB (Samba), Active Directory Domain Controller (Samba), FTP, NFS, TFTP, AFP, RSYNC, Unison, iSCSI (initiator and target), HAST, CARP, Bridge, UPnP

და Bittorent, რომლებიც კონფიგურირებადია მისი WEB ინტერფეისდან. NAS4Free ინსტალაცია შეიძლება Compact Flash / USB / SSD დისკებზე, ასევე ხისტი დისკსა ან ჩატვირთვად LiveCD / LiveUSB- ის მოწყობილობაზე, რაშიც შეინახება კონფიგურაციის ფაილები.

Openmediavault – არის პროგრამული პროდუქტი, რომლითაც შექმნილია მონაცემთა ქსელური საცავის (NAS) სისტემა, რომელიც დაშენებულია დებიან ლინუქსის დისტრიბუციაზე. იგი უზრუნველყოფს მარტივ ვებ-ინტერფეისს, აქვს მრავალენოვანი მხარდაჭერა, ლოგიკური დისკოების მენეჯმენტი, მონიტორინგი და მოდული სისტემა, რომლითაც ხორციელდება LDAP, Bittorrent და iSCSI ტექნოლოგიების შესაძლებლობების გამოყენება.

მას აქვს შემდეგი პროტოკოლების მხარდაჭერა: CIFS (Samba), FTP, NFS, rsync, AFP, iSCSI, rapport S.M.A.R.T. რაც უფრო დიდია ბიზნესი, მით უფრო მეტად არის საჭირო მონაცემთა ქსელური საცავი (NAS).

Openmediavault არის მომდევნო თაობის NAS გადაწყვეტა Debian Linux- ზე, რომელიც შეიცავს სერვისებს, მათ შორის SSH, (S) FTP, SMB / CIFS, DAAP მედია სერვერი, RSync, BitTorrent კლიენტი და მრავალი სხვ. Openmediavault განკუთვნილია შესანიშნავად შეესაბამება სახლში და მცირე ბიზნესის, მისი უამრავი plugins საშუალების, მას შეუძლია ადვილად დააკმაყოფილოს საშუალო ბიზნესის მოთხოვნები [37].

Openmediavault არის მომავალი თაობის ქსელური საცავი (NAS), აგებული Debian Linux-ის საფუძველზე. იგი შეიცავს აგრეთვე სერვისებს SSH, (S) FTP, SMB / CIFS, DAAP-ის მედია-სერვერს, RSync, BitTorrent-ის კლიენტს და მრავალი სხვ. ფრეიმვორკის მოდულური კონსტრუქციის წყალობით შესაძლებელია მისი გაუმჯობესება პლაგინების საშუალებით. Openmediavault განკუთვნილია ძირითადად მცირე ოფისების ან სახლის ოფისებისათვის, მაგრამ იგი ამით არ შემოიფარგლება. ესაა მოხერხებული და მარტივად გამოსაყენებელი გადაწყვეტა, რომელიც საშუალებას აძლევს

ყველას, ღრმა ცოდნის გარეშე დააყენოს და ადმინისტრირება გაუწიოს ქსელურ საცავს. [37-39].

Openmediavault მოიცავს შემდეგ ძირითად ფუნქციებს [38]:.

- *ფუნქციები:*
  - Debian Linux (Jessie) OS;
  - Web based administration;
  - Easy system updates via Debian package management;
  - Usermanagement;
  - Scheduled jobs;
  - Multilanguage support;
  - Service announcement via DNS-SD;
  - Plugin system;
  - Networking;
  - Link aggregation;
  - Wake On Lan;
  - IPv6 support;
  - Volume management;
  - HDD power management (APM/AAM);
  - GPT partitions;
  - EXT3/EXT4/XFS/JFS filesystem support;
  - Software RAID JBOD/0/1/5/6/…
  - Quota (per volume)
  - ACL
  - Share management
- *მონიტორინგი:*
  - Syslog;
  - Watchdog;
  - S.M.A.R.T.;
  - SNMP (v1/2c/3) (read-only);
  - Email notifications;
  - Proactive process and system state monitoring;

- *სერვისები:*
  - SSH;
  - FTP;
  - TFTP;
  - NFS (v3/v4);
  - SMB/CIFS;
  - RSync;
- *მოდულები:* (პლაგინების სისტემის საშუალებით შეიძლება სერვისების დამატება):
  - LVM;
  - LDAP Directory Service;
  - AFP;
  - Bittorrent client;
  - DAAP server;
  - UPS;
  - iSCSI Target;
- *ანტივირუსი:*
  - clamav – Antivirus;
  - forked-daapd – DAAP media server;
  - ldap – User authentication via LDAP;
  - lvm2 – LVM management;
  - netatalk – AFP file sharing;
  - nut – Network UPS Tools;
  - route – Additional route management;
  - shairport – AirPlay/RAOP receiver;
  - usbbackup – USB autobackup;

SAN (Storage Area Network) - მონაცემთა საცავების ქსელი დღეისათვის ფუნქციურად წამყვანი ტექნოლოგიაა მაღალი საიმედოობისა და სისწრაფის გამო. მასში გამოიყენება ძალიან სწრაფი ოპტიკურ-ბოჭკოვანი შეერთება (Fiber Channel), ხოლო მონაცემთა შენახვის სისტემა ორგანიზაციის საერთო ქსელიდან

განცალკევებულია. ასეთი არქიტექტურის პირობებში მნიშვნელოვნად იზრდება რეზერვირებისა და არქივაციის სიჩქარეც და საიმედოობაც.

SAN მასობრივად ინერგება კორპორაციულ ქსელებში და მაღალი საიმედოობისა და სისწრაფის გამო დღეს მონაცემთა შენახვის ყველაზე გავრცელებული ტექნოლოგიაა. იგი დაფუძნებულია ძალიან სწრაფ ოპტიკურ-ბოჭკოვანი კავშირის Fibre Channel-ტექნოლოგიაზე, რომლის ინფორმაციის გადაცემის სიჩქარეა 10 გბ/წმ. ამასთანავე, მონაცემთა საცავის ქსელი ფიზიკურად განცალკევებულია ძირითადი კორპორაციული ქსელიდან.

მონაცემთა საცავისა და ძირითადი ქსელის განცალკევება ძლიერ ზრდის ინფორმაციის შენახვის საიმედოობას, არქივაციის პროცედურებისა და სარეზერვო ასლების შექმნის ეფექტურობას.

ასევე, მნიშვნელოვანია ძირითადი ქსელური არხების განტვირთვის ფაქტორიც, რადგან სწორედ არქივაცია და სარეზერვო კოპირება მოითხოვს ყველაზე დიდი ოდენობით ქსელურ რესურსებსა და ინფორმაციული არხების მაღალ გამტარუნარიანობას.

თანამედროვე სერვერული სისტემების ბაზარზე SAN-საცავების მრავალი ვარიანტი არსებობს, რაც დამკვეთი კომპანიების რადიკალურად განსხვავებული მოთხოვნების დაკმაყოფილებას უზრუნველყოფს. მაგალითად, თუ ჩვეულებრივი SCSI-ინტერფეისების მოქმედების რადიუსი მხოლოდ 25 მეტრს შეადგენს, Fibre Channel ტექნოლოგიის გამოყენება შეიძლება 100 კილომეტრის რადიუსში [40]. SAN-ქსელის ცენტრალური კომპონენტია ოპტიკური არხის ადაპტერი (Fibre Channel host bus adapter — FC HBA), რომელიც უზრუნველყოფს კავშირს სერვერებსა და მონაცემთა საცავებს შორის. კერძოდ, იგულისხმება შემდეგი ფუნქციონალი:

- ინტერფეისი მონაცემთა საცავებსა და სერვერებს შორის ნებისმიერი ქსელური ტოპოლოგიის ფარგლებში;

- მონაცემთა საცავების ცენტრალიზებული მართვა - კონფიგურაცია;
- მონიტორინგი და ქსელის კომპონენტების ანალიზი;
- დისკურ მასივებთან წვდომის უფლებების მართვა.

მონაცემთა დამუშავების ცენტრი (Data Center) მონაცემთა საცავების ყველაზე მასშტაბური მოდიფიკაციაა. იგი დიდი მოცულობის და მრავალფეროვანი ინფორმაციის (მონაცემთა ბაზები, მულტიმედია, ტექსტური) ეფექტურ და საიმედო შენახვას უზრუნველყოფს. მონაცემთა დამუშავების ცენტრი ნაკლებადაა გავრცელებული მისი მაღალი ღირებულების გამო. მეტწილად, საკუთარი მონაცემთა ცენტრები გააჩნიათ დიდ კორპორაციებს და ინტერნეტ სერვის პროვაიდერებს, რომლებიც მომხმარებლებს კარგად განვითარებულ ჰოსტინგ სერვისს სთავაზობენ [41].

სარეზერვო კოპირება და არქივაცია - კორპორაციული ქსელის სერვერული სისტემა წარმოუდგენელია სარეზერვო კოპირებისა და არქივაციის სისტემის გარეშე. ინფორმაციის სარეზერვო საცავები იძლევა იმის გარანტიას, რომ კორპორაციული ინფორმაცია არ დაიკარგება, უკიდურეს შემთხვევაშიც კი, თუნდაც ხანძრის ან სტიქიური უბედურების შედეგად. არსებობს სარეზერვო კოპირების მრავალფეროვანი სისტემები, რომელთაგან თითოეულს, შესასრულებელ ამოცანათა მოცულობისა და კომპიუტერული ქსელის მასშტაბების მიხედვით, საკუთარი გამოყენების სფერო გააჩნია.

მონაცემთა შენახვის აღნიშნული სისტემების მრავალ-ფეროვნება შესაძლებელს ხდის ორგანიზაციების მოთხოვნების დაკმაყოფილებას წვრილმანების გათვალისწინებითაც. აღნიშნული ტექნოლოგიების გამოყენებით შექმნილი მონაცემთა შენახვის სისტემების საშუალებით შესაძლებელია შემდეგი ამოცანების გადაწყვეტა:

- მონაცემთა ავტომატური რეზერვაცია და საჭიროების შემთხვევაში სწრაფად აღდგენა;

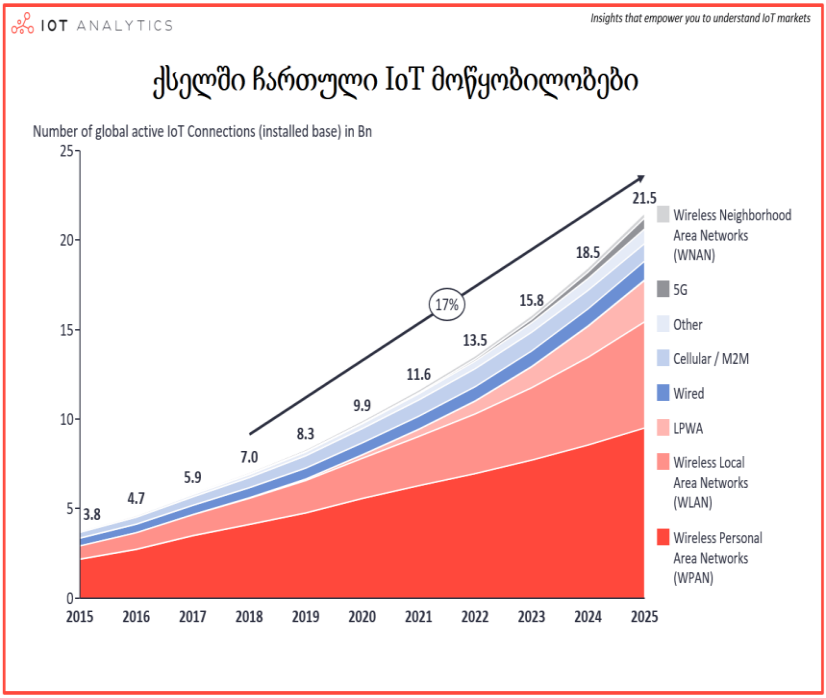
- დიდი მოცულობის მონაცემთა ცენტრალიზაცია და მართვის გამარტივება;
- იშვიათად გამოყენებადი ინფორმაციის არქივაცია და საიმედოდ შენახვა რამდენიმე ათეული წლის განმავლობაში.

მონაცემთა შენახვის სისტემების ასაგებად აღნიშნული ტექნოლეოგიების გამოყენება კრიტიკული ინფორმაციის დაცვას ადამიანური ფაქტორით გამოწვეული შეცდომების, ხანძრის და სხვა ავარიულ შემთხვევებში უზრუნველყოფს.

2.4. ამოცანის დასმა

თანამედროვე სამყაროში ტექნოლოგიების განვითარებამ შესაძლებელი გახადა „IoT“ (Internet of Things - საგანთა ინტერნეტი) მოწყობილობების ქსელში ჩართვა [36], რამაც გაუმარტივა მომხმარებლებს მოწყობილობების მიერ დაგენერირებულ ინფორმაციაზე წვდომა. ძალზე პოპულარული გახადა მსგავსი ხელსაწყოების შექმნა და გამოყენება. IoT Analytics-ის ცნობით, აღნიშნული მოწყობილობების რაოდენობა 2018 წლისთვის 7 მილიარდს შეადგენს ხოლო 2025 წლისთვის 21.5 მილიარდამდე გაიზრდება (ნახ. 9) [10,42,90].

შესაბამისად ამ მოწყობილობების მიერ დაგენერირებული მონაცემების დამუშავება დიდ სირთულეს წარმოადგენს. მსოფლიოს წამყვანი ორგანიზაციები აქტიურად არიან ჩართული „Big Data“ ტიპის მონაცემების დამუშავების ალგორითმებისა და ტექნოლგიების შექმნაში. დღეისათვის დიდი ზომის ინფორმაციის დამუშავების ყველაზე ეფექტურ მიდგომად ითვლება გამოთვლითი რესურსის ჰორიზონტალური ზრდის მოდელის გამოყენება. აღნიშნული მოდელი გვაძლევს საშუალებს მონაცემების მიხედვით გავზარდოთ გამოთვლითი რესურსი და მონაცემები მოქნილი ალგორითმებისა და პროგრამული კოდის დახმარებით დავამუშავოთ.



ნახ.2.9. ქსელში ჩართული IoT მოწყობილობები

მსგავსი გამოწვევების წინაშე დავდექით საქართველოს გადაუდებელი დახმარების ოპერატიული მართვის ცენტრი „112“-ის გუნდიც. ვინაიდან ყოველდღიურად დაახლოებით 25 000 ზარი ფიქსირდება, წლების განმავლობაში დაგროვილი ინფორმაციის რეალურ დროში დამუშავებას და საჭირო სტატისტიკური მონაცემების დათვლას ტრადიციული მონაცემთა ბაზა ვეღარ უმკლავდება. გამოთვლებს დიდი დრო სჭირდება და შესაბამისად იყო შემთხვევები, როდესაც აღნიშნული პროცესის გამო სხვა მთავარი ბიზნეს-პროცესები ფერხდებოდა [10,43].

მსგავსი პრობლემა დადგა, როდესაც ერთ-ერთმა ფიტნეს კლუბმა მოგვმართა დახმარებისათვის. ისინი იყენებენ ტრადიციულ რელაციურ მონაცემთა ბაზას და აქვთ მოძველებული ტიპის

მონოლითური საბაზო სისტემა, რომელიც მომხმარებლის მოთხოვნებს ვეღარ აკმაყოფილებს. საჭირო გახდა მათი მონაცემების დამუშავებისთვის ალტერნატიული ტექნოლოგიების მოძიება.

აღნიშნული გამოწვევების გადასალახად საჭირო იყო შემდეგი ნაბიჯების გადადგმა: აგვერჩია კომპიუტერული ტექნიკა რომელიც მოგვცემდა ჰორიზონტალური სკალირების შესაძლებლობას. გაგვემართა ოპერაციული სისტემა და მოგვემზადებინა მონაცემების შესანახად. ეკოსისტემის გამართვის შემდეგ შეგვექმნა პროგრამული კოდი, რომელიც მონაცემებს დაამუშავებდა და უკვე დამუშავებულ ინფორმაციას შეინახავდა „Data Warehouse“-ში.

112-ის შემთხვევაში შევქმენით *ვირტუალური მანქანები* VMware ვირტუალიზაციის პლატფორმაზე, სადაც დავაყენეთ მონაცემთა ბანკი, რომელშიც ვინახავთ უკვე დათვლილ მონაცემებს. ხოლო რაც შეეხება კლუბის პრობლემას, მის გადასაჭრელად დავწერეთ პროგრამული პროდუქტი, რომლის არქიტექტურაც დეტალურად გვაქვს აღწერილი წინამდებარე ნაშრომში.

წიგნის ფარგლებში მთავარი ამოცანა იყო ზემოაღნიშნული პრობლემების გადაწყვეტისთვის დაბალი ღირებულების ტექნოლოგიების შერჩევა. მიზნად დავისახეთ, შეგვემუშავებინა მონაცემთა შენახვა-დამუშავების ეკოსისტემის ისეთი არქიტექტურა, რომლის განხორციელებისას გამოყენებული იქნებოდა დაბალი ღირებულების როგორც კომპიუტერული, ასევე პროგრამული ტექნოლოგიები და მსურველები შეძლებდენ აღნიშნული არქიტექტურის გამოყენებას სასწავლო ან კომერციული მიზნებისთვის. აღნიშნული არქიტექტურა იმდენად მოქნილი უნდა ყოფილიყო, რომ საჭიროების შემთხვევაში შესაძლებელი გამხდარიყო სასწავლო პროექტის მასშტაბურ კომერციულ პროექტად გარდაქმნაც.

III თავი. ეკოსისტემის ინფრასტრუქტურის დაპროექტება

3.1. კვლევის მეთოდოლოგია

კვლევის მეთოდოლოგიად ავირჩიეთ არსებული ტექნოლოგიებით უკვე შემუშავებული გადაწყვეტილებების შესწავლა და მათი გაანალიზების შედეგად პროექტის მიზნის განხორციელება, რაც გულისხმობს მონაცემთა ეფექტური შენახვა-დამუშავების განაწილებული ეკოსისტემის აგების კონცეფციის შემუშავებას [44,45].

პროექტის განსახორციელებლად ვიყენებთ ჰორიზონტალური ზრდის მოდელს, რაც ითვლება ინოვაციურ და თანამედროვე მიდგომად. აღნიშნული მოდელი გულისხმობს სერვერების გაერთიანებით ერთიანი რესურსის შექმნას, რასაც გამოვიყენებთ ინფორმაციის შენახვა-დამუშავებისთვის. აღნიშნული მოდელის უპირატესობაა რესურსის დამატება საჭიროებისამებრ, მარტივად. ამასთან, აღნიშნული მოდელის გამოყენებისას, ტექნიკის დაზიანებისგან გამოწვეული ხარვეზები კრიტიკული არ არის და ადვილად გამოსწორებადია.

ჩვენი მიზნების გათვალისწინებით უნდა შევარჩიო დაბალი ღირებულების ტექნიკა, რომელსაც გამოვიყენებთ პროექტის ფარგლებში, აღნიშნული ტექნიკის გამოყენებით საგრძნობლად ვამცირებთ ხარჯებს. ტექნიკის ფასის და ხარისხის გათვალისწინებით, მოსალოდნელია ტექნიკის მწყობრიდან გამოსვლა. რაც უფრო დაბალია კომპიუტერული ტექნიკის ფასი, მით უფრო იზრდება მისი მწყობრიდან გამოსვლის რისკი. ჩვენი ამოცანაა, რისკების შესამცირებლად ისეთი არქიტექტურის შემუშვება, რომლის საშუალებითაც შევძლებთ, ყოველგვარი ჩავარდნების გარეშე შევცვალოთ დაზიანებული ტექნიკა. აღნიშნული მიზნის მიღწევა რთულია, მაგრამ შესაძლებელი. სისტემის სწორი პროექტირების შედეგად გამარტივდება ტექნიკის მოვლაც, რაც შეამცირებს ტექნიკის მოვლისთვის გათვალსწინებულ ხარჯებს.

3.2. კვლევის მოსალოდნელი შედეგების სამეცნიერო ღირებულება და მისი პრაქტიკული გამოყენება

როგორც აღვნიშნეთ, პროექტის მიზანია, თანამედროვე და რაც შეიძლება დაბალი ღირებულების ტექნოლოგიებით დავაპროექტო ისეთი ეკოსისტემა, რომელსაც გამოვიყენებთ ინორმაციის შენახვა-დამუშავებისთვის. წარმოდგენილი პროექტის უმნიშვნელოვანესი ეტაპია პროექტის ფარგლებში განსახორციელებელი პრაქტიკული სამუშაოები.

ჩატარებული კვლევის შედეგები მრავალმხრივ გამორჩეული იქნება ჩვენთვის - ერთი მხრივ, კვლევის მეთოდოლოგიის სრულყოფა, დისერტაციის დასრულება და წარდგენა ინფორმატიკის დოქტორის აკადემიური ხარისხის მოსაპოვებლად. მეორე მხრივ, მიღებული გამოცდილების აქტიურად გამოყენება, რაც დამეხმარება პროფესიულ წინსვლაში.

აღნიშნული ტექნოლოგიების გამოყენება თავისუფლად არის შესაძლებელი ისეთი კლასის ამოცანებში, როგორიცაა: ინფორმაციის დამუშავება და ანალიზი, მცირე ორგანიზაციებისთვის სატელეფონო ინფრასტრუქტურის გამართვა, მონიტორინგისა და ავტომატიზაციის სისტემების დანერვა და ა.შ.

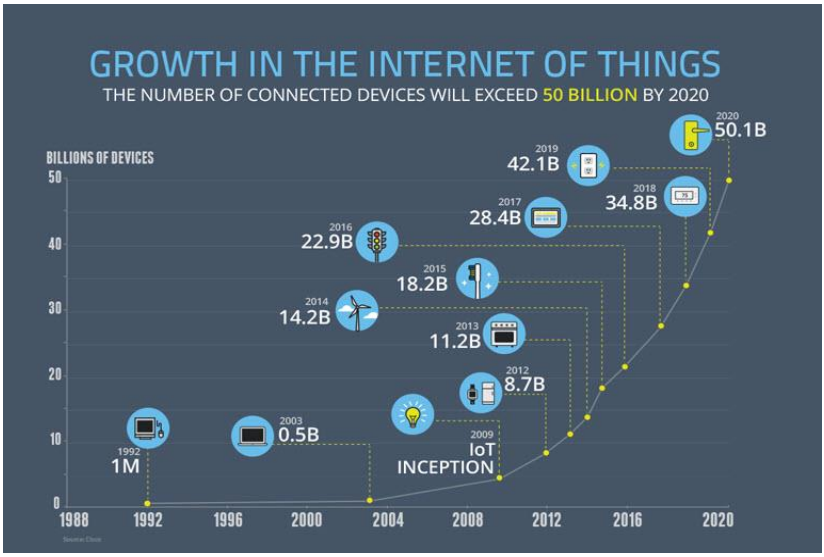
მიღწეული შედეგების გავრცელებას ვაპირებ საქართველოს ტექნიკური უნივერსიტეტის დოქტორანტურის ფარგლებში განხორციელებული აქტივობებით. ასევე, უზრუნველვყოფ მიღწეული შედეგების ინტერნეტში, ღიად განთავსებას, რაც, ვფიქრობ, დაეხმარება პროექტის საკვლევი თემით დაინტერესებულ ახალგაზრდა მკვლევარებს.

აღნიშნულ ტექნოლოგიებთან მუშაობის გამოცდილება საშუალებას მომცემს, პროექტის დასრულების შემდეგაც განვაგრძო ინფორმაციის შენახვა-დამუშავების თანამედროვე ტექნოლოგიების კვლევა. როგორც უკვე აღვნიშნე, პროექტის მიზნის მისაღწევად საჭიროა, შესრულდეს შემდეგი ამოცანები:

- აპარატურის მოძიება;
- ოპერაციული სისტემების მოძიება;
- ინფორმაციის შენახვის განაწილებული სისტემების მოძიება;
- განაწილებული გამოთვლითი სისტემების მოძიება;
- მოძიებული ტექნოლოგიების ერთმანეთთან დაკავშირება და მონაცემთა შენახვა-დამუშავების განაწილებული ეკოსისტემის პროექტირება.

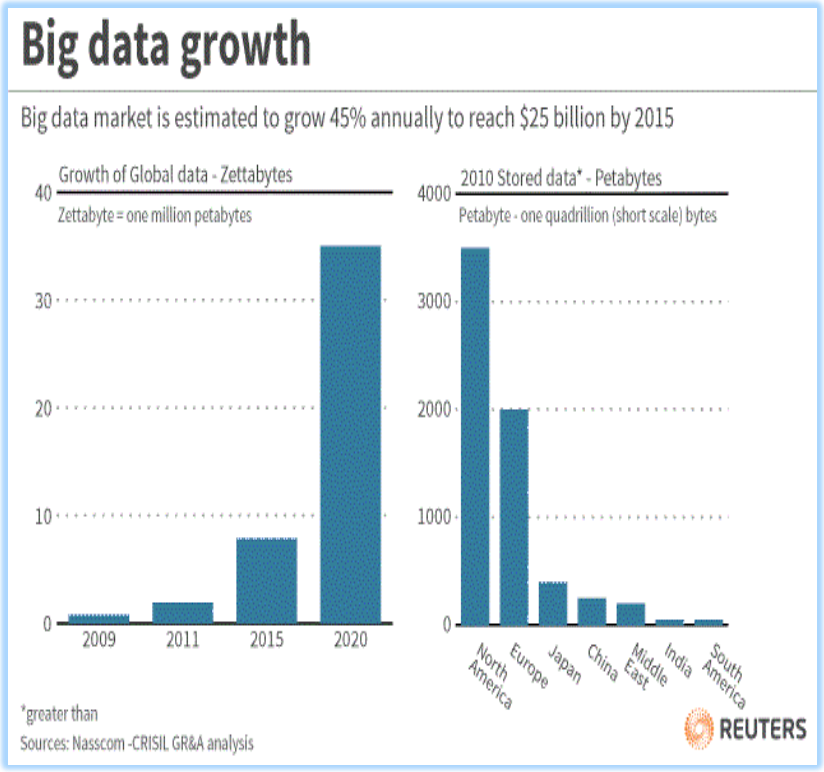
3.3. საპრობლემო სფეროს ინფორმაციის შენახვა-დამუშავება

ინფორმაციის შენახვა-დამუშავება თანამედროვე ეპოქის სირთულეა, რადგან დასამუშავებელი ინფორმაცია დღითიდღე მატულობს. ყოველდღიურად იზრდება ინტერნეტ მომხმარებლების რაოდენობავ, თანამედროვე ტექნოლოგიების დახმარებით შესაძლებელია ქსელში ჩართული იყოს არა მხოლოდ კომპიუტერები, არამედ ყველა საყოფაცხოვრებო მოწყობილობაც (ნახ.3.1) [46].



ნახ.3.10. ქსელში ჩართული მოწყობილობების რაოდენობა

3.1 ნახაზზე ნაჩვენებია ქსელში ჩართული მოწყობილობების რაოდენობის ზრდა დროის მიხედვით. ეს რაოდენობა 2020 წლისთვის 50 მილიარდ მოწყობილობამდე გაიზრდება. შესაბამისად გაიზრდება ამ მოწყობილობების მიერ შექმნილი მონაცემები, რომლებიც, ორგანიზაცია „NASSCOM“-ის კვლევის თანახმად, ზეტაბაიტებს მიაღწევს [47] (ნახ.3.2).



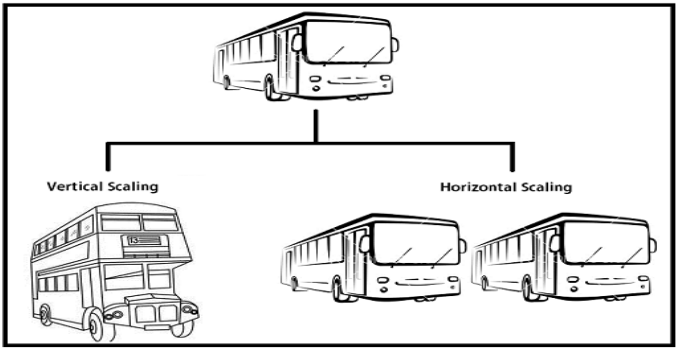
ნახ.3.2.. მონაცემების ზრდა

მონაცემთა დამუშავება ტრადიციული რელაციური ბაზების მოდელით თითქმის შეუძლებელია, რადგან აღნიშნული მოდელით მონაცემების მატებასთან ერთად საჭირო გახდება აპარატურული რესურსის გაზრდა, რაც არის ვერტიკალური ზრდის მოდელი.

ასეთი მოდელი შეზღუდულია, რადგან არსებული ტექნოლოგიების განვითარება მონაცემების ზრდასთან შედარებით გაცილებით ნაკლებია. შესაბამისად, არსებული ინფორმაციის დასამუშავებლად საჭირო გახდა ალტერნატიული გზების მოძიება.

3.4. ჰორიზონტალური ზრდა

ჰორიზონტალური ზრდა არის სწორედ ის ალტერნატიული გზა, რომელიც *დიდი მონაცემების* დამუშავების საშუალებას იძლევა [45]. ჰორიზონტალური ზრდის მოდელი საშუალებას გვაძლევს მონაცემების დამუშავებისთვის გამოვიყენოთ ერთდროულად რამდენიმე სერვერი, დასამუშავებილი ინფორმაცია ნაწილდება სერვერების რაოდენობაზე და ინფორმაციის დამუშავების სიჩქარე ბევრად გაიზრდება (ნახ.3.3).



ნახ.3.3..ჰორიზონტალური სკალირება

ნახაზზე მოყვანილია ვერტიკალური და ჰორიზონტალური ზრდის მაგალითები, სადაც ცალსახად ჩანს ჰორიზონტალური ზრდის უპირატესობა. როდესაც დაგვჭირდება მეტი გამოთვლითი სიმძლავრე, არ მოგვიწევს მთელი არქიტექტურის შეცვლა. საკმარისია, მაგალითად, ათასი სერვერის დამატება. რაც ბევრად უფრო მოსახერხებელია ვერტიკალურ ზრდასთან შედარებით. ვერტიკალური ზრდა შესაძლებელია სერვერის მონაცემების

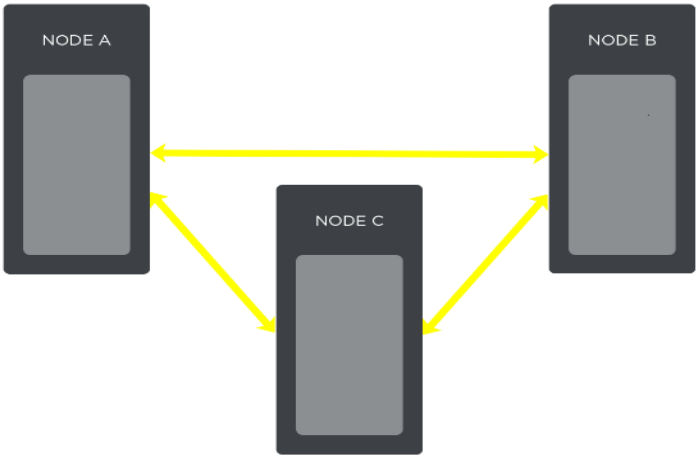
დეტალების გამოცვლით ან დამატებით, რასაც თან არაერთი პრობლემა ახლავს: უნდა გავითვალისწინოთ არსებული აპარატურის გაძლიერების შესაძლებლობა და შევამოწმოთ ახალი დეტალების თავსებადობა რასაც საბოლოოდ ერთ კონკრეტულ ბრენდამდე მივყავართ, რის შედეგადაც ჩვენ ვხდებით კონკრეტულ მწარმოებელზე დამოკიდებული. ხშირ შემთხვევაში შესაძლოა მთლიანი აპარატურის გამოცვლაც კი გახდეს საჭირო.

თანამედროვე ტექნოლოგიები, რომლებიც ორიენტრებულია ჰორზონტალურ ზრდაზე, საშუალებას გვაძლევს, დავივიწყოთ აპარატურის თავსებადობის მსგავსი პრობლემები. თანამედროვე ტექნოლოგიების საშუალებით შესაძლებელია მონაცემების შენახვა-დამუშავების ერთიანი ეკოსისტემის სხვადასხვა მწარმოებლის და მოდელის აპარატურისგან შექმნა. ეს ყველაფერი საერთო ჯამში ბევრად მოსახერხებელი და დაბალ ბიუჯეტიანი გადაწყვეტილებაა.

3.4.1. მონაცემების შენახვა-დამუშავების განაწილებული სისტემები: „Apache Hadoop“ და „Mongodb“

მონაცემების შენახვა-დამუშავების განაწილებული სისტემები გვაძლევს საშუალებას, საჭიროებისამებრ გამოვიყენოთ აპარატურული რესურსი. იმისათვის, რომ აღნიშნულმა სისტემებმა ფუნქციონირება შეძლონ, აუცილებელია შემდეგი წინაპირობების არსებობა [44]:

- კომპიუტერული ტექნიკა, რომელზეც გაიმართება ოპერაციული სისტემა;
- ოპერაციული სისტემა (რეკომენდებულია Linux ოპერაციული სისტემა);
- ქსელური ინფრასტრუქტურა: კომპიუტერული ტექნიკა, რომელზე გამართულ სისტემებსაც უნდა შეეძლოს ერთმანეთთან კავშირის დამყარება და მონაცემების გაცვლა (ნახ.3.4).



ნახ.3.4. მონაცემების შენახვა-დამუშავების განაწილებული სისტემები

3.5. პროექტის ფარგლებში მოძიებული ტექნიკა

წიგნის ფარგლებში მოძიებული გვაქვს ტექნიკა, რომელიც დააკმაყოფილებს პროექტის მინიმალურ მოთხოვნებს:

- Raspberry Pi 3 Model B (მინი კომპიუტერი, რომელსაც უნდა მოვარგოთ ოპერაციული სისტემა. აღნიშნული ტექნიკა გვაძლევს საშუალებას, დავაპროექტოთ და შევქმნათ ახალი პროდუქტები);
- Micro SD (მიკრო ჩიპი რომელიც დამონტაჟდება მინიკომპიუტერში და მასში ჩაიწერება მინი კომპიუტერისთვის გათვალისწინებული ოპერაციული სისტემა);
- Power HuB (ელექტრო ენერგიის წყარო, რომელზეც დაერთდება მინი კომპიუტერები);
- External hard drive (გარე მყარი მეხსიერება, რომელსაც გამოვიყენებ პროექტის ფარგლებში);

- Anker PowerLine Micro USB (1ft) (მინი კომპიუტერისთვის 30 სანტიმეტრის სიგრძის ელექტრო სადენი);
- Anker PowerLine Micro USB (3ft) (მინი კომპიუტერისთვის 90 სანტიმეტრის სიგრძის ელექტრო სადენი);
- Patch Cord 1.5ft (მინი კომპიუტერისთვის 45 სანტიმეტრის სიგრძის კომპიუტერული ქსელის სადენი);
- Patch Cord 2 ft (მინი კომპიუტერისთვის 60 სანტიმეტრის სიგრძის კომპიუტერული ქსელის სადენი);
- Switch (ქსელის კომუტატორი მინიკომპიუტერებისთვის).

ზემოთ ჩამოთვლილი აპარატურის გამოყენებით მოხდება ინფორმაციის შენახვა-დამუშავების გაერთიანებული ეკოსისტემის პროექტირება.

აღნიშნული პროექტის განხორციელებისთვის საჭიროა დამატებითი ტექნიკის შეძენაც, რასაც გამოვიყენებთ ეკოსისტემის მონიტორინგის, გაგრილებისა და ტესტირებისთვის:

- ASUS PRIME Z270-A (დედა პლატა, რომელსაც უნდა ჰქონდეს მინიმუმ 16 GB ოპერატიული მეხსიერებისა და ახალი ტიპის მყარი მეხსიერების M.2 დაერთების მხარდაჭერა);
- Intel i7 (პროცესორი);
- Corsair Dominator Platinum Series 16GB (ოპერატიული მეხსიერება რომელიც უნდა იყოს მინიმუმ 16 GB ზომის);
- Samsung 960 EVO Series - 500GB NVMe (ახალი თაობის HD);
- Corsair RMx Series, RM850x (კვების ბლოკი, რომელიც უნდა იყოს მოდულარული, შეეძლოს გაგრილების სისტემის დაერთება);
- Coolers (ჰაერით გაგრილების სისტემა).

პროექტის ფარგლებში აპარატურასთან ერთად მოძიებული გვაქვს ოპერაციული სისტემები, კერძოდ განხილულია linux-ის დადებითი და უაროფითი მხარეები, რის საფუძველზეც ავარჩიეთ ოპერაციული სისტემები. ამ მიმართულებით წიგნის თანაავტორმა (ბ. კახელმა) გაიარა შემდეგი ტრენინგები:

- VS-ICM VMware vSphere ინსტალაცია, კონფიგურაცია, მართვა. (საქართველოს შინაგან საქმეთა სამინისტროს სსიპ '112'-ის მხარდაჭერით);
- Oracle DBA Advanced Topics Training course. (საქართველოს შინაგან საქმეთა სამინისტროს სსიპ '112'-ის მხარდაჭერით);
- Linux Fundamentals and Administration (საქართველოს ინოვაციების და ტექნოლოგიების სააგენტოს მხარდაჭერით);
- Asterisk Advanced (ქ. დუბაი, კომპანია Digium-ის ოფიციალურ ტრეინინგ ცენტრი. საქ. შინაგან საქმეთა სამინისტროს სსიპ '112'-ის მხარდაჭერით);
- M102: MongoDB for DBAs (Mongo_უნივ-ის მხარდაჭერით);
- M202: MongoDB Advanced Deployment and Operations (Mongo უნივ.);
- M310: MongoDB Security (Mongo_უნივ.);
- M312: Diagnostics and Debugging (Mongo_უნივ.);
- M201: MongoDB Performance (Mongo_უნივ.).

როგორც უკვე აღვნიშნეთ, მონაცემთა შენახვა-დამუშავების ეკოსისტემის ფორმირებისთვის აუცილებელი ელემენტები მოძიებული და შესწავლილია. აღნიშნული კვლევის შედეგად წარმოდგენილია პრაქტიკული სამუშაოები, რომლებიც ორიენტირებულია სხვადასხვა მიზნის მიღწევაზე.

3.6. Raspberry Pi მოწყობილობაზე ოპერაციული სისტემის გამართვა

გადმოვიწეროთ ოპერაციული სისტემის შესაბამისი საინსტალაციო პაკეტები ჩვენი შერჩეული მოწყობილობისთვის რასპერი პი 3-სთვის (ნახ.3.5) [16]. შეგვიძლია გადმოვწეროთ შემდეგი სახის საინსტალაციო პაკეტი: „DESKTOP", რომელიც გათვლილია მომხმარებლებზე, ვისაც ესაჭიროებათ გრაფიკული ხელსაწყოები ოპერაციულ სისტემებთან სამუშაოდ. „LITE", ვერსია სადაც არის

დაყენებული მინიმალური ხელსაწყოების ნაკრები და გათვლილია ტექსტური ბრძანებების შეყვანის რეჟიმზე მუშაობისთვის.

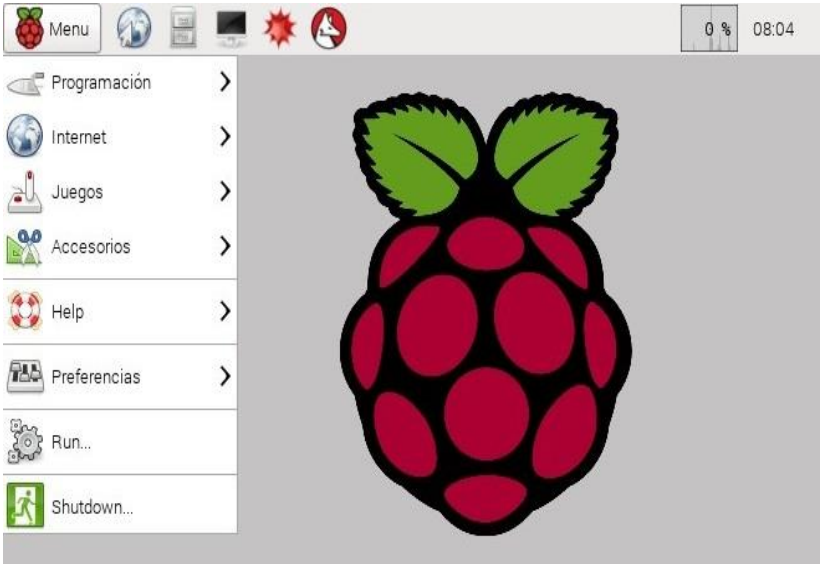
ნახ.3.5. Raspbian ოპერაციული სისტემა

მას შემდეგ, რაც ავარჩევთ და გადმოვწერთ ჩვენთვის სასურველ საინსტალაციო პაკეტს, უნდა მოვძებნოთ და დავაყენოთ შემდეგი პროგრამული უზრუნველყოფა „Win32 Disk Imager“, რომელსაც გამოვიყენებთ გადმოწერილი საინსტალაციო პაკეტის ჩიპზე ჩაწერისთვის (ნახ.3.6).

ნახ.3.6. Win32 პროგრამა

შემდეგ, როდესაც ეს პროგრამული უზრუნველყოფა დაასრულებს გადმოწერილი ფაილების ჩიპზე ჩაწერას, ჩიპი უნდა დავამონტაჟოთ „Raspberry Pi 3"_ში.

რასპერი პი 3_ზე ოპერაციული სისტემის გამართვამდე მასზე აუცილებლად უნდა გვქონდეს შეერთებული: მონიტორი, კლავიატურა და მაუსი. თუ ყველა საჭირო მოწყობილობა შეერთებულია, ვრთავთ მოწყობილობას და ვიწყებთ ოპერაციული სისტემის კონფიგურაციას. ამ პროცესში ვირჩევთ ოპერაციული სისტემის ინსტალაციას, დავარქმევთ კვანძს (host) სახელს, ვირჩევთ დროის სალტეს და ა.შ. სისტემის გამართვის შემდეგ ჩაიტვირთება ოპერაციული სისტემა (ნახ.3.7 და 3.8).



ნახ.3.7..Raspbian ლინუქსის Desktop ვერსია

„DESKTOP" ვერსია, რომელშიც ჩაწერილია გრაფიკული ხელსაწყოების ნაკრები.

„LITE" ვერსია, გათვლილია ტექსტური ბრძანებების რეჟიმში მუშაობისთვის.

ნახ.3.8.. Raspbian ლინუქსის LITE ვერსია

ოპერაციულ სისტემაში შესასვლელად მომხმარებლის ველში უნდა ჩავწეროთ „pi“ ხოლო პაროლის ველში „raspberry“.

ინფორმაციული უსაფრთხოების გათვალისწინებით აუცილებელია სისტემაში შესვლის შემდეგ პაროლის გამოცვლა.

„DESKTOP” ვერსიის შემთხვევაში ვაწვებით კლავიშების შემდეგ კომბინაციას „ctr+alt+t“, რის შემდეგაც გაიხსნება ტერმინალის ფანჯარა, სადაც შევძლებთ ტექსტური ბრანებების გაშვებას. გავუშვათ ტერმინალში „passwd” ბრძანება, რის შემდეგაც,

სისტემა მოგვთხოვს ახალი პაროლის შეყვანას. პაროლის გამოცვლის შემდეგ, აუცილებელია გავუშვათ სისტემის განახლება.

ტერმინალში შემდეგი ბრძანებების, „sudo apt-get update & sudo apt-get upgrade“ გაშვების შემდეგ ოპერაციული სისტემა მოძებნის და დააყენებს ყველა საჭირო განახლებას, რაც შეასწორებს და აღმოფხვრის ყველა ცნობილ პროგრამულ ხარვეზს. ეს ჩვენს ოპერაციულ სისტემას შედარებით უფრო მეტად სტაბილურს და საიმედოს გახდის.

მას შემდეგ რაც პაროლი შევცვალეთ და განახლებები დავაყენეთ, აუცილებელია სისტემაზე გარედან წვდომის მექანიზმის გააქტიურება, ამისთვის ტერმინალში უნდა გავუშვათ შემდეგი ბრძანება „sudo service ssh start“.

იმისათვის, რომ აღნიშნული სერვისი სისტემის გადატვირთვის შემდეგ ავტომატურად გაეშვას, აუცილებელია „/boot“ დირექტორიაში შევქმნათ სერვისის ავტომატურად გამშვები ფაილი ბრძანებით „sudo touch /boot/ssh“.

აღნიშნული სერვისი გვაძლევს ოპერაციული სისტემის ტერმინალთან წვდომას „IP” პროტოკოლის დახმარებით. მას შემდეგ, რაც სერვისი გააქტიურდება, შევძლებთ ოპერაციულ სისტემაში შესვლას და მის მართვას ტექსტური ბრძანებებით, ნებისმიერი კომპიუტერიდან. ამისთვის საჭიროა ვიცოდეთ: „IP“ მისამართი, მომხმარებლის სახელი და პაროლი.

ამ სერვისის სარგებლობისას შესაძლებელია მხოლოდ ტექსტური ბრძანებების რეჟიმში მუშაობა. მათ, ვისაც ესაჭიროება გრაფიკულ ინტერფეისთან კავშირის დამყარება, შეუძლიათ გამართონ „VNC” ან „TeamViewer“ სერვისი და მათი მეშვეობით დაუკავშირდნენ სერვერს [48,49].

აღნიშნული ბიჯების შესრულების შემდეგ სახეზე გვექნება სუფთა სერვერი, რომლის გამოყენებაც შესაძლებელი იქნება ჩვენი წიგნის მომდევნო (პრაქტიკულ) ნაწილში. ნაშრომის მიზნებიდან გამომდინარე, რაც გულისხმობს მონაცემების შენახვა-დამუშავების

განაწილებული ეკოსისტემის პროექტირებას. ჩვენ გადავწყვიტეთ პრაქტიკული სამუშაოები ჩავატაროთ ორი მიმართულებით: მონაცემების შენახვის განაწილებული გარემოს შექმნა და მონაცემების დამუშავების განაწილებული გარემოს გამართვა. ამ ყველაფრისთვის გამოვიყენებთ ზემოაღწერილ ტექნიკას და ლინუქსის ტიპის ოპერაციულ სისტემას „RAID“ მექანიზმებით.

3.7. დიდი ზომის მონაცემთა შენახვა მობილურ აპლიკაციებში

პროექტში გამოიყენება ინფორმაციის შენახვისა და დამუშავების თანამედროვე მიდგომები. წიგნში მოკლედ არის აღწერილი ჩატარებული სამუშაოები.

ინფორმაციის ლოკალურად ჩაწერა აპლიკაციას სასურველი ინფორმაციის offline რეჟიმში მიღების ან შენახვის საშუალებას აძლევს.

აპლიკაციის საჭიროებიდან გამომდინარე, ანდროიდის ფაილური სისტემა SQLite მონაცემთა ბაზის გამოყენებით იძლევა მონაცემთა შენახვის მდგრად მექანიზმს [50].

SQLite-ის გამოყენებით შესაძლებელია რელაციური მონაცემთა ბაზის შექმნა, რომელიც გამოყენებული იქნება ინფორმაციის სტრუქტურირებულად ჩაწერისა და შენახვისთვის. SQLite არის მსუბუქი მონაცემთა ბაზა და შედგება ფაილური სისტემისგან, რომელიც შენახულია მობილურის საქაღალდეში:

/data/data/<package__name/databases>.

მათზე სრული უფლებებით წვდომა მხოლოდ იმ აპლიკაციას შეუძლია, რომელმაც აღნიშნული საქაღალდე შექმნა. ასევე შესაძლებელია ინფორმაციაზე წვდომის მიცემა სხვა აპლიკაციისთვისაც.

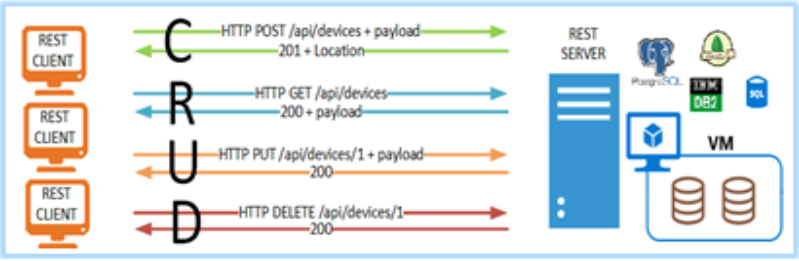
SQLite მონაცემთა ბაზებთან სამუშაოდ ანდროიდ სისტემის SDK მრავალ საჭირო კლასს შეიცავს, ამ კლასთა უმეტესობა android.database.sqlite package-ში მდებარეობს.

აქაა განთავსებული ისეთი კლასები, რომელთა საშუალებითაც შეიქმნება მონაცემთა ბაზა, გაკონტროლდება მათი ვერსიები, აიგება SQL ბრძანებები და ა.შ. მათ შორისაა სპეციალური Cursor კლასი, რომელიც ბაზიდან ამოღებული ინფორმაციის დამუშავებას უზრუნველყოფს. აქვეა იმ შეცდომების დამუშავების საშუალებაც, რომელიც შესაძლოა დაფიქსირდეს SQLite-ის მონაცემთა ბაზასთან მუშაობისას. მონაცემთა დაცვისა და უსაფრთხოების გაძლიერებისთვის, შესაძლებელია დაშიფრული ფაილების მქონე SQLite მონაცემთა ბაზის შექმნა [51].

SQLite მონაცემთა ბაზის შესაქმნელად რამდენიმე გზა არსებობს. ყველაზე მარტივია openOrCreateDatabase მეთოდის გამოყენება SQLiteDatabase კლასიდან.

გადაუდებელი დახმარების ოპერატიული მართვის ცენტრის მიერ დასმული ამოცანის მიხედვით აუცილებელია ინფორმაციის შენახვა მის მფლობელობაში არსებულ მონაცემთა ბაზების სერვერებზე. მობილურ აპლიკაციასა და „112“-ის მონაცემთა ბაზასთან კავშირისთვის გამოიყენება ASP.NET Web API სერვისები.

ASP.NET Web API წარმოადგენს framework-ს, რომლის საშუალებითაც იქმნება HTTP სერვისები. მათი გამოყენება შესაძლებელია მრავალი მომხმარებლის, მათ შორის browser-ების და მობილური მოწყობილობების მიერ. ASP.NET Web API იდეალური პლატფორმაა RESTful აპლიკაციების შესაქმნელად (ნახ.3.10) [52-54].



ნახ.3.9. RESTful API სერვისის დანიშნულება

სერვისებზე წვდომისთვის აუცილებელია კავშირი ინტერნეტთან. იმისათვის, რომ შექმნილი აპლიკაცია არ იყოს ინტერნეტზე დამოკიდებული და ჰქონდეს მუდმივი წვდომა „112“-ის სერვისებთან, GSM პროვაიდერების მეშვეობით შეიქმნა ალტერნატიული არხი. „112“-ის მიმართულებით გაგზავნილი მოკლე ტექსტური შეტყობინება ალტერნატიული გზის გავლით მიღწეული იქნება აღნიშნულ სერვისებთან. შედეგად, აპლიკაციას „112“-თან მუდმივი წვდომა ექნება და გადაუდებელი დახმარების სიტუაციაში მყოფი ადამიანი დროულად მიიღებს დახმარებას [41].

გადაუდებელი დახმარების ოპერატიული მართვის ცენტრისთვის მომხმარებლებიდან მიღებული ინფორმაცია ერთ-ერთი ყველაზე ძვირადღირებული და აუცილებელი კომპონენტია. შესაბამისად, გაჩნდა ინფორმაციასთან წვდომისა და შენახვის საიმედოობის ამაღლების საჭიროება. რამაც წინა პლანზე წამოწია მონაცემთა დუბლირების საკითხი RAID (redundant array of independent disks) მასივის გამოყენებით. ინფორმაციის მოცულობისა და რაოდენობის ექსპონენციალურმა ზრდამ საჭირო გახადა დიდი ზომის მონაცემების შენახვა-დამუშავება რეალურ დროში. აღნიშნულმა მოთხოვნამ თანამედროვე ტექნოლოგიები ახალი გამოწვევების წინაშე დააყენა. მოთხოვნების რეალურ დროში დასამუშავებლად შეიქმნა განაწილებული სისტემები: Hadoop, Spark, MongoDB და სხვ. [45].

დიდი მოცულობის მონაცემების შემნახავი სისტემის დასაპროექტებლად საჭიროა წინასწარი ცოდნა იმის შესახებ, თუ რა მონაცემებს შევინახავთ და როგორი მოთხოვნებით უნდა ინახებოდეს ინფორმაცია. ეს მოთხოვნები შეიძლება იყოს: დიდი მოცულობის ინფორმაციის ჩაწერა, წაკითხვა, საიმედოობა, სწრაფქმედება [9].

არსებული მოთხოვნების დასაკმაყოფილებლად შეიქმნა RAID მასივი. RAID მასივი მონაცემთა შენახვის *ვირტუალიზაციის ტექნოლოგიაა* და დამოუკიდებელ მყარი დისკოებს მონაცემთა

ჭარბი მასივით ერთ ლოგიკურ ბლოკში აერთიანებს. ლოგიკური ბლოკის მიზანია საიმედოობის და წარმადობის სრულყოფა.

საიმედოობა მიიღწევა დუბლირების გზით, წარმადობის გასაუმჯობესებლად კი ინფორმაციის დამუშავება პარალელურად, სხვადასხვა დისკოებიდან ხდება. შეიქმნა RAID მასივის განსხვავებული არქიტექტურები: RAID_0 -:- RAID_10. მასივის სხვადასხვა ტიპები ძირითადად ორ მიზანს ემსახურება: მონაცემთა შენახვის საიმედოობისა და დისკოზე ჩაწერა/წაკითხვის სისწრაფის გაზრდას.

აღნიშნული ამოცანის გადაწყვეტის მიზნითა და არსებული ტექნოლოგიების ანალიზის გათვალისწინებით, შემუშავებულ იქნა ინფორმაციის შენახვის ჩვენთვის მისაღები არქიტექტურა, რომელიც შემდეგი ეტაპებისაგან შედგება:

- აპარატურის მოდელირება;
- ოპერაციული სისტემის გამართვა;
- მონაცემთა მართვის სისტემის გამართვა (მონაცემთა ბაზა).

სერვერის შერჩევისას არჩეულ იქნა აპარატურა RAID კონტროლერი „Hot Swap“-ის მხარდაჭერით. მათ უერთდება მინიმუმ ორი დისკო, რომლებიც მუშაობს RAID0 („Striping“) არქიტექტურით. RAID მასივის გამოყენება გაზრდის ინფორმაციის დამუშავების სისწრაფეს.

აპარატურის გამართვის შემდეგ დაინსტალირდა „Linux“ ოპერაციული სისტემა. როგორც ცნობილია, იგი UNIX ოპერაციული სისტემის ბაზაზე შეიქმნა, თავისუფლად ვრცელდება და დამყარებულია GPL ლიცენზიაზე. ლინუქს ოპერაციული სისტემის ბირთვი 22 მილიონი სტრიქონის მოცულობის კოდისაგან შედგება, შესაბამისად საკმაოდ მცირე რესურსს მოითხოვს [55]:

- პროცესორი 400 Mhz;
- მეხსიერება მყარ დისკოზე 10 MB.
- ოპერატიული მეხსიერება 256 MB;

აპარატული და პროგრამული რესურსის მინიმალური მოხმარება ოპერაციულ სისტემას მომხმარებლისთვის ბევრად უფრო საიმედოსა და მოსახერხებელს ხდის. ლინუქს ოპერაციული სისტემა ასევე გამოირჩევა ისეთი მძლავრი თავდაცვითი მექანიზმით, როგორიცაა „Firewall“. მონაცემების მართვის სისტემად არჩეულ იქნა თანამედროვე ტექნოლოგიებზე დაფუძნებული მონაცემთა ბაზა, რომლის გამოყენებითაც RAID_0 არქიტექტურას საიმედოობა დაემატა. ამ ამოცანის ფარგლებში გამოყენებულ იქნა NoSQL ტიპის მონაცემთა ბაზის MongoDB-ს რეპლიკა სეტი, რომელიც იძლევა მაღალ საიმედოობას.

აღნიშნული არქიტექტურა რამდენიმე ელემენტისგან შედგება:

- *მობილური აპლიკაცია* - შესაძლებელს ხდის კომუნიკაციას „112“-ის ოპერატორებთან;
- *შუალედური სერვერი* - აკავშირებს მობილურ აპლიკაციას მონაცემთა ბაზასთან;
- *მონაცემთა გაზა* - ქსელური კავშირი, რომელიც უზრუნველ-ყოფს ინფორმაციის მიმოცვლას.

თანამედროვე ინფორმაციული ტექნოლოგიები საშუალებას იძლევა შეიქმნას ალტერნატიული დაკავშირების არხი გადაუდე-ბელი დახმარების სამსახურთან („112“), რაც გაუმარტივებს ცენტრის თანამშრომლებს ზარის ინიციატორისთვის დახმარების სწრაფად გაწევას. შედეგად მიიღება ალტერნატიული არხი „112“-თან კომუნიკაციისთვის და შესაძლებელი იქნება დამატებითი სიცოცხლის გადარჩენა.

აღნიშნული შედეგის მისაღწევად გამოყენებულ იქნა RAID_0 მასივი, რომლის საშუალებითაც გავზარდეთ ინფორმაციის ჩაწერის და წაკითხვის სისწრაფე. „MongoDB“ რეპლიკა სეტის გასამართად საჭიროა მინიმუმ სამი სერვერი. რეპლიკა სეტის დახმარებით ხდება მონაცემების სამ სხვადასხვა სერვერზე შენახვა, რომლებიც შესაძლებელია განთავსებული იყოს სხვადასხვა გეოგრაფიულ

წერტილში. რაც ჩვენ მონაცემებს იცავს არა მხოლოდ აპარატურული, არამედ გლობალური საფრთხეებისგანაც.

მობილური აპლიკაციის სახით მივიღეთ მდგრადი პროგრამული პროდუქტი, რომელსაც შეუძლია იმუშაოს როგორც „online“, ასევე „offline“ რეჟიმში. „offline“ რეჟიმში მუშაობის საშუალებას ანდროიდის „SDK“-ში ჩაშენებული მძლავრი, მსუბუქი და საიმედო „SQLite“ რელაციური მონაცემთა ბაზა გვაძლევს, სადაც ინფორმაცია საჭირო რელაციური სტრუქტურით ინახება.

3.8. ღრუბლოვანი ინფრასტრუქტურის გამართვა Raspberry Pi გამოყენებით

პროექტის ფარგლებში ჩატარებული კვლევების შედეგად შევარჩიეთ როგორც აპარატურული ასევე პროგრამული უზრუნველყოფები, რომელთა მეშვეობითაც დავაპროექტეთ და გავმართეთ ინფორმაციის შენახვა დამუშავების განაწილებული სისტემა. ამისთვის გამოვიყენეთ შემდეგი ტექნოლოგიები: აპარატურული „Raspberry Pi 2“, „Kubernetes“ და „ Fabric8“ პროგრამული უზრუნველყოფები [55].

Fabric8 არის ღია პლატფორმა, ბაზირებული „Docker, Kubernetes“ პლატფორმებზე რომელიც გვაძლევს საშუალებას შევქმნათ და ვმართოთ ჩვენი მიკროსერვისები დისტრიბუციულ გარემოში. „Open Shift“ ეს არის ცნობილი კომპანიის „Red Hat“_ის პროგრამული პროდუქტი, რომელსაც შეუძლია მუშობა „container-based“ პროგრამულ პროდუქტებთან და მათი მართვა. მას აქვს შემდეგი დისტრიბუციული სისტემების მხარდაჭერა : „Kubernetes, Docker“. იგი არის შექმნილი Docker-ის კონტეინერების და Kubernetes ის კასეტური მენეჯმენტის მაგალითებზე. მისი საწყისი კოდი ხელმისაწვდომია ინტერნეტში და ვრცელდება Apache ლიცენზიის ვერსია 2.0 _ის ფარგლებში [56, 57].

„Kubernetes“ არის ღია სისტემა, რომელიც კონტეინერიზე-ბული აპლიკაციების განლაგების, სკალირებისა და მართვის

ავტომატიზებას ახდენს, იგი თავდაპირველად შეიქმნა Google-ის მიერ და ახლა მის განვითარებაზე ზრუნავს „Cloud Native Computing Foundation“. მისი მიზანია პლატფორმების გაშვების, სკალირებისა და ოპერაციების ავტომატიზება პლატფორმების კლასტერებზე. ის იყენებს მთელი რიგი კონტეინერებზე ორიენტირებულ ინსტრუმენტებს, მათ შორის Docker-ს [58, 59].

პრაქტიკული სამუშაოსთვის აუცილებელია შემდეგი:

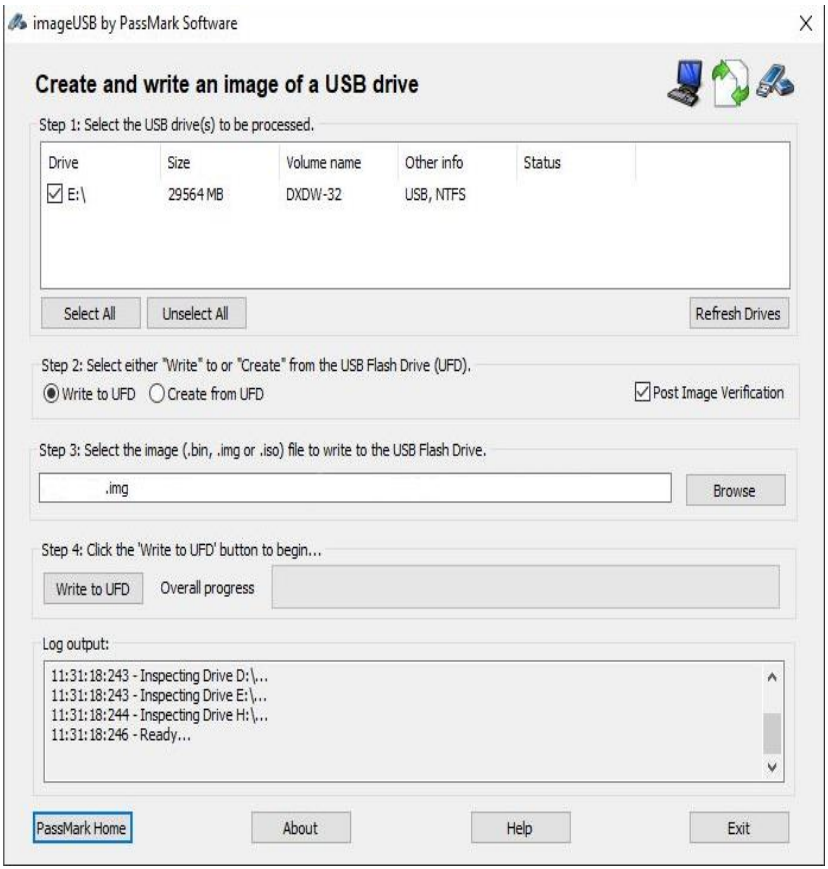
- 4 ცალი Raspberry Pi;
- 4 ცალი 16GB MicroSD card (Class 10);
- 1 ცალი 60 ვატიანი კვების წყარო;
- 4 ცალი Micro Usb კაბელი კვების წყაროზე შესაერთებლად;
- 5 ცალი Cat 5 ტიპის ქსელის კაბელი;
- 1 ცალი ქსელური კომუტატორი რომლის გამტარიანობაც თითოეულ ბორტზე მინიმუმ 100MBps იქნება.

აუცილებელია მიკროჩიპები იყოს Class 10, რადგან მასზე იქნება დამოკიდებული მთლიანი სისტემის სიჩქარე. იმისათვის, რომ „ბოთლის შევიწროებული ყელის“ (“Bottleneck”) პრობლემა შევამციროთ, აუცილებელია მაღალი წარმადობის ჩიპების შერჩევა.

- **მიკრო ჩიპების გამზადება:**

გადმოვიწეროთ და ჩიპზე ჩავწეროთ „Hyprito Docker image for Raspberry Pi“. „Hyprito“ არის ლინუქსის დებიანი დისტრიბუციის გადაკეთებული ვერსია, რომელშიც გათვალისწინებულია ყველა ის საჭირო დეტალი, რომელიც დაგვჭირდება განაწილებულ გარემოსთან სამუშაოდ.

მას შემდეგ, რაც ავარჩევთ და გადმოვწერთ ჩვენთვის სასურველ საინსტალაციო პაკეტს, უნდა მოვძებნოთ და დავაყენოთ შემდეგი პროგრამული უზრუნველყოფა „Win32 Disk Imager“, რომელსაც გამოვიყენებთ გადმოწერილი საინსტალაციო პაკეტის ჩიპზე ჩასაწერად (ნახ.3.10).



ნახ.3.10. საინსტალაციო პაკეტის SD ბარათზე ჩაწერა

მას შემდეგ, რაც აღნიშნული პროგრამული უზრუნველყოფა დაასრულებს გადმოწერილი ფაილების ჩიპზე ჩაწერას, ჩიპი უნდა გავამზადოთ რასპერი პი 2_ში ჩასამონტაჟებლად.

გადმოწერილი იმიჯის ჩიპებზე ჩაწერის შემდეგ ვაერთებთ მათ რასპერი პიში და ვრთავთ მოწყობილობას. იმისათვის რომ ჩვენმა კლასტერმა იმუშაოს, აუცილებელია, გვქონდეს DNS სერვერი, რომელსაც გაუწერთ ყველა მოწყობილობას და თავად სერვერში გავწერთ ყველა მოწყობილობის IP მისამართს და ჰოსტის

სახელს. იმისათვის, რომ დავზოგოთ ტექნიკა და არ გამოვიყენოთ დამატებითი სერვერი. ჩვენ შეგვიძლია თითოეულ მოწყობილობაში „/etc/hosts“ ფაილში გავწეროთ DNS ჩანაწერები.

მართალია სათითაოდ ყველა ჰოსტზე მოგვიწევს ერთიდა-იმავე პარამეტრების გაწერა, რაც შეცდომის დაშვების დიდი რისკია, მაგრამ ამით ვიცილებთ თავიდან დამატებითი სერვერის გამართვის და მოვლის აუცილებლობას.

მოწყობილობების ჩართვის შემდეგ, ოპერაციულ სისტემაში შევდივართ „root“ მომხმარებლის სახელით და „hypriot“ პაროლით.

სისტემაში შესვლის შემდეგ აუცილებელია გადავარქვათ ჰოსტებს სახელები ასეთი თანმიმდევრობით:

„Master , Node-1, Node-2, Node-3“.

მას შემდეგ, რაც გადავარქმევთ სახელებს, უნდა გადავტვირთოთ მოწყობილობები. ჩართვის შემდეგ, ინფორმაციული უსაფრთხოების გათვალისწინებით აუცილებელია სისტემაში შესვლის მერე გამოვცვალოთ პაროლი. გაუშვათ ტერმინალში „passwd” ბრძანება, რის შემდეგაც სისტემა მოგვთხოვს ახალი პაროლის შეყვანას. პაროლის გამოცვლის შემდეგ აუცილებელია გაუშვათ სისტემის განახლება ტერმინალში შემდეგი ბრძანებების გაშვებით „sudo apt-get update & sudo apt-get upgrade“.

ბრძანებების გაშვების შემდეგ ოპერაციული სისტემა მოძებნის და დააყენებს ყველა საჭირო განახლებას, რაც შეასწორებს და აღმოფხვრის ყველა ცნობილ პროგრამულ ხარვეზს, რაც ჩვენს ოპერაციულ სისტემას შედარებით უფრო მეტად სტაბილურს და საიმედოს გახდის.

მას შემდეგ, რაც პაროლი შევცვალეთ და განახლებები დავაყენეთ, აუცილებელია სისტემაზე გარედან წვდომის მექანიზმის გააქტიურება, ამისთვის ტერმინალში უნდა გავუშვათ შემდეგი ბრძანება „sudo service ssh start“. იმისათვის, რომ აღნიშნული სერვისი სისტემის რესტარტის შემდეგ ავტომატურად გაეშვას, აუცილებელია „/boot“ დირექტორიაში შევქმნათ სერვისის

ავტომატურად გამშვები ფაილი ბრძანებით „sudo touch /boot/ssh“. აღნიშნული სერვისი გვაძლევს ოპერაციული სისტემის ტერმინალთან წვდომას „IP” პროტოკოლის დახმარებით.

მას შემდეგ, რაც სერვისი გააქტიურდება, შევძლებთ ოპერაციულ სისტემაში შესვლას და მის მართვას ტექსტური ბრძანებებით, ნებისმიერი კომპიუტერიდან. ამისთვის საჭიროა ვიცოდეთ: „IP“ მისამართი, მომხმარებლის სახელი და პაროლი. აღნიშნული სერვისის სარგებლობისას შესაძლებელია მხოლოდ ტექსტური ბრძანებების რეჟიმში მუშაობა.

მოწყობილობების მუშა რეჟიმში მოყვანის შემდეგ აუცილებელია ყველა ჰოსტის „/etc/hosts“ ფაილში ჩავამატოთ შემდეგი პარამეტრები:

- 10.112.1.9 Master
- 10.112.1.21 Node-1
- 10.112.1.18 Node-2
- 10.112.1.23 Node-3

ამ პარამეტრების მიხედვით ოპერაციულ სისტემას ეცოდინება თუ რომელი ჰოსტის სახელი რომელ IP მისამართზე მდებარეობს. აღნიშნული მეთოდის ნაკლი ის გახლავთ, რომ რომელიმე IP მისამართის ცვლილების შემთხვევაში ყველა სერვერზე იქნება აღნიშნული კონფიგურაცია ხელით შესაცვლელი.

3.9. Swap_ის პარამეტრების გამართვა ოპერაციულ სისტემაში

შევდივართ სისტემაში და ტერმინალში უშვებთ შემდეგ ბრძანებას „dd if=/dev/zero of=/swap/swapfile bs=1M count=1024 mkswap /swap/swapfile swapon /swap/swapfile“.

ამ ბრძანების გაშვების შემდეგ, ჩვენ მასტერ ნოდზე გამოიყოფა სპეციალური სვაპ სივრცე ერთი გიგაბაიტი მოცულობის. შესამოწმებლად გაუშვათ „top“ ბრძანება, სადაც გამოჩნდება ჩვენი სვაპის ზომა, იმისათვის რომ აღნიშნული

კონფიგურაცია სისტემის გადატვირთვის შემდეგ არ დაგვეკარგოს, აუცილებელია, შევიდეთ „/etc/fstab“ კონფიგურაციის ფაილში „vi“ ტექსტური რედაქტორის დახმარებით და ფაილში ჩავამატოთ შემდეგი ხაზი „/swap/swapfile none swap sw 0 0“.

ამის შემდეგ, შევდივართ „/etc/sysctl.conf“ ფაილში და ვამატებთ შემდეგ პარამეტრს „vm.swappiness = 1“. აღნიშნული პარამეტრი გულისხმობს იმას, რომ სისტემა გადავა სვაპ სივრცის გამოყენებაზე მას შემდეგ, რაც სრულად ამოწურავს ოპერატიული მეხსიერების რესურს.

3.10. Kubernetes ინსტალაცია

იმისათვის, რომს ჩვენმა სისტემამ მუშაობა შეძლოს, აუცილებელია გავმართოთ kubernetes მასტერ ნოდი. ამისათვის საჭიროა შევიდეთ ჩვენს Master ჰოსტზე და გაუშვათ ტერმინალში შემდეგი ბრძანება:

„git clone git@github.com:Project31/kubernetes-installer-rpi.git
cd kubernetes-install-rpi ./build-master.sh“

ამ ჰოსტზე გაშვებული იქნება kubernetes მასტერ ნოდი, რომელზეც გამართული იქნება: „REST API“, „schedule“, პროქსი, რეპლიკაციის მაკონტროლებელი მექანიზმები და ა.შ.

მას შემდეგ, რაც ინსტალაცია დასრულდება, ტერმინალში გავუშვათ ბრძანება „docker ps“, რომლითაც შევამოწმებთ, ვიყენებთ თუ არა დოკერის კონტეინერებს. ყველაფრის გულდასმით შემოწმების შემდეგ, აუცილებელია, გამოვაცხადოთ შემდეგი ცვლადი:

„KUBERNETES_MASTER=http://master:8080“,

რის შემდეგაც, „kubectl“ შეუძლია, ისარგებლოს რესტ აპით მთავარ საკვანძო სერვერზე(მასტერ ნოუდზე). ცვლადის აღწერის შემდეგ შევიდეთ kubectl ტერმინალში და გავუშვათ ბრძანება „get pods“, რომელიც გამოგვიტანს აქტიურ პოდებს, თუ რამდენი დოკერის კონტეინერია გაშვებული ჩვენს მასტერ ნოდზე.

Kubernetes ნოდის ინსტალაცია განსხვავდება მასტერის ინსტალაციისგან. რადგან მისი ფუნქციონალიც განსხვავებულია, მას შეუძლია მხოლოდ პროქსისთან და პოდებთან მუშაობა.

ინსტალაციის დაწყებისთვის შევდივართ NODE ჰოსტზე და ტერმინალში ვუშვებთ შემდეგ ბრძანებას:

„git clone git@github.com:Project31/kuberneres-install-rpi.git cd kubernetes-install-rpi“

ბრძანების შესრულების შემდეგ ჩავასწოროთ კონფიგურაციის ფაილები, სადაც უნდა მივუთითოთ ჩვენი მასტერ ნოდის მისამართი. „kube-procy.yaml“ ფაილში უნდა მივუთითოთ:

„—master=http://master:8080“

ამის მერე შევდივართ „kubelet.service“ ფაილში და ვუთითებთ ჩვენი მასტერ ნოდის რესტ აპის მისამართს „http://10.112.1.9:8080“.

კონფიგურაციის გაწერის შემდეგ, ვუშვებთ ინსტალაციის ბრძანებას „./build-worker.sh“. ინსტალაციის დასრულების შემდეგ, შევამოწმოთ ჩვენი პროქსი ნოდზე. თუ ყველაფერი წესრიგშია, შემდეგი ბრძანებით „docker ps“. მას შემდეგ, რაც დავრწმუნდებით, რომ ყველაფერი გამართულად მუშობს, აუცილებელია, გამოვაცხადოთ შემდეგი ცვლადი:

„KUBERNETES_MASTER=http://master:8080“

რის შემდეგადაც „kubectl“ შეუძლია ისარგებლოს რესტ აპით მასტერ ნოუდზე. აღნიშნული სამუშაოები უნდა ჩავატაროთ ყველა ნოდზე რის შემდეგაც შეგვიძლია შევამოწმოთ ყველა ნოდი დაემატა თუ არა კლასტერში. გაუშვათ შემდეგი ბრძანება „kubectl get nodes“, რამაც უნდა გამოიტანოს სია, სადაც ჩამოთვლილი იქნება ჩვენი სერვერები.

იმისათვის რომ ვისარგებლოთ ჩვენი განაწილებული გარემოთი, აუცილებელია, მასზე გავხსნათ გარე წვდომა, ამისათვის დაგვჭირდება, შევიდეთ მასტერ ნოდზე და შევცვალოთ გარკვეული

პარამეტრები. შევიდეთ „/etc/default/docker “ კონფიგურაციის ფაილში და გავწეროთ შემდეგი:

„DOCKER_OPTS=”–H tcp://10.112.1.9:2375 –H unix:///var/run-/docker.sock –storage-driver=overlay -D” “

ამის შემდეგ, შეგვიძლია დისტანციური წვდომით გავუშვათ წინასწარ გამზადებული აპლიკაციის გამოსახულება დოკერზე.

იმისათვის, რომ შევამოწმოთ, ნამდვილად სწორად მუშაობს თუ არა ჩვენი გარემო, გავუშვათ სატესტო სერვისი, რომელსაც გავუშვებთ ორ ნოდზე. გავუშვათ შემდეგი ბრძანება მასტერ ნოდზე:

„kubectl -s http://localhost:8080 run httpd --image=hypriot/rpi-busybox-httpd --port=80 kubectl scale --replicas=2 rc httpd kubectl get pods -o wide“.

დავინახავთ, რომ ბრძანება გაეშვება მასტერ ნოდზე, ხოლო სერვისი პროქსი - ნოდების Pod-ებზე. ყველაფერი მუშაობს იმისდამიუხედავად, რომ მასტერ ნოდს არ აქვს საკმარისი მეხსიერება, იმისთვის, რომ გაუშვას სერვისები. სამაგიეროდ პროქსი ნოდებზე არის საკმარისი მეხსიერება და მეტი სიმძლავრის საჭიროების შემთხვევაში მარტივად, პროქსი ნოდის დამატებით შესაძლებელი იქნება გამოთვლითი რესურსის დამატება.

აღნიშნულ არქიტექტურას გამოვიყენებთ ჩვენს მიერ შემუშავებული მონიტორინგის და სტატისტიკის პროგრამას, რომელიც დაწერილი გვაქვს Python პროგრამირების ენაზე ვიყენებთ Pandas, numpy და ნეირონული ქსელების ელემენტებს [58,59].

IV თავი. ექსპერიმენტული ნაწილი: სისტემის დემო ვერსია

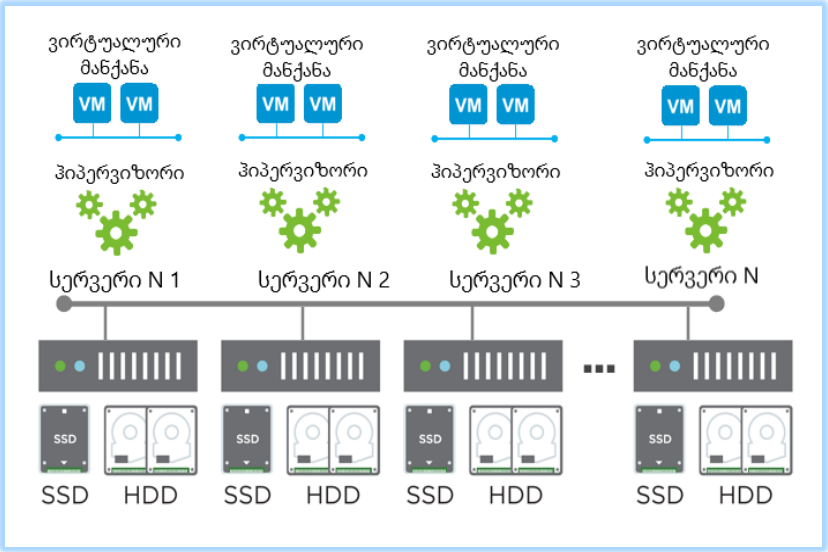
4.1. Linux-ის ოპერაციული სისტემის გამართვა

Linux-ის ბირთვი წარმოადგენს 22 მილიონი სტრიქონის მოცულობის კოდის მქონე ოპერაციულ სისტემას, რაც ძალიან მცირე რესურსს მოითხოვს: პროცესორი 400 Mhz, მეხსიერების ზომა მყარ დისკზე 10 MB, ოპერატიული მეხსიერება 256 MB, რის გამოც ოპერაციული სისტემა ბევრად უფრო მოსახერხებელი ხდება.

მომხმარებელს ოპერაციული სისტემის დაყენებისას აქვს მინიმალური პაკეტის დაყენების არჩევანი, რის შემდეგაც დაყენდება მხოლოდ ის სისტემები რომლებიც მას სჭირდება. შესაბამისად, ოპერაციული სისტემის ზომა მინიმალური გამოდის, რაც ამცირებს რისკებს და ზრდის სისტემის საიმედოობას. ლინუქსს, ამ მახასიათებლების გამო ხშირად იყენებენ სხვადასხვა მიზნებისთვის: web server, http/https server, proxy server, ftp server, firewall gateway, distributoin systems, database server და ა.შ. [20,55].

ლინუქსის ოპერაციული სისტემა ფუნქციონირებისთვის საჭირო პარამეტრებს ინახავს ტექსტურ ფაილებში. კონფიგურაციის შესაცვლელად საკმარისია შესაბამისი ტექსტური ფაილის რედაქ-ტირება. სწორედ ამ მახასიათებლების გამო შევაჩერეთ არჩევანი ლინუქსის ტიპის ოპერაციულ სისტემებზე, კერძოდ კი “CentOS 7”.

აღნიშნული ოპერაციული სისტემა არის ერთ-ერთი ყველაზე ცნობილი ღია სისტემა, რომელიც დაწერილია „Red Hat Enterprise Linux(RHEL)“ საწყისი კოდის ბაზაზე. Red Hat არის პროგრამულ უზრუნველყოფაზე ორიენტირებული კომპანია [20]. იმისათვის რომ დავიწყოთ Centos 7-ის ინსტალაცია და გამართვა, პირველ რიგში, უნდა განვსაზღვროთ თუ რა პლათმორმაზე იმუშავებს სისტემა. თანამედროვე ტექნოლოგიების გათვალისწინებით, ჩვენს წიგნში აღწერილი ეკოსისტემის გამართვისთვის გამოყენებულია შემდეგი არქიტექტურა (ნახ.4.1):



ნახ.4.1. დატაცენტრის ლოგიკური ინფრასტრუქტურა

ფიზიკური სერვერები (სერვერ N1, სერვერ N2, სერვერ N3) უკავშირდება ერთმანეთს 10 გიგაბაიტიანი ქსელის საშუალებით, სერვერებზე მიერთებულია „DAS“ სთორიჯი, რაც უზრუნველყოვს სერვერზე არსებული ინფორმაციის საიმედოდ შენახვას.

მყარი დისკის დაზიანების შემთხვევაში მონაცემები არ დაიკარგება და „hot-plug“ ტექნოლოგიის დახმარებით სერვერი უწყვეტ რეჟიმში გააგრძელებს მუშაობას [55].

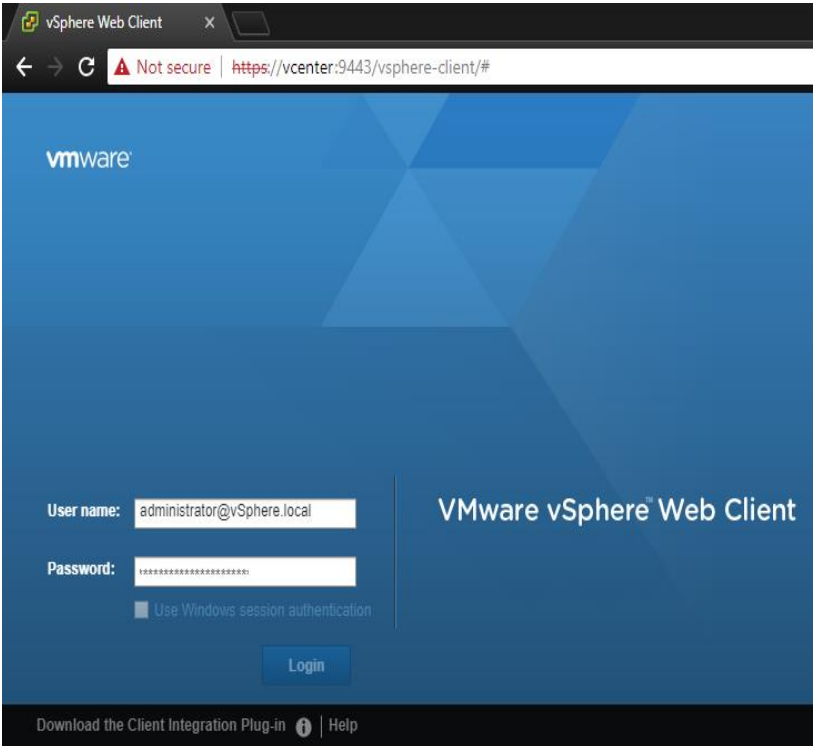
რესურსების ეფექტურად გამოყენების მიზნით აღნიშნულ სერვერებზე გამართულია ჰიპერვიზორი, რომელიც უზრუნველყობს ვირტუალური მანქანების მართვას და გვაძლევს ფიზიკური მანქანის რესურსის სრულად გამოყენების საშუალებას.

4.1.1. ვირტუალური მანქანის შექმნა

Linix ოპერაციული სისტემის გამართვის დაწყებამდე, აუცილებელია ვირტუალური მანქანის შექმნა. განვიხილოთ „VMware“ ვირტუალიზაციის პლატფორმის მაგალითი. საჭიროა შევიდეთ „vCenter“-ში, საიდანაც იმართება მთლიანი ვირტუალური პლატფორმა. გავხსნათ ვებ-ბრაუზერი (რეკომენდებულია „Google Chrome“) და დავწეროთ შემდეგი მისამართი:

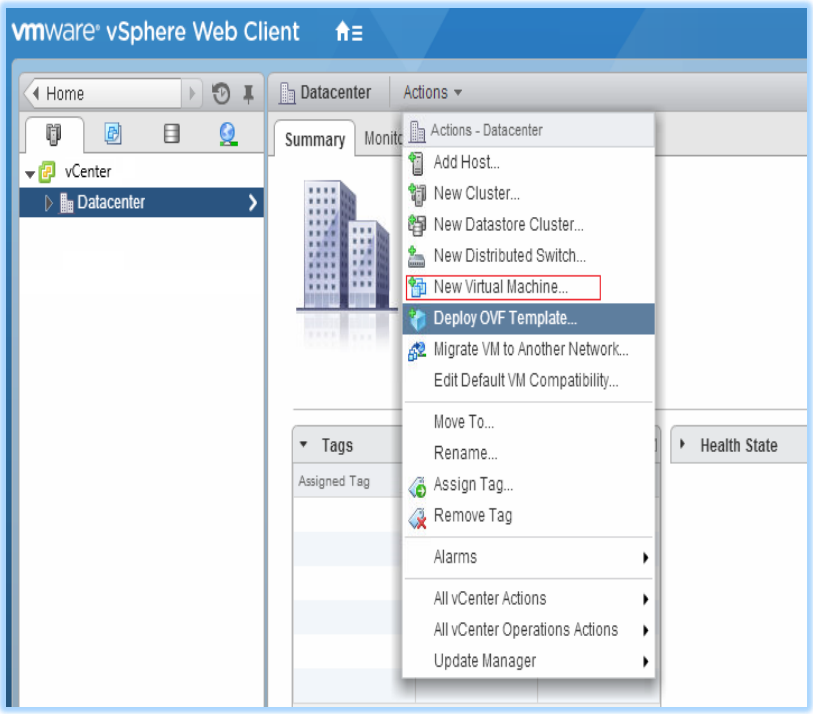
https://vcenter:9443/vsphre-client/#

ვებ-მისამართზე შესვლის შემდეგ საჭიროა გავიაროთ ავტორიზაცია (ნახ.4.2).



ნახ.4.2..ვირტუალური პლატფორმის მართვის ცენტრი

ავტორიზაციის შემდეგ სანავიგაციო ველში გადავდივართ სერვერებისა და კლასტერების განყოფილებაში, სადაც ვირჩევთ დატაცენტრს „Datacenter“ და „Actions“. ღილაკის დახმარებით გამოგვაქვს მოქმედებების მენიუ. აღნიშნულ მენიუში ვირჩევთ „New Virtual Machine“ მოქმედებას და ვქმნით ვირტუალურ მანქანას (ნახ.4.3).



ნახ.4.3. ვირტუალური მანქანის შექმნა

ვირტუალური მანქანის შექმნისას უნდა გავითვალისწინოთ, თუ რომელი ოპერაციული სისტემისთვის ვქმნით მას და რა სერვისები იქნება შემდგომში მისთვის გაშვებული. მაგალითად, თუ იქნება უბრალო პროქსი სერვერი, რომლის ფუნქციაც მხოლოდ მონაცემების ქსელური ნაკადის გატარებაა, მისთვის მნიშვნელოვანი იქნება ქსელის წარმადობა და პროცესორის სიმძლავრე.

თუ ვირტუალრ მანქანაზე დაგეგმილია მონაცემთა ბაზის გამართვა ქსელთან და პროცესორთან ერთად, მაშინ მნიშვნელოვანია სწორად დაიგეგმოს მყარი დისკოების რაოდენობა და სისწრაფე.

სხვადასხვა არქიტექტურის და ლიტერატურის კრიტიკული ანალიზის საფუძველზე მივედით იმ დასკვნამდე, რომ თუ ვირტუალური მანქანა იქმნება მონაცემების შენახვა-დამუშავების მიზნით, მას უნდა ჰქონდეს მინიმუმ სამი ვირტუალური დისკო „vmdk“. აღნიშნულ დისკოების განაწილება იქნება შემდეგნაირად: ერთ დისკოზე განთავსდება „SWAP“ არეა, რომელიც სასურველია იყოს „NVMe“ ტიპის სწრაფი დისკოების „RAID“ მასივზე; მეორე დისკოზე განთავსდება ოპერაციული სისტემა, რომელიც შესაძლებელია იყოს „RAID_5“ ტიპის დისკოების მასივზე. ხოლო მესამე დისკო გათვლილია მონაცემების ჩაწერა-წაკითხვაზე. აღნიშნული დისკოს წარმადობა უნდა გამოითვალოს მოსალოდნელი დატვირთვის მიხედვით, რეკომენდებულია მინიმუმ „RAID_10“ ტიპის მასივი [26, 28,31].

4.1.2. CentSO 7 ის გამართვა

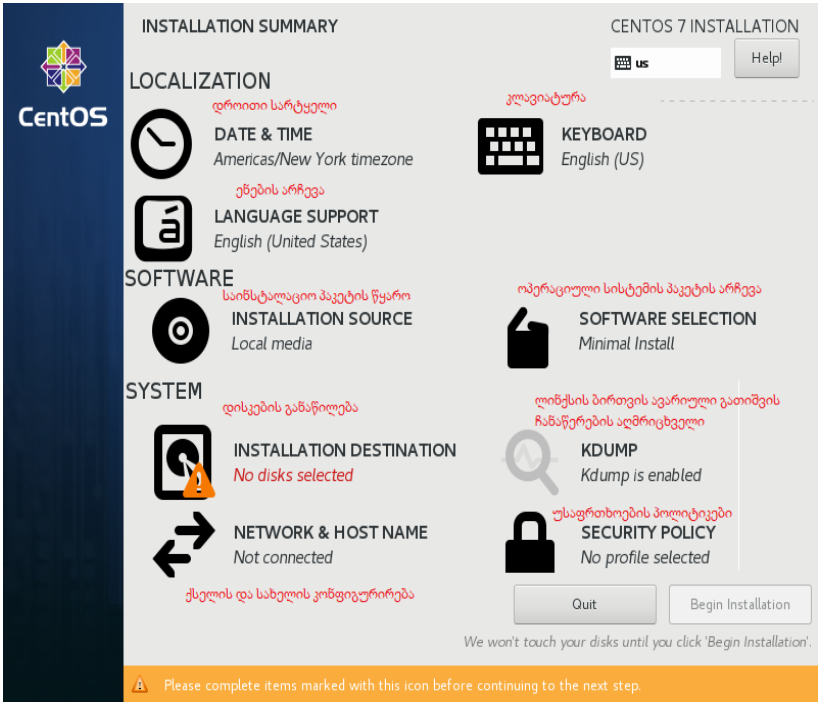
ვირტუალური მანქანის შექმნის შემდეგ შეგვიძლია ლინუქს ოპერაციული სისტემის გამართვის დაწყება. ამისათვის აუცილებელია შევიდეთ ვებ-გვერდზე:

https://www.centos.org/download/

სადაც ვირჩევთ მინიმალური პაკეტის გადმოწერას „Minimal ISO“.

ყურადღება გავამახვილოთ, რომ ამ დროს გადმოვიწეროთ 64 ბიტიანი არქიტექტურის საინსტალაციო პაკეტი („CentOS-7-x86_64-Minimal-1804.iso“). საინსტალაციო პაკეტის გადმოწერის შემდეგ ვრთავთ ვირტუალურ მანქანას და ვიწყებთ ოპერაციული სისტემის გამართვას. პირველ რიგში ვირჩევთ ჩვენთვის სასურველ კლავიატურის წყობას („English (US)“) , ვირჩევთ დროის სარტყელს. რადგანაც ჩვენ თავიდანვე გადმოვიწერეთ მინიმალური

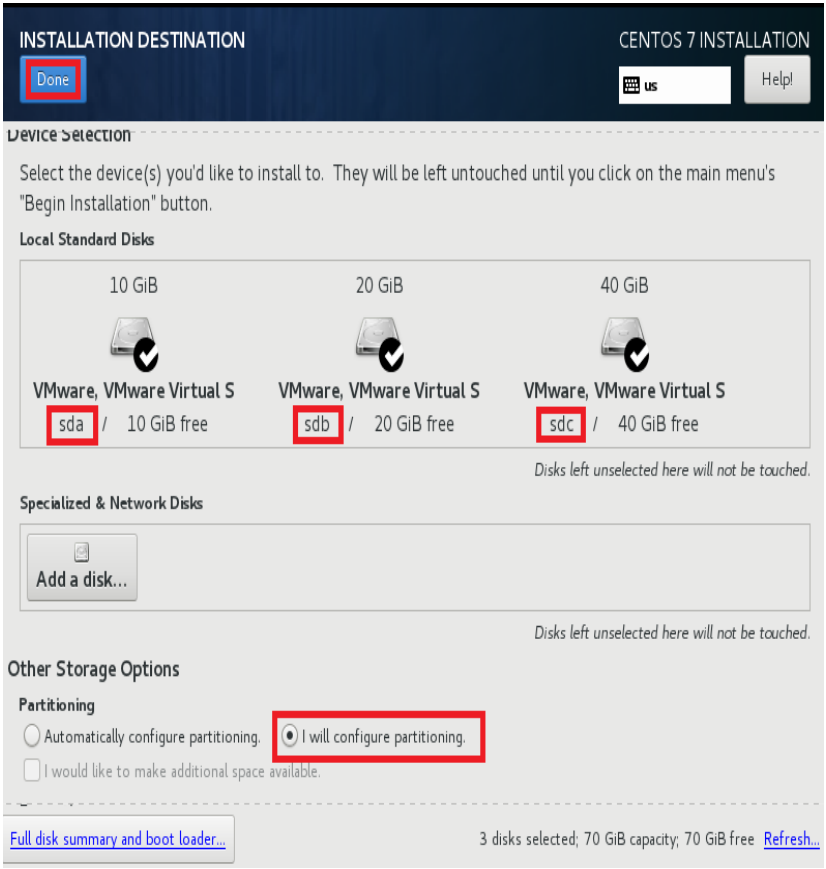
საინსტალაციო პაკეტი „SOFTWARE SELECTION“ ველში ავტომატურად არის მონიშნული მინიმალური პაკეტის ინსტალაცია. სხვა ვერსიის გადმოწერის შემთხვევაში მოგვიწევდა საინსტალაციო ვერსიის ხელით მითითება (ნახ.4.4).



ნახ.4.4. CentOS პარამეტრები

SYSTEM ჩანართში უნდა მივუთითოთ კომპიუტერის ქსელის პარამეტრები, სერვერის სახელი, Linux-ის ბირთვის ხარვეზების ლოგირების ფუნქციონალი, უსაფრთხოების პარამეტრები და დისკოების განაწილების მეთოდოლოგია. აღნიშნული პარამეტრებიდან ყველაზე მნიშვნელოვანია დისკოების სწორად განაწილება, რადგან ოპერაციული სისტემის ინსტალაციის შემდეგ ყველა პარამერტრის შეცვლა არის შესაძლებელი გარდა დისკოების განაწილებისა. ამიტომაც აუცილებელია თავიდანვე სწორად გავანაწილოთ.

ამისათვის შევდივართ დისკოების მართვის ცენრტში „INSTALLATION DESTINATION“, სადაც ვირჩევთ ყველა იმ დისკოს, რომელზეც ვაპირებთ ოპერაციული სისტემის გამართვას. ჩვენს შემთხვევაში ვირჩევთ „sda , sdb, sdc“ დისკოებს და ვნიშნავთ პარამეტრს „I will configure partitioning“, რომელიც საშუალებას გვაძლევს ხელით ავირჩიოთ თუ რომელ დისკოზე რა ინფორმაცია ჩაიწერება და ვაწვებით ღილაკს „Done“ (ნახ.4.5).



ნახ.11.5. CentOS დისკოების არჩევა

სანამ მომდევნო ეტაპზე გადავალთ, მოკლედ გავიხსენოთ თუ როგორ არის მოწყობილი Linux-ის ფაილური სისტემა.

ოპერაციული სისტემის დირექტორიები ფორმირებულია ხის სტრუქტურით. მთავარია „ / “ დირექტორია, რომელიც შეიცავს შემდეგ დირექტორიებს (აღნიშნულ ეტაპზე გამოვყოფთ მხოლოდ იმ დირექტორიებს, რომლებიც ჩვენი კვლევის თანახმად არის კრიტიკულად მნიშვნელოვანი) იხილეთ ცხრილი 4.1.

ოპერაციული სისტემის დირექტორიები ცხრ.4.1

დირექტორიები	აღწერა
„/“	მთავარი დირექტორია
„/boot“	ინახება ოპერაციული სისტემის ჩატვირთვისთვის აუცილებელი ინფორმაცია
„/home“	ინახება მომხმარებლის ინფორმაცია
„/var“	ინახება ლოგები ჩანაწერები
„/data“	აღნიშნულ დირექტორიას ვიყენებთ მანაცემების შესანახად
„/swap“	სვაპი სპეც-ფორმატის დირეტორიაა, რომელშიც ჩაწერა ხდება მხოლოდ იმ შემთხვევაში, თუ ოპერატიული მეხსიერება არის ბოლომდე შევსებული

რადგან დისკოების განაწილების საკითხი ძალზე მნიშვნელოვანია და ყოველთვის შეიძლება დადგეს იმის საჭიროება, რომ გავზარდოთ ჩვენ მიერ გამოყოფილი ადგილი ამა თუ იმ დირექტორიისთვის, შევიმუშავეთ „LVM“ ტექნოლოგია, რომლის მუშაობის პრინციპიც არის შემდეგნაირი:

ვქმნით ვირტუალურ დისკოების ჯგუფს „virtual volume groups“, რომელშიც შეგვიძლია დავამატოთ ჩვენი ფიზიკური დისკოები, რაც საშუალებას გვაძლევს სურვილის შემთხვევაში დავამატოთ ფიზიკური დისკოები და გავზარდოთ ჩვენი

ვირტუალური ჯგუფის ზომა. მას შემდეგ რაც შევქმნით ვირტუალურ ჯგუფს და მივაბავთ მას ფიზიკურ დისკოს, ჩვენ შგვიძლია შევქმნათ „LVM“ ვირტუალური დისკოები, რომლებიც გაიყოფს ვირტუალური ჯგუფის ზომას და წარმადობას. აღნიშნულ ლოგიკურ დისკოზე უკვე შეგვიძლია მივაბათ დირექტორიები.

ნაშრომის კვლევის ფარგლებში მოვიძიეთ და შევიმუშვეთ ფაილების შემდეგი სახის განლაგება (ცხრილი 4.2).

დისკების განაწილება **ცხრ.4.2**

ვირტუა-ლური ჯგუფის სახელი	ლოგიკური დისკის სახელი	დირექ-ტორიის სახელი	ზომა	ფორ-მატი	ფიზიკური დისკი
SwapVG	swapLV	swap	10 GiB	swap	sda (10 GiB)
DataVG	dataLV	/data	40 GiB	XFS	sdc (40 GiB)
RootVG	logLV	/log	8 GiB	XFS	sdb2 (19 GiB)
RtootVG	homeLV	/home	2 Gib	XFS	sdb2 (19 GiB)
RootVG	rootLV	/	9 GiB	XFS	sdb2 (19.5 GiB)
		/boot	1 GiB	esct4	sdb1 (1 GiB)

სერვერის პროექტირებისას შევეცადეთ, რომ სხვადასხვა მოთხოვნების შემთხვევაში მინიმალური ცვლილებებით შეგვძლებოდა სასურველი მიზნის მიღწევა.

შევქმენით *ვირტუალური მანქანის შაბლონი*, რომელსაც მოერგებოდა ჩვენი მიზნები ძირეული ცვლილებების გარეშე. შევქმენით ვირტუალური მანქანა სამი „vmdk“ ფაილით, რომლებიც Linux-ის სისტემისთვის არის ფიზიკური მყარი დისკოები. დავიწყეთ ოპერაციული სისტემის დაყენება ვირტუალურ მანქანაზე და დისკოების დაგეგმვის ნაწილში შევქმენით სამი ძირითადი ვირტუალური დისკოების ჯგუფი:

- SwapVG – ვირტუალური ჯგუფი, რომელზეც არის მიბმული „sda“ ფიზიკური დისკო. მისი ზომა არის 10 Gb. შევქმენით ლოგიკური დისკო „swapLV“ და მასზე მივაერთეთ swap დირექტორია,

რომელიც არის დაფორმატებული swap ფორმატში. ოპერაციული სისტემა იყენებს მას ოპერატიული მეხსიერების მაგივრად, როდესაც ოპერატიული მეხსიერება გადავსებულია;

- DataVG – ვირტუალური ჯგუფი, რომელშიც გაწევრიანებულია „sdc“ ფიზიკური დისკო. მისი ზომა არის 40 Gb. შევქმენით ლოგიკური დისკო „dataLV“, რომელსაც მივაბით „/data“ დირექტორია. აღნიშნული დირექტორი გვჭირდება ისეთ შემთხვევებში, როდესაც სერვერის დანიშნულებაა მონაცემების შენახვა-დამუშავება, მაგალითად, მონაცემთა ბაზაა, ფაილსერვერი ან სხვა მსგავსი დანიშნულების. ეს არქიტექტურა გვაძლევს საშუალებას, საჭიროებისამებრ, დავამატოთ მონაცემების ჩაწერა-წაკითხვის სიჩქარე და საიმედოობაც;
- RootVG – ვირტუალური დისკოების ჯგუფი. ეს ჯგუფი გათვლილია ოპერაციული სისტემისთვის. მასში არის გაწევრიანებული „sdb2“ ფიზიკური დისკო. მისი ზომაა 19 Gb. ამ ჯგუფში შევქმენით შემდეგი ლოგიკური დისკოები: „logLV“, რომელზეც მიმაგრებული იქნება „/log“ დირექტორია. მასში ჩაიწერება როგორც ოპერაციული სისტემის, ასევე მასზე გაშვებული სერვისების ლოგ ჩანაწერები. აღნიშნული დირექტორია აუცილებელია იყოს გამოყოფილი სხვა დირექტორიებიდან, რადგანაც ის მუდმივად იზრდება. იმ შემთხვევაში თუ არ გამოვყოფთ, მას ცალკე შეუძლია გადაავსოს მთლიანი მყარი დისკო, რაც ოპერაციული სისტემის დაზიანებას გამოიწვევს და ჩვენი სერვისები მიუწვდომელი გახდეს.

„homeLV“ ლოგიკური დისკო. მასზე მიერთდება „/home“ დირექტორია, სადაც განთავსებულია ოპერაციული სოსტემის მომხმარებელთა ფაილები. აღნიშნული დირექტორიის ზომა უნდა განისაზღვროს სერვერის დანიშნულების მიხედვით. არის ისეთი შემთხვევები, როდესაც „/home“ დირექტორიაში ინახება მომხმარებლის გარდა სხვა მონაცემებიც, მაგალითად, db2 მანაცემთა ბაზის ინსტანტის მონაცემები ავტომატურად ინახება „home“ დირექტორიის ქვესაქაღალდეებში. „rootLV“ ლოგიკური

დისკო, რომელზეც მიერთებული იქნება „/“ დირექტორია. ეს იმას ნიშნავს, რომ ყველა სხვა დირექტორიები, რომლებიც არის ოპერაციულ სისტემაზე მიერთებული, იქნება აღნიშნულ ლოგიკურ დისკოზე და გაიყოფენ ამ დისკოს წარმადობას.

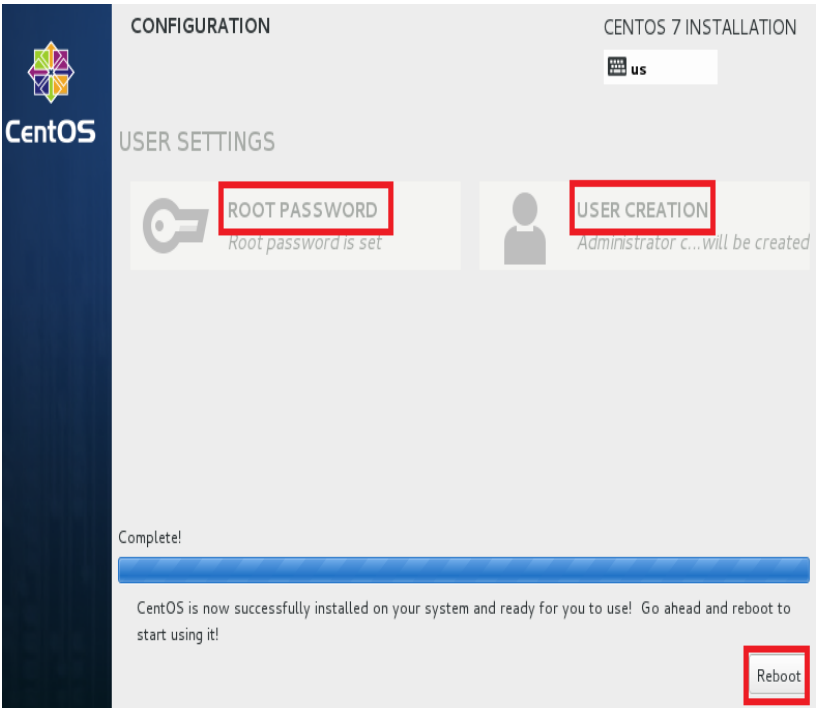
4.2 ცხრილის მიხედვით დისკოს ვირტუალურ დისკოებთან ერთად არის მოხსენიებული „/boot“ დირექტორია, რომელიც არ არის მიბმული არცერთ ლოგიკურ დისკოზე. „/boot“ არის დირექტორია, სადაც ჩაწერილია ინფორმაცია, რომლითაც ოპერაციული სისტემა იტვირთება.

აღნიშნული დირექტორია უნდა იყოს ჩაწერილი აუცილებლად ფიზიკურ დისკოზე. ამ შეზღუდვის გამო „sdb“ ფიზიკური დისკო, რომელიც 20 Gb იყო, დავყავით ორ ნაწილად „sdb1“ 1 Gb ფიზიკური დისკო, რომელსაც მივაბით „/boot“ დირექტორია და „sdb2“ 19 Gb ფიზიკური დისკო, რომელიც მიერთებულია „RootVG“ ვირტუალური დისკოების ჯგუფზე.

მას შემდეგ, რაც დისკოების განაწილება დამთავრდა, გავაგრძელეთ ოპერაციული სისტემის ინსტალაცია. ინსტალაციის შემდეგ ეტაპი არის „USER SETTINGS“ მომხმარებლის კონფიგურირება, სადაც უნდა განვსაზღვროთ ოპერაციული სისტემის „root“ მომხმარებლის პაროლი, ასევე შეგვიძლია ინსტალაციის პროცესშივე დავამატოთ ჩვენთვის სასურველი სხვა მომხმარებლებიც (ნახ.4.6).

„root“ მომხმარებელი არის Linux ოპერაციულ სისტემაში პრივილეგირებული მომხმარებელი, რომელსაც ყველაფრის უფლება აქვს.

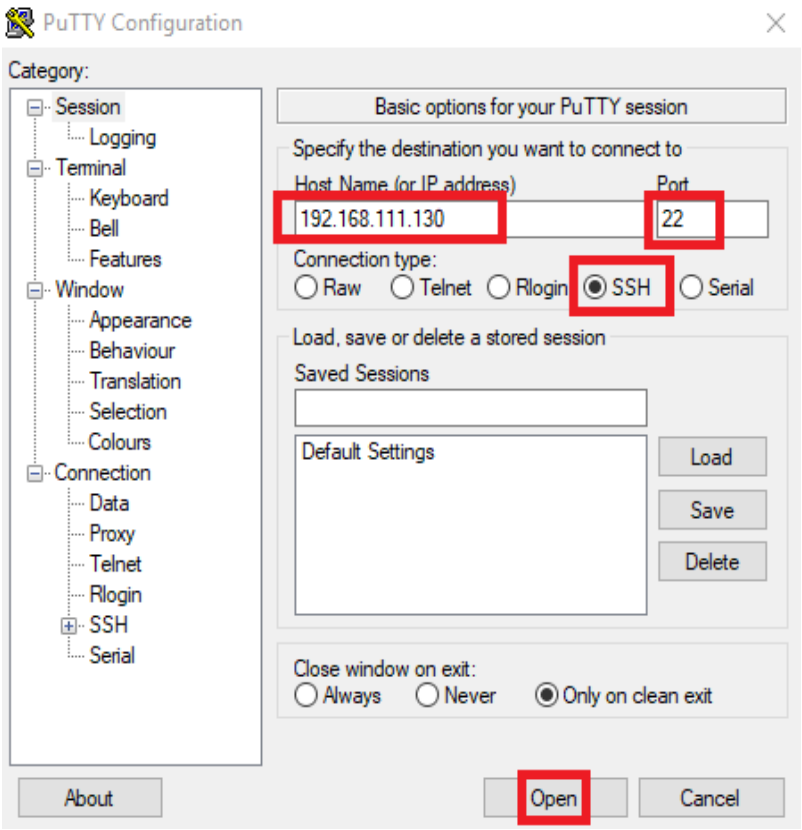
მას შემდეგ რაც დავაკონფიგებთ მომხმარებელს და ოპერაციული სისტემაც მორჩება ინსტალაციას „Reboot“ ღილაკით გადავტვირთავთ და შემდეგ ჩართვაზე ჩაიტვირთება ჩვენს მიერ დაყენებული სისტემა.



ნახ.4.6..ოპერაციული სისტემის მომხმარებლის კონფიგურირება

4.1.3. საჭირო პაკეტების ჩაწერა და ოპერაციული სისტემის განახლება

აღნიშნული არქიტექტურის გათვალისწინებით ჩვენს მიერ გამართულ სერვერზე შესასვლელად შეგვიძლია გამოვიყენოთ ორი ხერხი: პირველი, ინსტალაციის დროს ქსელის პარამეტრების გასწორებისას ამოწერილი IP მისამართით „ssh“ პროტოკოლის გამოყენებით ან *ვირტუალური მანქანის* კონსოლის გახსნით არის შესაძლებელი. სერვერზე შესასვლელად გამოვიყენოთ „Putty“ (პროგრამა, რომელიც გვაძლევს საშუალებას „ssh“ პროტოკოლის დახმარებით დავუკავშირდეთ სერვერს) (ნახ.4.7).



ნახ.4.7. Putty ქსელური წვდომის აპლიკაცია

აპლიკაცია შეიძლება გადმოიწეროს www.putty.org მისამართზე. სერვერზე შესასვლელად ვხსნით აპლიკაციას „Connection type“, ველში ვნიშნავთ „SSH“, ხოლო „Host Name“ ველში ვუთითებთ სერვერის „IP“ მისამართს. გადავამოწმოთ, რომ სერვერს ნამდვილად 22 პორტით ვუკავშირდებით და დავაჭიროთ „Open“ ღილაკს. შემდეგ გაიხსნება სესია სერვერთან, სადაც უნდა გავიაროთ ავტორიზაცია, შეგვყავს ოპერაციული სისტემის მომხმარებილის სახელი და პაროლი.

ავტორიზაციის გავლის შემდეგ ჩაიტვირთება „Bash shell“, რითიც ხდება ოპერაციული სისტემის მართვა. ბაშის დახმარებით ჩვენ შეგვიძლია ოპერაციულ სისტემას გადავცეთ სხვადასხვა დავალებები. ახლად გამართულ სისტემაში, პირველ რიგში, უნდა შევამოწმოთ იმ ბრძანებების (პროგრამული პაკეტების) ნუსხა, რომელიც მოყვება Linux-ის ვერსიას და შემდეგ ჩვენთვის საჭირო პაკეტები დავაყენოთ. შევამოწმოთ არის თუ არა ყველა ის საჭირო პაკეტი დაყენებული ოპერაციულ სისტემაზე (ცხრილი 4.4).

ლინუქს ოპერაციული სისტემის პრძანებები **ცხრ.4.2**

ბრძანება	ხელმისა-წვდომია	აღწერა
w	კი	გვიჩვენებს სერვერზე მყოფი მომხმარებლების სიას.
vmstat	კი	მონიტორინგის ხელსაწყო რომელიც გვიჩვენების ოპერატიული მეხსიერებისა და პროცესორის დატვირთულობას.
df	კი	გვიჩვენებს ლინუქსის ფაილურის სისტემის ჯამურ ზომებს.
ps	კი	აღნიშნულ ბრძანება გამოაქვს მიმდინარე პროცესები. -A პარამეტრის დამატების შემთხვევაში აჩვენებს ოპერაციულ სისტემაში გაშვებულ ყველა პროცესს.
top	კი	აპლიკაციაა რომელიც გვიჩვენებს პროცესორისა და ოპერატიული მეხსიერების დატვირთულობას მიმდინარე რეჟიმში.
free	კი	გვიჩვენებს თავისუფალ და შევსებულ ოპერატიულ მეხსიერებას მათ შორის swpa სივრცესაც.
uptime	კი	გვიჩვენებს თუ რამდენი დროა გასული სერვერის ჩართვიდან, რამდენი მომხმარებელი არიჩ შესული სისტემაში და ოპერაციული სისტემის საშუალო დატვირთულობას.
ip add	კი	გვიჩვენებს ჩვენს ip მისამართს

who am i	კი	გამოაქვს მიმდინარე სესიის მონაცემები მომხმარებლის სახელი როდის დაუკავშირდა სერვერს და ip მისამართი.
ifstat	კი	გვიჩვენებს ქსელის ადაპტერების აქტივებებს.
ifconfig	არა	პროგრამული პაკეტი რომელიც გვეხმარება ქსელის პარამეტრებთან მუშაობასი გვიჩვენებს ip მისამართს და პაკეტების სტატისტიკას.
tcpdump	არა	ეს არის პროგრამული პაკეტი რომელიც საშუალებას გვაძლევს ქსეის ნაკადის ჩაწერას. “tcpdump -w - -p -n -s 0 -i ens33 > /tmp/test.pcap” აღნიშნული ბრძანებით ჩვენი ქსელის ნაკადს ვიწერთ „temp“ დირექტორიაში „pcap“ ფორმატით. ამ ფაილის გაანალიზება შესაძლებელია ერთ-ერთი მძლავრი პაკეტების ანალიზარორი პროგრამის „wireshark“ დახმარებით.
lsof	არა	გამოაქ გახსნილი ფაილების სისა.
iotop	არა	გვიჩვენებს დისკებზე ჩაწერა წაკითხვის დატვირთვას.
wget	არა	პროგრამული პაკეტი რომელიც გვაძლევს საშუალებას გადმოვიწეროთ ფაილები „HTTP“, „HTTPS“ და „FTP“ პროტოკოლების გამოყენებით.
traceroute	არა	კომპიუტერული ქსელის დიაგნოსტიკის ხელსაწყოა, რომელიც გვიჩვენებს თუ რა გზას გადის პაკედი დანიშნულების ადგილამდე.
zip	არა	ფაილების დასაარქივებელი ხელსაწყო
unzip	არა	დაარქივებული ფაილებისს გასახსნელი ხელსაწყო
htop	არა	გვიჩვენებს სისტემაში მიმდინარე პროცესებს და სერვერის დატვირთულობას .
nmon	არა	მონიტორინგის ხელსაწყო რომელიც გვიჩვენებს სერვერის დატვირთულობას სხვადასხვა ჭრილში.

იმისათვის, რომ დავაყენოთ ჩვენთვის საჭირო აპლიკაციების პაკეტები, უნდა გავუშვათ შესაბამისი ბრძანებები (ცხრილი 4.5).

საკვანძო პაკეტების ინსტალაცია ცხრ.4.3

ბრძანება
yum -y install net-tools
yum -y install tcpdump
yum -y install lsof
yum -y install wget
yum -y install iotop
yum -y install traceroute
yum -y install bash-completion
yum -y install yum-utils
yum -y install zip
yum -y install unzip
yum -y install epel-release
yum -y install htop
yum -y install nmon
yum -y install sysstat
yum -y install bind-utils

წიგნის ფარგლებში მოვიძიეთ ყველა საჭირო პროგრამული პაკეტი. მათი ინსტალაცია ხდება „yum“ ბრძანებით. ვწერთ „yum“ ბრძანებას და შემდეგ იმ პროგრამული პაკეტის დასახელებას, რომლის დაყენებაც გვსურს.

ბრძანების აკრეფის შემდეგ „yum“ ხელსაწყო ნახულობს ოპერაციულ სისტემაში არსებული რეპოზიტორების ჩამონათვალს და ამ რეპოზიტორებში ეძებს ჩვენს მიერ მოთხოვნილ აპლიკაციას. თუ იპოვის, გადმოწერს აპლიკაციას და შეუდგება ინსტალაციას.

ინსტალაცია შედგება რამდენიმე ეტაპისგან: ფაილის გადმოწერა, შემოწმება - აკმაყოფილებს თუ არა ოპერაციული სისტემა პროგრამის მოთხოვნებს და, საჭიროებისამებრ, იმ აპლიკაციების დაყენებასაც, რაც ჩვენ მოთხოვნილ პაკეტს სჭირდება ფუნქციონირებისთვის. ყოველივე ამის შემოწმების შემდეგ ხდება ჩვენს მიერ მოთხოვნილი აპლიკაციის ინსტალაცია. შესაბამისად

უნდა ვიზრუნოთ იმაზე, რომ ჩვენი რეპოზიტორების ჩამონათვალში იყოს ყველა საჭირო რეპოზიტორი. წინააღმდეგ შემთხვევაში ჩვენს მიერ მოთხოვნილი აპლიკაცია არ დაყენდება.

იმისათვის, რომ დავაყენოთ აპლიკაციები, ჩვენ შეგვიძლია გავუშვათ სათითაო ბრძანება, რომლებიც მოყვანილია ცხრილში ან მოვახდინოთ ამ ბრძანებების ერთ ბრძანებად გადაკეთება და ერთი ბრძანების გაშვებით დავაყენოთ ჩვენთვის სასურველი ყველა აპლიკაცია. მაგალითად:

„yum -y install net-tools tcpdump lsof wget iotop traceroute bash-completion yum-utils zip unzip epel-release htop nmon sysstat bind-utils“

ამ ბრძანების გაშვების შემდეგ ყველა აპლიკაცია ავტომატურად დაყენდება. იმისათვის, რომ გადავამოწმოთ ნამდვილად ჩაიწერა თუ არა ყველა აპლიკაცია, გავუშვათ შემდეგი ბრძანება „yum list installed“, რომელიც გამოიტანს სიას, სადაც იქნება ყველა იმ პროგრამული პაკეტის დასახელება რომელიც არის ოპერაციულ სისტემაში ჩაწერილი (ნახ.4.8).

მას შემდეგ, რაც დავამთავრებთ აპლიკაციების ჩაწერას, აუცილებელია ოპერაციულ სისტემას გავუკეთოთ განახლება. ამისათვის ვუშვებთ შემდეგ ბრძანებებს:

„yum clean all“ ასუფთავებს დაქეშილ ფაილებს.

„yum -y update && yum -y groupinstall core && yum -y groupinstall base“.

აღნიშნული ბრძანებით მოვახდინეთ ოპერაციული სისტემის განახლება. სისტემის განახლების დასრულების შემდეგ გავუშვით „package-cleanup --oldkernels --count=1“ ბრძანება, რომლითაც ჩატვირთვის სიიდან წავშალეთ Linux-ის ბირთვის ძველი ვერსიები.

ოპერაციულმა სისტემამ სრულად და მაქსიმალური წარმადობით მუშაობა რომ შეძლოს, ამისათვის დავაყენეთ „yum install open-vm-tools“ პროგრამული პროდუქტი, რომელიც უზრუნველყოფს სისტემის გამართულ ფუნქციონირებას *ვირტუალური პლატფორმის* შემთხვევაში.

```
[root@CentOS ~]# yum list installed
Loaded plugins: fastestmirror, langpacks
Repodata is over 2 weeks old. Install yum-cron? Or run: yum makecache fast
Determining fastest mirrors
 * base: mirror.centos.ge
 * epel: epel.grena.ge
 * extras: mirror.centos.ge
 * updates: mirror.centos.ge
Installed Packages
GeoIP.x86_64                                    1.5.0-11.el7
NetworkManager.x86_64                           1:1.10.2-16.el7_5
NetworkManager-libnm.x86_64                     1:1.10.2-16.el7_5
NetworkManager-team.x86_64                      1:1.10.2-16.el7_5
NetworkManager-tui.x86_64                       1:1.10.2-16.el7_5
NetworkManager-wifi.x86_64                      1:1.10.2-16.el7_5
htop.x86_64                                     2.2.0-1.el7
iotop.noarch                                    0.6-2.el7
iproute.x86_64                                  4.11.0-14.el7
nano.x86_64                                     2.3.1-10.el7
traceroute.x86_64                               3:2.0.22-2.el7
unzip.x86_64                                    6.0-19.el7
wget.x86_64                                     1.14-15.el7_4.1
which.x86_64                                    2.20-7.el7
yum-utils.noarch                                1.1.31-45.el7
zip.x86_64                                      3.0-11.el7
zlib.x86_64                                     1.2.7-17.el7
```

ნახ.4.8. Yum_ით დაყენებული აპლიკაციების ნუსხა

ზემოთ მოყვანილი სამუშაოების ჩატარების შემდეგ გამზადებული გვაქვს *ვირტუალური მანქანა*, რომელზეც გამართულია Linux-ის ოპერაციული სისტება და დაყენებულია ყველა ის მინიმალური პროგრამული პაკეტი, რომელიც აუცილებელია ჩვენი წიგნის ფარგლებში. შესაბამისად შეგვიძლია შევქმნათ აღნიშნული მანქანის „Template“, რომლის დახმარებითაც შევძლებთ აღნიშნული *ვირტუალური მანქანის კოპირებას* იმდენჯერ, რამდენჯერაც საჭირო იქნება და გვექნება წინასწარ გამზადებული იმიჯი, რომელსაც მინიმალური ცვლილებებით მოვარგებთ ჩვენს მოთხოვნებს.

4.1.4. ქსელის პარამეტრების გასწორება

ქსელის პარამეტრების გასწორება შესაძლებელია ორი გზით, განვიხილოთ ისინი. იმ შემთხვევაში, თუ ოპერაციულ სისტემაში არ არის ჩაწერილი „NetworkManager-tui“ პროგრამული ხელსაწყო, ჩვენ შეგვიძლია ქსელის პარამეტრები ჩავასწოროთ შემდეგ ფაილებში: „/etc/sysconfig/network-scripts/ifcfg-ens33“.

აღნიშნულ ფაილში არის ქსელის ბარათის პარამეტრები. ამ პარამეტრებით ბარათი მისამართს იღებს ავტომატურ რეჟიმში იმისათვის, რომ შევძლოთ ქსელის პარამეტრების ხელით მითითება. ამ ფაილში უნდა ჩავასწოროთ კონფიგურაციის პარამეტრები. „BOOTPROTO=”dhcp”“ ჩანაცვლდება „BOOTPROTO=”static”“-ით და ასევე დაემატება შემდეგი პარამეტრები:

- „IPADDR=192.168.111.130“ ქსელის მისამართი;
- „NETMASK=255.255.255.0“ ქსელის მასკა,
- „GATEWAY=192.168.111.1“ ლოკალური ქსელიდან გასვლის წერტილი (ნახ.4.9).

```
TYPE="Ethernet"
PROXY_METHOD="none"
BROWSER_ONLY="no"
BOOTPROTO="dhcp"
DEFROUTE="yes"
IPV4_FAILURE_FATAL="no"
IPV6INIT="no"
IPV6_AUTOCONF="yes"
IPV6_DEFROUTE="yes"
IPV6_FAILURE_FATAL="no"
IPV6_ADDR_GEN_MODE="stable-privacy"
NAME="ens33"
UUID="a14c6b1d-f2b3-42b4-a4ef-e38941950038"
DEVICE="ens33"
ONBOOT="yes"
```

```
TYPE="Ethernet"
PROXY_METHOD="none"
BROWSER_ONLY="no"
BOOTPROTO=" static "
DEFROUTE="yes"
IPV4_FAILURE_FATAL="no"
IPV6INIT="no"
IPV6_AUTOCONF="yes"
IPV6_DEFROUTE="yes"
IPV6_FAILURE_FATAL="no"
IPV6_ADDR_GEN_MODE="stable-privacy"
NAME="ens33"
UUID="a14c6b1d-f2b3-42b4-a4ef-e38941950038"
DEVICE="ens33"
ONBOOT="yes"
IPADDR=192.168.111.130
NETMASK=255.255.255.0
GATEWAY=192.168.111.1
```

ნახ.4.9. ქსელის პარამეტრები

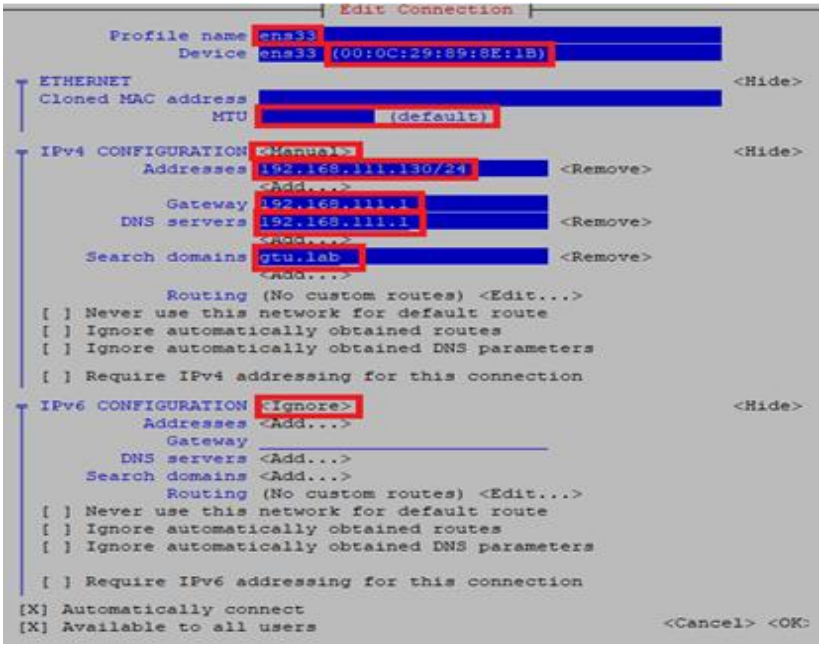
ქსელის პარამეტრების გაწერისას ყურადღება უნდა მიექცეს „DNS“ სერვერის მითითებას. „/etc/resolv.conf“ კონფიგურაციის ფაილში „vi“ ტექსტური რედაქტორის დახმარებით უნდა მივუთითოთ „nameserver 8.8.8.8“ სერვერის „IP“ მისამართი, რის შემდეგადაც „systemctl restart network“ ბრძანებით დავარესტარტებთ ქსელის ბარათის პარამეტრებს.

იმ შემთხვევაში, როდესაც სერვერზე არის დაყენებული „NetworkManager-tui“, მაშინ ქსელის პარამეტრების შეცვლა რადიკალურად განსხვავებული მეთოდით ხდება. კრიტიკული კვლევის შედეგად მივედით იმ დასკვნამდე, რომ ქსელის პარამეტრების სამართავად სასურველია გამოვიყენოთ ხელსაწყო „NetworkManager-tui“ [55]. ხელსაწყოს ჩაწერისათვის საჭიროა გავუშვათ შემდეგი ბრძანება „yum install NetworkManager-tui“. ჩაწერის დასრულების შემდეგ შეიძლება გამოვიყენოთ ახალი ხელსაწყო. ხელსაწყოს მართვის ბრძანებები იხილეთ 4.5 ცხრილში.

ქსელის სამართავი ბრძანებები **ცხრ.4.5**

ბრძანება	განმარტება
systemctl stop NetworkManager	სერვისის გაჩერება
systemctl start NetworkManager	სერვისის გააქტიურება
systemctl restart NetworkManager	სერვისის რესტარტი
systemctl status NetworkManager	სერვისის მიმდიანრე მდგომარეობის გამოტანა
systemctl diasble NetworkManager	ჩართული სერვისის გამორთვა
systemctl enable NetworkManager	გამორთული სერვისის ჩართვა
nmcli device status	გვიჩვენებს ქსელის ადაფტერების მიმდინარე მდგომარეობას
nmtui	ქსელის პარამეტრების მართვის მენიუ
nmtui edit ens33	კონკრეტული სქელის ბარათის პარამეტრების შეცვლა

ქსელის პარამეტრების გასწორება შსაძლებელია „nmtui“ ხელსაწყოს დახმარებით. „nmtui“ ბრძანების შეყვანის შემდეგ იხსნება ხელსაწყო, რომლის დახმარებითაც შესაძლებელია ოპერაციული სისტემის სახელის შეცვლა და ქსელის ბარათის პარამეტრების შეცვლა. იმისათვის რომ შევცვალოთ ოპერაციული სისტემის სახელი, შევდივართ „set system hostname“ განყოფილებაში, ხოლო ქსელის პარამეტრების შესაცვლელად შევდივართ „Edit a connection“, სადაც ვირჩევთ ჩვენთვის სასურველ ქსელის ბარათს და ვიწყებთ პარამეტრების შეცვლას. აღნიშნული ხელსაწყოს დახმარებით შევიყვანეთ პარამეტრები: ქსელის ბარათის სახელი, ფიზიკური მისამართი, „MTU“ ქსელის გამტარობის ერთეული, „IPv4“ მისამართი, სხვა ქსელში გასვლის წერტილი. ხოლო რაც შეეხება IPv6 სტანდარტს, არ ვიყენებთ შესაბამისად „Ignore“ პარამეტრით გავთიშეთ აღნიშნული ფონქციონალი (ნახ.4.10).



ნახ.4.10. ქსელის პარამეტრების გასწორება

4.1.5. „Firewall“_ის პარამეტრების გასწორება

Linux-ის ერთ-ერთი ყველაზე ძლიერი ხელსაწყო არის „Firewall“, რომელიც გვაძლევს საშუალებას დავიცვათ ოპერაციული სისტემა არასასურველი და არაავტორიზებული წვდომისგან.

აღნიშნული ხელსაწყო თავის ფუნქციონალით არ ჩამოუvარდება ფიზიკურ „Firewall“ მოწყობილობებს. შესაბამისად, პროფესიონალის ხელში ძლიერ იარაღად იქცევა. ქსელის პარამეტრების გასწორებასთან ერთად აუცილებელია გავასწოროთ „Firewall“-ის პარამეტრებიც. ქსელური წვდომის საშუალება უნდა დავუტოვოთ მხოლოდ იმ სერვისებს, რომლებსაც ეს სჭირდება. ამით თავიდან ავირიდებთ მთელ რიგ ქსელურ შეტევებს, ყურადღებას გავამახვილებთ კონკრეტულ სერვისებზე, რაც გაგვიადვილებს ანომალიური შემთხვევების აღმოჩენას და შეამცირებს ინციდენტებზე რეაგირების დროს [60]. ხელსაწყოს მართვისთვის ბძანებები მოცემულია 4.7 ცხრილში.

Firewall_ის მართვის ბრძანებები **ცხრ.4.4**

ბრძანება	აღწერა
firewall-cmd --state	
firewall-cmd --reload	
firewall-cmd --list-ports	
firewall-cmd --get-default-zone	
firewall-cmd --get-active-zones	
firewall-cmd --list-all	გამოაქვს მიმდინარე პარამეტრების ჯამური ინფორმაცია.
firewall-cmd --get-zones	
firewall-cmd --zone=home --list-all	
firewall-cmd --list-all-zones | more	
firewall-cmd --get-services	
firewall-cmd --zone=BekaPhd --list-services	

აღნიშნული ხელსაწყოს ქსელური უსაფრთხოების მართვის პრინციპები არის ზონაზე ორიენტირებული. იმისათვის, რომ მარტივად შევძლოთ პარამეტრების მართვა,

ჩვენ ვქმნით „xml“ ფაილებს, რომლებშიც არის აღწერილი სერვისის ფუნქციონირებისთვის საჭირო პარამეტრები, თუ როგორ უნდა მოექცეს „Firewall“ პაკეტებს, რომლებიც აღნიშნულ სერვისს მოაკითხავენ.

ჩვენი სერვერისთვის დავწერეთ შემდეგი ფაილები „proxy-dhcp.xml“, „squid.xml“, „http“, „https“, „ftp“, „ssh“. რომლებიც მივაბით ჩვენ „public“ ზონას შემდეგი ბრძანებებთ:

- „firewall-cmd --zone= BekaPhd --add-service=proxy-dhcp --permanent“
- „firewall-cmd --zone= BekaPhd --add-service=squid --permanent“
- „firewall-cmd --zone= BekaPhd --add-service=http --permanent“
- „firewall-cmd --zone= BekaPhd --add-service=https --permanent“
- „firewall-cmd --zone= BekaPhd --add-service=Mongodb --permanent“
- „firewall-cmd --zone= BekaPhd --add-service=ftp --permanent“
- „firewall-cmd --zone= BekaPhd --add-service=ssh --permanent“
- „firewall-cmd --zone= BekaPhd --add-port=10000-15000/udp --permanent“

შესაბამისად, ქსელის ადაფტერები, რომლებიც გაწევრიანდება „public“ ზონაში, იმუშავებს ამ ბრძანებებით. ყველა ბრძანების გაშვების შემდეგ ჩვენი „firewll“-ის პარამეტრები ასე გამოიყურება (ნახ.4.11).

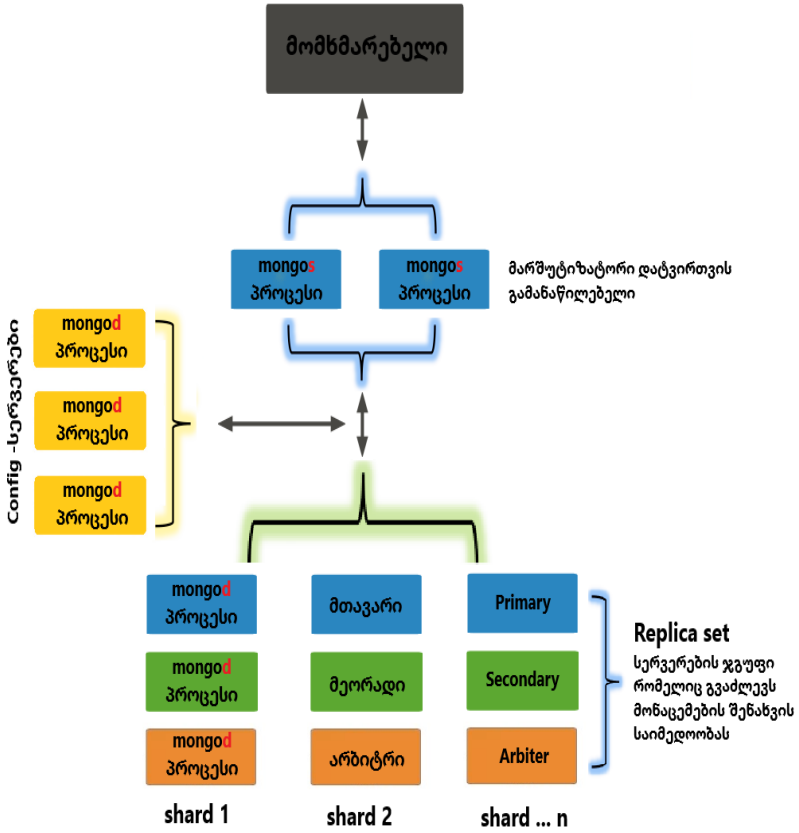
```
[root@CentOS ~]# firewall-cmd --list-all
public (active)
  target: default
  icmp-block-inversion: no
  interfaces: ens33
  sources:
  services: ssh dhcpv6-client proxy-dhcp squid http https ftp
  ports: 10000-15000/udp
  protocols:
  masquerade: no
  forward-ports:
  source-ports:
  icmp-blocks:
  rich rules:
```

ნახ.4.11. Firewall_ის პარამეტრები

4.2. მონაცემთა ბაზის ინფრასტრუქტურის გამართვა

ქსელის და „Firewall“-ის პარამეტრების გასწორების შემდეგ, ძირითადი სამუშაოები შესრულებულია. ამის შემდეგ უნდა მოხდეს სერვერის კონკრეტული დანიშნულებისთვის მომზადება. წიგნის ფარგლებში, მონაცემების შესანახად, კრიტიკული ანალიზის შედეგად, ავირჩიეთ დოკუმენტზე ორიენტირებული NoSQL მონაცემთა ბაზა MongoDB, რომელშიც რეალიზებადია ჰორიზონტალური ზრდის მოდელი.

MongoDB მონაცემთა ბაზის ეკოსისტემის გასამართად გამზადებულ *ვირტუალურ მანქანაზე* გავმართეთ მონაცემთა ბაზის ინფრასტრუქტურა. მას ვიყენებთ ეკოსისტემის პლატფორმად. აღნიშნული ეკოსისტემა შედგება მთელი რიგი სერვერებისგან (ნახ. 4.12).



ნახ.4.12. Mongodb ეკოსისტემის ლოგიკური სქემა

როგორც ნახაზზეა ნაჩვენები, მთლიანი ეკოსისტემა შედგება პროცესებისგან: „Mongos“ პროცესი რომელსაც უკავშირდება მომხმარებელი, მისი მთავარი ფუნქციაა მომხმარებლისგან მიიღებული ტრანზაქციების შესრულება. მომხმარებელი იყენებს მთლიანი ეკოსისტემის რესურსს, მაგრამ იგი ურთიერთობს

მხოლოდ „Mongos“ სერვისთან, რომელიც აკონტროლებს ეკოსისტემაში არსებულ პროცესებს [61].

ასევე გვაქვს „mongod“ პროცესი, რომელიც გამოიყენება სხვადასხვა მიზნისთვის: მონაცემების შესანახად, არბიტრად და „config“ სერვერადაც.

საიმედოობის ასამაღლებლად გამოიყენება „replica set“, რომელიც შედგება მინიმუმ სამი წევრისგან. მათგან შეგვიძლია ორ წევრზე გვქონდეს მონაცემები დუბლირებული, ხოლო მესამე წევრი იყოს არბიტრი, რომელზეც მონაცემები არ არის. მისი ძირითადი ფუნქციაა მთავარი სერვერის წყობიდან გამოსვლის შემთხვევაში დაეხმაროს მეორად სერვერს და გახადოს იგი მთავარ სერვერად. ამრიგად ტექნიკური ხარვეზისგან გამოწვეული შეფერხება მინიმუმამდეა შემცირებული.

Sharding ტექნოლოგია მონაცემთა ბაზის დაყოფის ერთ-ერთი სახეა. „Shard“ ერთი მთლიანის პატარა ნაწილს ნიშნავს. შარდინგის დახმარებით დიდი ზომის უმართავი ბაზა იყოფა მცირე ზომის, სწრაფ და ადვილად მართვად ნაწილებად [61,62]. რაც შეეხა Shard-ს, ისინი შედგება „replica set“-ებისგან. მათი ძირითადი გამოყენებაა არსებული ინფორმაციის დაყოფა, მაგალითად, შეგვიძლია ერთ Shard-ზე შევინახოთ ერთი რეგიონის ინფორმაცია, ხოლო მეორე Shard-ზე კი - მეორე რეგიონის. ეს ტექნოლოგია გვაძლევს საშუალებას, საჭიროების შემთხვევაში, ჰორიზონტალურად გავზარდოთ რესურსი.

პროცესი მონაცემებს ინახავს ჩანკებში (chunks), რომლის რეკომენდებული ზომაა 64 Mb, მას შემდეგ რც ჩანკი მაქსიმალურ ზომას მიაღწევს, იგი იყოფა ორ ნაწილად. შესაბამისად გამოდის, რომ ჩვენი მონაცემები არის ჩაწერილი სხვადასხვა ჩანკში, რომლებიც შეიძლება იყოს განაწილებული სხვადასხვა Shard-ზე. შესაბამისად აქ გვჭირდება „config“ სერვერები, რომლებმაც იცის თუ რომელ ჩანკში რა ინფორმაცია წერია და აღნიშნული ჩანკი რომელ სერვერზე იმყოფება.

კრიტიკული ანალიზისა და კვლევის შედეგად შევიმუშავეთ MongoDB მონაცემთა ბაზის პროცესების განაწილების ჩვენთვის უმჯობესი არქიტექტურა, ეკოსისტემის გასამართად გამოვიყენეთ 9 ვირტუალური სერვერი, რომლებზეც გავანაწილეთ პროცესები შემდეგი სახით: შევქმენით სერვერების სამი ჯგუფი („replica set“), რომლებიც დავყავით Shard-ებად (shard 1, shard 2, shard 3). თითოეულ სერვერზე გაშვებულია მინიმუმ ორი პროცესი, რომლებიც დავაჯგუფეთ შემდეგი თანამიმდევრობით:

- ვირტუალური სერვერი 1- „mongos, mongod“ პროცესები;
- ვირტუალური სერვერი 2-„mongos, mongod“ პროცესები;
- ვირტუალური სერვერი 3-„mongos, mongod“ პროცესები;
- ვირტუალური სერვერი 4 -„mongos, mongod“ პროცესები;
- ვირტუალური სერევრი 5-„mongos, mongod“ პროცესები;
- ვირტუალური სერვერი 6-„mongos, mongod“ პროცესები;
- ვირტუალური სერვერი 7-„mongod“ არბიტრ და „config“ პროცესები;
- ვირტუალური სერვერი 8-„mongod“ არბიტრ და „config“ პროცესები;
- ვირტუალური სერვერი 9-„mongod“ არბიტრ და „config“ პროცესები.

ლოგიკური და ფიზიკური სტრუქტურის განაწილების შემდეგ დავიწყეთ სერვერების მომზადება. იმისათვის, რომ ერთი-დაიგივე სამუშო ყველა სერვერზე არ გვეკეთებინა, გადავწყვიტეთ ერთ სერვერზე დაგვეყენებინა „Mongodb“ მონაცემთა ბაზის ინფრასტრუქტურა და ამის შემდეგ გაგვემრავლებინა ეს სერვერი.

მონაცეთა ბაზის ინფრასტრუქტურის ინსტალაციისათვის საჭიროა https://docs.mongodb.com ოფიციალური გვერდიდან გადმოვიწეროთ რეპოზიტორის ფაილი და ჩვენ სერვერში ჩავწეროთ. ამისათვის უნდა შევქმნათ ფაილი სახელით „mongodb-org-4.0.rep“ რეპოზიტორების საქაღალდეში „/etc/yum.repos.d/“. ფაილის შესაქმნელად ვუშვებთ შემდეგ ბრძანებას:

„touch /etc/yum.repos.d/mongodb-org-4.0.repo “.

მას შემდეგ, რაც ფაილი შეიქმნება, უნდა გავხსნათ ტექსტური რედაქტორით და ჩავწეროთ შიგნით რეპოზიტორის მონაცემები. „vi“ ტექსტური რედაქტორის დახმარებით ვხსნით ფაილს ბრძანებით („vi /etc/yum.repos.d/mongodb-org-4.0.repo“). იმისათვის, რომ ფაილში მონაცემების შეტანა მოვახერხოთ „i“ ღილაკზე დაჭერით ტექსტური რედაქტორი გადაგვყავს ჩაწერის რეჟიმში და ვწერთ შემდეგ სტრიქონებს (ნახ.4.13).

```
[mongodb-org-4.0]
name=MongoDB Repository
baseurl=https://repo.mongodb.org/yum/redhat/$releasever/mongodb-org/4.0/x86_64/
gpgcheck=1
enabled=1
gpgkey=https://www.mongodb.org/static/pgp/server-4.0.asc
```

ნახ.4.13. Mongodb Roposítory

სტრიქონების შეტანისა და „Esc“ ღილაკზე დაჭერის შემდეგ, ტექსტური რედაქტორი გადაგვყავს ბრძანებების შეტანის რეჟიმში. ვუშვებთ „wq“ ბრძანებას, რომელიც შეინახავს და გათიშავს ფაილს.

რეპოზიტორის დამატების შემდეგ ვუშვებთ ბრძანებას:

„sudo yum install -y mongodb-org“,

რომელიც დააყენებს მონაცემთა ბაზის ინფრასტრუქტურას. ინფრასტრუქტურის ინსტალაციის დასრულდების შემდეგ შეგვიძლია დავიწყოთ სერვერების გამართვა.

„VMware” ვირტუალიზაციის დახმარებით შექმნილი *ვირტუალური მანქანა* დავადუბლირეთ ცხრა ეგზემპლარად. სწორედ ამდენი ვირტუალური მანქანაა მინიმუმ საჭირო „Mongodb“-ს ინფრასტრუქტურამ გამართულად რომ იმუშაოს.

აღნიშნულ სერვერებს შევუცვალეთ სახელები და დავარქვით შემდეგი მეთოდით. „server“-ს დავუმატეთ სერვერის „ip“ მისამართის ბოლო ოქტეტის ციფრები, რისთვისაც გამოვიყენეთ

„nmtui“ ხელსაწყო. სახელებისა და ქსელის მისამართის გასწორების შემდეგ სერვერებზე შევქმენით საქაღალდეები „mkdir -p /data/r0“ ბრძანებით, რომლებსაც გამოიყენებს მონაცემთა ბაზა.

„-p“ პარამეტრი რეკურსიულად შექმნის ყველა საჭირო ქვე საქაღალდეს. ასევე შევქმენით სამი ტიპის პარამეტერების ფაილები („conf“, „mongod“, „mongos“), რომლებშიც აღწერილია თუ როგორ უნდა მუშაობდეს მონაცემთა ბაზა (ცხრილი 4.8).

მონაცემთა ბაზის პარამეტრები ცხრ.4.5.

სერვერის სახელი	ip მისამათი	საქაღალდეები	ფაილები
server101	192.168.111.101 255.255.255.0	/data/r0 /data/mongos/log	mongod-101.cfg mongos-101.cfg
server102	192.168.111.102 255.255.255.0	/data/r0 /data/mongos/log	mongod-102.cfg mongos-102.cfg
server103	192.168.111.103 255.255.255.0	/data/r0 /data/csRS/log	mongod-103.cfg conf-103.cfg
server104	192.168.111.104 255.255.255.0	/data/r1 /data/mongos/log	mongod-104.cfg mongos-104.cfg
server105	192.168.111.105 | 255.255.255.0	/data/r1 /data/mongos/log	mongod-105.cfg mongos-105.cfg
server106	192.168.111.106 | 255.255.255.0	/data/r1 /data/csRS/log	mongod-106.cfg conf-106.cfg
server107	192.168.111.107 | 255.255.255.0	/data/r2 /data/mongos/log	mongod-107.cfg mongos-107.cfg
server108	192.168.111.108 | 255.255.255.0	/data/r2 /data/mongos/log	mongod-108.cfg mongos-108.cfg
server109	192.168.111.109 | 255.255.255.0	/data/r2 /data/csRS/log	mongod-109.cfg conf-109.cfg

განვიხილოთ სამივე ტიპის პარამეტრების ფაილები:
mongod-*.cfg - აღნიშნული პროცესი უზრუნველყოფს მონაცემების შენახვას და რეპლიკაციას. გავარჩიოთ „server101“-ის კონფიგურაციის ფაილის მაგალითი.

- net: ქსელის პარამეტრები
  - bindIp: 192.168.111.101 ip მისამართი რომელსაც გამოიყენებს პროცესი
  - port: 27018 ქსელის პორტი
- processManagement: პროცესის მართვა
  - fork: true სერვისად გაშვება
- systemLog: ლოგ ჩანაწერების მართვა
  - destination: "file" ლოგ ჩანაწერების ფორმატი
  - path: "/data/r0/log" ლოგების ფაილი
  - logAppend: true ერთიდაიმავე ლოგ-ფაილის გამოყენებისთვის
- storage: მონაცემების შენახვის პარამეტრები
  - dbPath: "/data/r0" მონაცემთა შენახვისთვის განკუთვნილი საქაღალდე
  - wiredTiger: მყარი დისკოს დაერთების ტიპის განსაზღვრა
- engineConfig: პარამეტრები
- cacheSizeGB: 0.5 ქეშის ზომა დამოკიდებულია დისკების დაერთების მეთოდზე
- sharding: შარდინგის პარამეტრები
  - clusterRole: shardsvr შარდინ კლასტერში პროცესის როლი (ფუნქცია)
- replication: რეპლიკაციის პარამეტრები
  - oplogSizeMB: 10 ოპერაციული ლოგების ზომა
  - replSetName: r0 რეპლკა სეტის სახელი

mongos-*.cfg – პროცესი, რომელიც ასრულებს მარშუტიზატორის ფუნქციას მონაცემებსა და მომხმარებელს შორის, ინფორმაციის მიმოცვლაზე არის პასუხისმგებელი:

- net: ქსელის პარამეტრები
  - bindIp: 192.168.111.101 ქსელის მისამართი რომელსაზ პროცესი მოუსმენს
  - port: 27017 ქსელის პორტი
- processManagement: პრიცესი მართვის პარამეტრები
  - fork: true სერვისად გაშვება
- systemLog: ლოგ ჩანაწერების მართვა
  - destination: "file" ჩანაწერების ფორმატი
  - path: "/data/mongos/log" ჩანაწერებისთვის განკუთვნილი საქაღალდე
  - logAppend: true ერთიდა იგივე ფაილში ჩანაწე-რების ჩამატება
- sharding: შარდინგის პარამეტრები
  - configDB: სერვერის როლი
  - - 'csRS/192.168.111.103:27019' პარამეტრების სერვერების მისამართი
  - -'csRS/192.168.111.106:27019'
  - –'csRS/192.168.111.109:27019'

conf-*.cfg – პროვესი, რომელიც უზრუნველყოფს მონაცემების განაწილებას ეკოსისტემაში:

- net: ქსელის პარამეტრების მართვა
  - bindIp: 192.168.111.103 ქსელის მისამართი
  - port: 27019 ქსელის პორტი
- processManagement: პროცესის მართვა
  - fork: true სერვისად გაშვება
- systemLog: ლოგ ჩანაწერების მართვა
  - destination: "file" ჩანაწერების ფორმატი
  - path: "/data/csRS/log" ჩანაწერებისთვის განკუთვნილი საქაღალდე
  - logAppend: true ერთ ფაილში ჩანაწერების ჩამატება

- storage: მონაცემების შენახვის მართვა
  - dbPath: "/data/csRS" მონაცემებისთვის განკუთვნილი საქაღალდე
    - wiredTiger: მყარ დისკოზე დაერთების ტიპი
    - engineConfig: პარამეტრები
    - cacheSizeGB: 0.5 ქეშის ზომა დამოკიდებული დისკოების დაერთების მეთოდზე
- sharding: შარდინგ გარემოში სერვერების როლი
  - lusterRole: configsvr პარამეტრების სერვერების როლი
- replication: რეპლიკაციის მართვა
  - oplogSizeMB: 10 ოპერეციული ლოგების ზომა
  - replSetName: csRS რეპლიკასეტის დასახელება

ეს ის ძირითადი პარამეტრებია, რომლებიც აუცილებელია ეკოსისტემის ფუნქციონირებისთვის. მას შემდეგ, რაც ყველა სერვერზე ჩავწერთ და გავმართავთ პარამეტრებს, შეგვიძლია გავუშვათ შემდეგი ბრძანებები (ცხრილი 4.9).

მონაცემთა ბაზის ეკოსისტემის გაშვება **ცხრ.4.6**

სერვერები	ბრძანებები
server101	mongod -f mongod-101.cfg | mongos -f mongos-101.cfg
server102	mongod -f mongod-102.cfg | mongos -f mongos-102.cfg
server103	mongod -f mongod-103.cfg | mongod -f conf-103.cfg
server104	mongod -f mongod-104.cfg | mongos -f mongos-104.cfg
server105	mongod -f mongod-105.cfg | mongos -f mongos-105.cfg
server106	mongod -f mongod-106.cfg | mongod -f conf-106.cfg
server107	mongod -f mongod-107.cfg | mongos -f mongos-107.cfg
server108	mongod -f mongod-108.cfg | mongos -f mongos-108.cfg
server109	mongod -f mongod-109.cfg | mongod -f conf-109.cfg

იმისათვის, რომ ჩვენმა სისტემამ მუშაობა შეძლოს, აუცილებელია შემდეგი თანმიმდევრობის დაცვა: ჯერ ვრთავთ რეპლიკა სეტების მთავარ სერვერებს, რომლებსაც „MongoDB Compass“ აპლიკაციით ვუკავშირდებით და ვახდენთ რეპლიკა სეტის ინიცირებას, რის შემდეგაც ვრთავთ დანარჩენ სერვერებს.

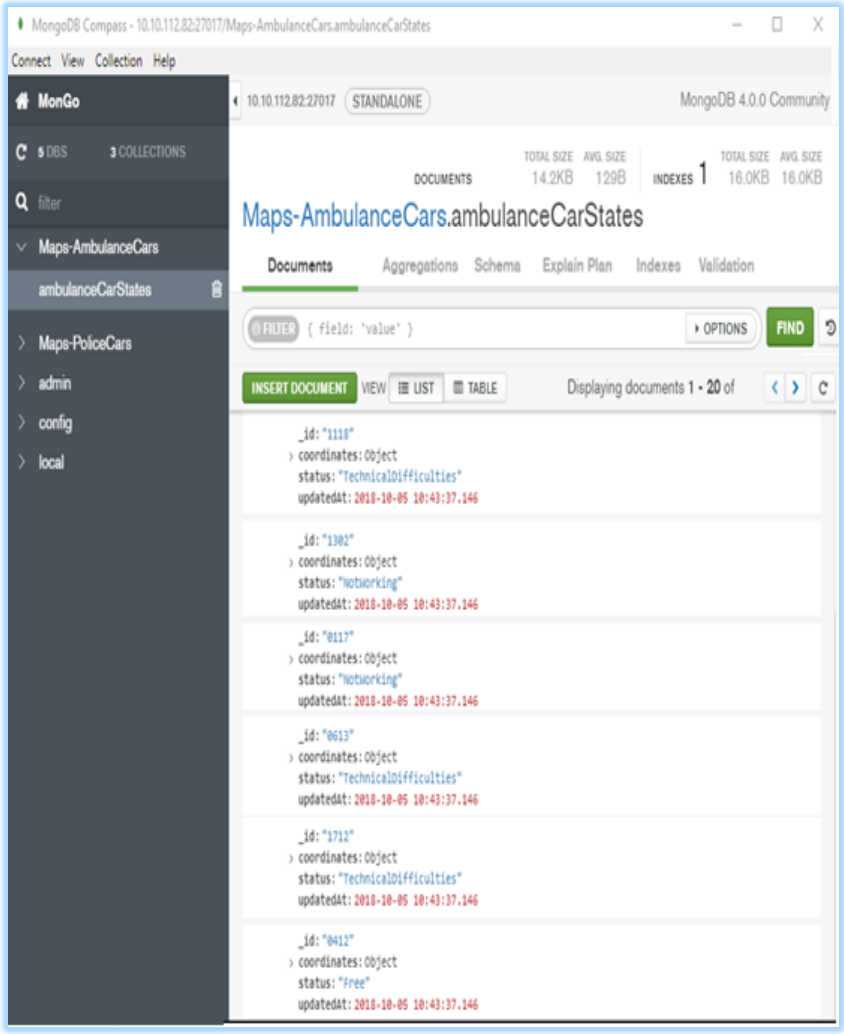
სერვერების პარამეტრების გასწორებისას ერთ-ერთი ყველაზე აუცილებელი და ყურადსაღებია ოპერეციული სისტემის პარამეტრები, რომლებიც აკონტროლებს ოპერეციულ სისტემაში პროცესებს და უზღუდავს მათ გახსნილი ფაილების რაოდენობას.

შესაბამისად, იმის გათვალისწინებით, რომ ჩვენ სერვერზე გაშვებული იქნება მონაცემთა ბაზის პროცესები და ამ ბაზას ეყოლება ბევრი ერთდროული მომხმარებელი, აუცილებელია ამ პარამეტრების მაქსიმალურამდე გაზრდა.

ამისათვის შევდივართ „vi /etc/security/limits.conf“ ფაილში და ვცვლით შემდეგ პარამეტრებს:

„ * soft nofile 65536“ ,
„ * hard nofile 65536“.

ამრიგად ჩვენი ეკოსისტემა მზად არის მონაცემებთან სამუშაოდ (ნახ.4.14).



ნახ.4.14. Mongodb Compass

4.3. მონაცემების დამუშავება

მონაცემთა დამუშავება განაწილებულ გარემოში პირობითად გავყავით ორ ნაწილად:

როდესაც გვაქვს დიდი რაოდენობის ინფორმაცია და გვინდა მისი ანალიზი „Apache Spark“ პლატფორმის დახმარებით, რომელიც არის ღია პლატფორმა. მისი გამოყენებით შესაძლებელია დიდი რაოდენობის მონაცემების ანალიზი.

„Apache spark“ შედგება ორი ტიპის სერვერისგან, კერძოდ, მთავარი სერვერი, რომელიც უზრუნველყოფს მონაცემების განაწილებას სხვადასხვა მუშა სერვერზე. მუშა სერვერები პარალელურ რეჟიმში ამუშავებს მონაცემებს და შედეგად ვიღებთ მძლავრ სისტემას, რომელსაც შეუძლია დაამუშაოს ტერაბაიტობით მონაცემები.

რა თქმა უნდა, არის მთელი რიგი შემთხვევები, როდესაც ჩვენთვის საინტერესო სტატისტიკის ამოღება შესაძლებელია ვერ მოხდეს სპარკის დახმარებით, რადგანაც ყველა ალგორითმის რამდენიმე სერვერზე განაწილება ხშირ შემთხვევაში შეუძლებელია.

წიგნის ფარგლებში ავაგეთ პროგრამული პროდუქტი, რომლის დახმარებითაც ვახდენთ მონაცემების დამუშავებას. აღნიშნული პროგრამული პროდუქტი დაწერილი გვაქვს მიკრო სერვისების პრინციპზე, რაც იმას ნიშნავს, რომ ერთი მთლიანი დავალება დანაწევრებულია სერვისების მიხედვით და ამოცანის შესასრულებლად ვიყენებთ სხვადასხვა „REST FULL“ სერვისებს, რომლებიც ერთმანეთთან ურთიერთობს „http/https“ პროტოკოლის მეშვეობით.

შესაბამისად აღნიშნულმა არქიტექტურამ საშუალება მოგვცა ეს „web“ სერვისები გაგვეშვა „kubernetes“ *ვირტუალიზაციის სისტემაში* და კუბერნეიტის პოდების დახმარებით მოგვეხდინა აპლიკაციისთვის საჭირო რესურსის მართვა.

„Kubernetes“ ღია სისტემაა, რომელიც კონტეინერიზებული აპლიკაციების განლაგების, სკალირებისა და მართვის ავტომატიზებას ახდენს. იგი თავდაპირველად შეიქმნა Google-ს მიერ და ახლა მის განვითარებაზე ზრუნავს „Cloud Native Computing Foundation“. მისი მიზანია პლატფორმების გაშვების, სკალირების და ოპერაციების ავტომატიზება პლატფორმების კლასტერებზე. ის იყენებს მთელ რიგ კონტეინერებზე ორიენტირებულ ინსტრუმენტებს, მათ შორის Docker-ს.

აღნიშნული პლატფორმის დახმარებით ჩვენს მიერ დაწერილ პროგრამული პროდუქტის სამართავად საჭიროა, რომ ჩვენი პროექტი მოვათავსოთ „Dorcker”-ის კონტეინერში, რის შემდეგადაც მას გავუშვებთ „kubernetes“ ვირტუალურ გარემოში.

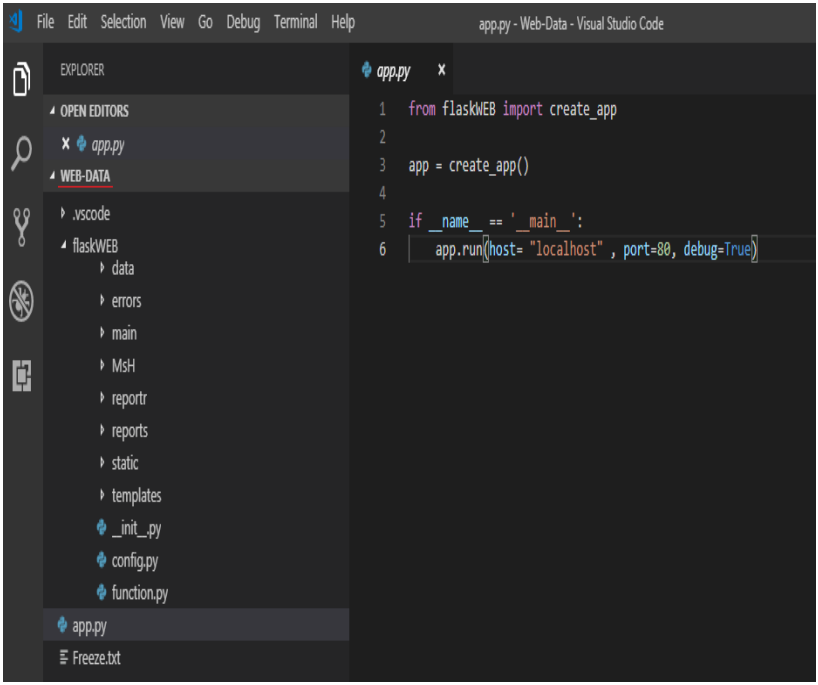
აღნიშნული აპლიკაციის დასაწერად გამოვიყენეთ ერთ-ერთი პოპულარული მრავალპარადიგმული პროგრამირების ენა „Python“, რომელიც იდეალური ხელსაწყოა მონაცემებთან სამუშაოდ. იგი არის მაღალი დონის პროგრამირების ენა რომელიც კოდის გაშვებას უზრუნველყობს ინტერპრეტატორის საშუალებით, რაც პროგრამირების ენას ხდის მულიტიპლატფორმულს და მასზე დაწერილი კოდი სხვადასხვა სისტემებში თითქმის ერთნაირად მუშაობს.

4.3.1. პროგრამული პროდუქტი

წიგნში წარმოდგენილი პროგრამული პროდუქტის შექმნის მთავარი მიზანია ერთ-ერთი ქართული კომპანიის მიერ დაგენერირებული ლოგების ანალიზის ავტომატიზება. აღნიშნული კომპანიის თანამშრომლბს უწევდათ დიდი რაოდენობის ლოგ ფაილების ხელით ანალიზი და სტატისტიკის გამოთვლა, რაც ხშირ შემთხვევათი დიდ რესურსსა და დროს მოითხოვდა. სამუშაოს სირთულისა და მასშტაბების გამო ხშირად იყო ადამიანური ფაქტორით გამოწვეული უზუსტობები.

ყოველივე ამის გათვალისწინებით გადავწყვიტეთ რეპორტინგის სისტემის დაეწერა, რომელიც შეამცირებდა სტატისტიკის

დათვლის დროსა და მინიმუმამდე დაიყვანდა ადამიანური ფაქტორით გამოწვეულ შეცდომებს. ამისათვის გამოვიყენეთ „Python“ პროგრამირების ენა და თანამედროვე მიდგომები, ისეთი როგორიცაა სუფთა კოდის კონცეფცია. მოვახდინეთ როპორტინგის სისტემის პროგრამული კოდის ლოგიკური დაცალკევება და ფუნქციონალის მიხედვით გავანაწილეთ სხვადასხვა საქაღალდეებსა და ფაილებში (ნახ.4.15).



ნახ.4.15. პროგრამული კოდი

პროგრამული კოდი მოთავსებულია მთავარ „WEB-DATA“ საქაღალდეში, რომელიც ლოგიკურად გაყოფილია ორ ნაწილად „flaskWEB“ საქაღალდეში მოთავსებულია პროგრამის ძირითადი კოდი, ხოლო „app.py“ ფაილი არის მთავარი გამშვები ფაილი,

რომელიც უზრუნველყოფს პროექტის გაშვებას. აღნიშნული ფაილის პროგრამული კოდი იძახებს „create_app“ ფუნქციას და გადასცემს პარამეტრებს, თუ რომელ ქსელის მისამართზე და პორტზე იყოს გაშვებული, ასევე კოდის მუშაობის რეჟიმს (ნახ.4.15).

რაც შეეხება „Freeze.txt“ ფაილს, აღნიშნულ ფაილში მოთავსებულია იმ არასტანდარტული (ბიბლიოთეკები, რომლებიც „Python“ პროგრამირების ენის სამუშაო გარემოს გამართვისას არ მოყვება და მათი დაყენება არის საჭირო „pip“ ხელსაწყოს დახმარებით) ბიბლიოთეკების სია, რომელსაც იყენებს აღნიშნული პრექტი (ცხრილი 4.10).

Python პროგრამირების ენის კომპონენტების ინსტალაცია ცხრ. 7

დასახელება	აღწერა
pip install pandas==0.22.0	მონაცემების დამუშვება ანალიზი
pip install Flask==0.12.2	აპლიკაციის Web მიკრო პლატფორმა
pip install requests==2.18.4	URL ლინკებთან მუშაობისთვის
pip install xlrd==1.1.0	ექსელის ტიპის ფაილებთან მუშაობისთვის
pip install lxml==4.2.5	XML და HTML ტიპის ფაილებთან მუშობისთვის
pip install beautifulsoup==4-4.6.3	XML და HTML ტიპის მონაცემების პარსირება

რაც შეეხება „flaskWEB“ საქაღალდეს, მასში არის მოთავსებული პროგრამული კოდის ძირითადი ნაწილი, რომელიც ლოგიკურად ორ ნაწილად იყოფა: „Front end“ და „Back end“ [63].

„Front end“ – კოდის ის ნაწილია, რომელიც პასუხისმგებელია „user interface“ და ასევე იმ კოდის ფრაგმენტებზე, რაც უშუალოდ მომხმარებლის კომპიუტერში უნდა გაეშვეს. 4.15 ნახაზის

მიხედვით აღნიშნული პროექტის „Front end“ კოდი არის განაწილებული შემდეგ ფაილებში: „static“, „Template“.

„static“ საქაღალდეში მოთავსებულია შემდეგი ფაილები:

„assets“ – მოთავსებულია ვებგ-ვერდში გამოყენებული გრაფიკული გამოსახულებები;

„css“ – აღნიშნულ საქაღალდეში არის ვებ-გვერდის დიზაინისა და სტილის განმსაზღვრელი „css“ ენაზე დაწერილი კოდის ფაილები;

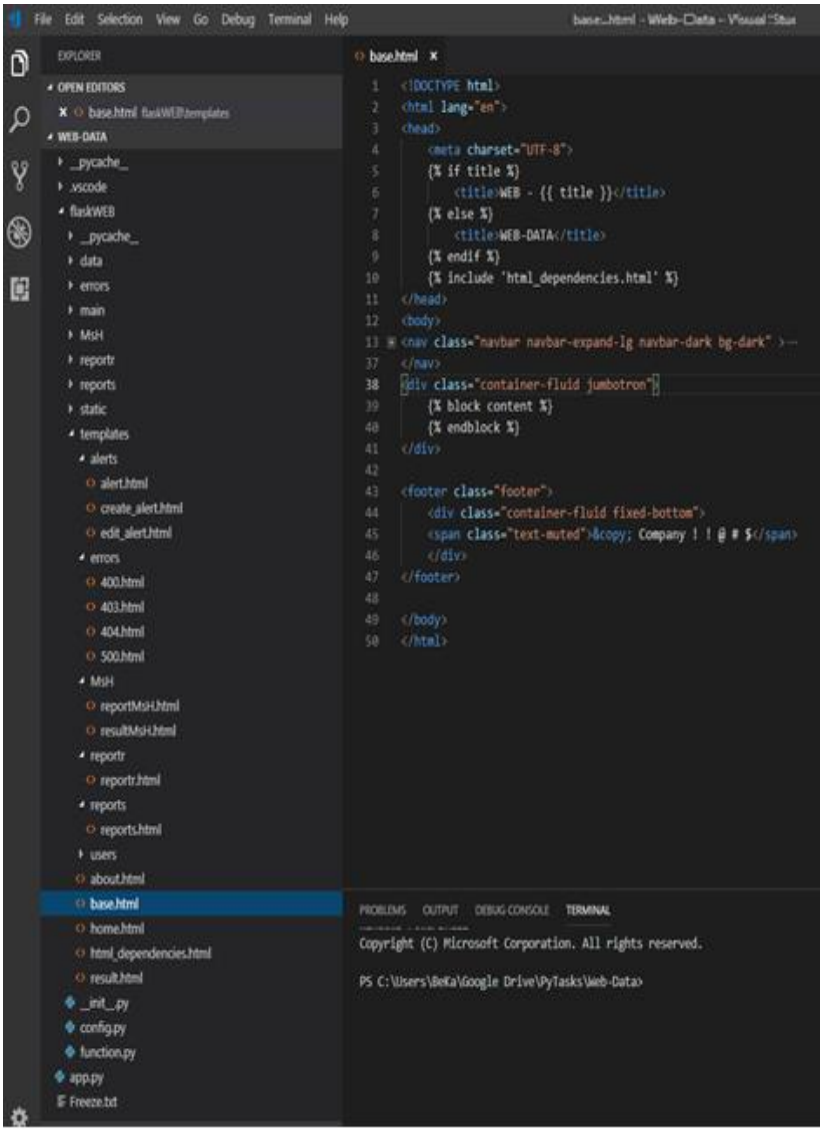
„fonts“ – საქაღალდეში თავმოყრილია ყველა ის ფონტი, რომელსაც ვიყენებთ პროექტის ფარგლებში;

„images“ – მოთავსებულია სხვადასხვა „png“ ფორმატის გამოსახულებები, რომლებსაც ვიყენებთ „UI“-ის ფარგლებში.

„js“ – საქაღალდე, რომელშიც მოთავსებულია JavaScript ფაილები. ისინი ეშვება კლიენტის მხარეს და პასუხისმგებელია როგორც დიზაინზე, ასევე სხვადახვა ფუნქციონალზეც;

„Template“ – საქაღალდე შეიცავს „HTML“ ფაილებს. ამ ფაილებისგან შედგება მთლიანი ვებ-გვერდი. საქაღალდეში შემდეგი ლოგიკით გვაქვს დაჯგუფებული „HTML“ ფაილები. აღნიშნულ საქაღალდეში ძირითად ფაილებად მოიაზრება „base.html“, რომლებიც ვებ-გვერდის ერთგვარი შაბლონია სხვა დანარჩენი გვერდებისათვის (პროგრამული კოდით ნახ.4.16).

როგორც ნახაზზეა ნაჩვენები, აღნიშნულ ფაილში არის აღწერილი თუ როგორი უნდა იყოს ვებ-გვერდი და ფიგურული ფრჩხილებით არის გამოყოფილი ის ადგილი სადაც მოხდება სხვადასხვა გვერდის კოდის ჩამატება.



ნახ.4.16. Visual Studio Code

ნახაზზე ასევე კარგად ჩანს სხვადასხვა გვერდი, რომლებიც არის დაჯგუფებული თემატურად:

- alert:
    - alert.html
    - crate_alert.html
    - edit_alert.html
- errors
    - 400.html
    - 403.html
    - 404.html
    - 500.html
- MsH
    - reportMsH.html
    - resultMSH.html
- reportr
    - reportr.html
- reports
    - reports.html
- users
    - alerts.html
    - login.html
    - register.html

ზემოაღნიშნული დირექტორიები და მაილები ქმნის პროგრამული პროდუქტის იერსახეს, ხოლო იმისათვის, რომ ყველაფერმა იმუშაოს ისე, როგორც ჩვენ გვჭირდება, აუცილებელია „Back end“ ნაწილი, სადაც პროგრამული კოდის ის ფრაგმენტებია, რაც სერვერის მხარეს მუშაობს.

ფუნქციონალის ძირითადი ნაწილი, რაც მონაცემთა დამუშავებას ემსახურება, სრულდება სერვერის მხარეს და მომხმარებელთან უკვე დამუშავებული ინფორმაცია აისახება.

ამ აპლიკაციის მთავარი დანიშნულებაა მომხმარებელს გაუმარტივოს ლოგების დამუშავება. აღნიშნულის მისაღწევად პროგრამის „Beck end“_ში შესრულებული გვაქვს შემდეგი სამუშაოები:

„flaskWeb” საქაღალდის „_init__.py“ ფაილში შექმნილია ფუნქცია „create_app“, რომელიც კონფიგურაციის პარამეტრებს იღებს „config.py“ ფაილიდან და იძახებს „Flask“ აპლიკაციას, სადაც გაწერილია საიტის გვერდების მარშრუტები.

```
1.  from flask import Flask
2.
3.  from flaskWEB.config import Config
4.
5.  def create_app(config_class=Config):
6.      app = Flask(__name__)
7.      app.config.from_object(Config)
8.
9.      from flaskWEB.reportr.routes import reportr
10.     from flaskWEB.reports.routes import reports
11.     from flaskWEB.main.routes import main
12.     from flaskWEB.errors.handlers import errors
13.     #
14.     from flaskWEB.MsH.routes import reportMsH
15.
16.     app.register_blueprint(reportr)
17.     app.register_blueprint(reports)
18.     app.register_blueprint(main)
19.     app.register_blueprint(errors)
20.     app.register_blueprint(reportMsH)
21.
22.     return app
```

განვიხილოთ ლოგების დამუშავების ერთ-ერთი მაგალითი. „MsH“ არის აღნიშნული რეპორტის ფუნქციონალი, კერძოდ, ვებ-გვერდის დახმარებით ლოგ-ფაილების გადაწოდება და კლიენტები-სათვის დამუშავებული სტატისტიკის დაბრუნება. აღნიშნულის მისაღწევად შევქმენით საქაღალდე „MsH“, რომელშიც მოვათავსეთ „MsH.py“ და „routes.py“.

„routes.py“ – ფაილში მოთავსებული კოდი ეშვება ამ კონკრეტული მარშრუტის გამოძახებისას.

```
1.  from flask import render_template, request, redirect, flash, Blue
    print
2.  from flaskWEB.function import remove_files_from_updoad_fol
    der
3.  from werkzeug.utils import secure_filename
4.  from flask import current_app as app
5.  import os
6.  from datetime import datetime, timedelta
7.  from flaskWEB.MsH.MsH import report_func_msh
8.  import numpy as np
9.
10. reportMsH = Blueprint('reportMsH', __name__)
11.
12.
13. ####
14.
15. def ifDateValueNone(value):
16.     if value == '':
17.         value = datetime.now()
18.     else:
19.         value = datetime.strftime(value, '%m/%d/%Y')
20.     return value
```

```
21.
22.
23. @reportMsH.route('/reportMsH', methods=['GET', 'POST'])
24. def MsH():
25.     if request.method == 'POST':
26.         files = request.files.getlist('files')
27.         afterContractEndDate = int(request.form['afterContractEnd
    Date'])
28.         MsHDate = ifDateValueNone(request.form['MsHDate'])
29.         MsHDate = MsHDate - timedelta(days=afterContractEndDa
    te)
30.
31.         Filenames = []
32.         for file in files:
33.             FileName = secure_filename(file.filename)
34.             FileName = FileName.lower()
35. file.save(os.path.join(app.config['UPLOAD_FOLDER'], FileName))
                Filenames.append(FileName)
36.
37.         df_result = report_func_msh(Filenames, MsHDate)
38.
39.         if df_result is not None:
40.             remove_files_from_updoad_folder(Filenames)
41.             return render_template('MsH/resultMsH.html',
42.                         ListAll=[df_result['ListAll'].to_html(index=False).
    replace('<table','<table class="display nowrap" id="ListAll"')],
43.                                 Statistics=[df_result['Statistics'].to_html(ind
    ex=False).replace('<table', '<table class="display nowrap" id="Stat
    istics"')],
44.                         Kids=[df_result['Kids'].to_html(index=False).repla
    ce('<table', '<table class="display nowrap" id="Kids"')],
```

```
45.                     Aqua=[df_result['Aqua'].to_html(index=False).rep
    lace('<table', '<table class="display nowrap" id="Aqua"')],
46.                         WorkOut=[df_result['WorkOut'].to_html(i
    ndex=False).replace('<table', '<table class="display nowrap" id="
    WorkOut"')],
47.                         Jazz=[df_result['Jazz'].to_html(index=False)
    .replace('<table', '<table class="display nowrap" id="Jazz"')],
48.                         Pop=[df_result['Pop'].to_html(index=False)
    .replace('<table', '<table class="display nowrap" id="Pop"')],
49.                         Solo=[df_result['Solo'].to_html(index=False
    ).replace('<table', '<table class="display nowrap" id="Solo"')],
50.                         Gym=[df_result['Gym'].to_html(index=Fals
    e).replace('<table','<table class="display nowrap" id="Gym"')],
51.                         Wall=[df_result['Wall'].to_html(index=Fals
    e).replace('<table', '<table class="display nowrap" id="Wall"')],
52.                         title='result')
53.         else:
54.             flash("არასწორი ფაილებია არჩეული")
55.             remove_files_from_updoad_folder(Filenames)
56.             return redirect(request.url)
57.
58.     return render_template('MsH/reportMsH.html', title='MsH')
```

ამ კოდით ხდება მომხმარებლისგან ლოგ-ფაილების წამოღება. მიღებული ფაილების „report_func_msh“ ფუნქციისთვის მიწოდება და მისგან დაბრუნებული შედეგების მომხმარებლისთვის ასახვა.

„MsH.py“ - ფაილში მოთავსებულია „report_func_msh“ ფუნქცია და მისთვის საჭირო დამხმარე ფუნქციები.

```
1.  from flask import current_app as app
2.  import pandas as pd
3.  import os, csv
4.  from datetime import datetime
5.  import numpy as np
6.  from bs4 import BeautifulSoup
7.
8.
9.  def read_excel_xml_sheet(path):
10.     try:
11.         file = open(path).read()
12.         soup = BeautifulSoup(file, 'xml')
13.         sheet_as_list =['none']
14.         for sheet in soup.findAll('Worksheet'):
15.             sheet_as_list = []
16.             for row in sheet.findAll('Row'):
17.                 row_as_list = []
18.                 for cell in row.findAll('Cell'):
19.                     try:
20.                         row_as_list.append(cell.Data.text)
21.                     except:
22.                         row_as_list.append(None)
23.                 sheet_as_list.append(row_as_list)
24.     except Exception as E:
25.         print(E)
26.         sheet_as_list = ['None']
27.     return sheet_as_list
28.
29.
30. def df_set_columns_names(df, conditionals):
31.     for index, row in df.iterrows():
```

```
32.         if conditionals in df.iloc[index].tolist():
33.             df.columns = df.iloc[index].tolist()
34.             break
35.     return df
36.
37.
38. def df_drop_rows_if_str_by_column(df, df_column):
39.     df = df[pd.to_numeric(df[df_column], errors='coerce').notnull
    ()]
40.     return df
41.
42.
43. def df_statistics_one_month(dfs):
44.     statistic_list = []
45.     statistic_table_list = []
46.     for key in dfs:
47.         df = dfs[key]
48.         statistic_list.append(key)
49.         statistic_list.append(df_msh_sum(dfs[key]))
50.         statistic_list.append(df['Contract'].count())
51.         statistic_list.append(datetime.now().strftime("%Y-%B"))
52.         statistic_table_list.append(statistic_list)
53.         statistic_list =[]
54.     return statistic_table_list
55.
56.
57. def df_msh_one_month_sum(df):
58.     df = df[df['paid up to'].dt.month == datetime.now().month]
59.     df_sum = pd.to_numeric(df['Amount']).sum()
60.     return df_sum
61.
```

```
62.
63. def df_msh_sum(df):
64.     df = df[df['paid up to'].dt.month == datetime.now().month]
65.     df_sum = pd.to_numeric(df['Amount']).sum()
66.     return df_sum
67.
68.
69. def list_2_data_frame(xml_2_list, MsHDate):
70.     df = pd.DataFrame(xml_2_list)
71.     df = df_set_columns_names(df, 'Contract')
72.     df = df_drop_rows_if_str_by_column(df, 'Contract')
73.     df = df_drop_rows_if_str_by_column(df, 'No.')
74.     df = df.dropna(axis='columns', how='all')
75.     df.insert(df.columns.get_loc('Validity'), 'Start', df['Validity'].st
    r.split(' - ').str.get(0))
76.     df.insert(df.columns.get_loc('Validity'), 'End', df['Validity'].str
    .split(' - ').str.get(1))
77.     df['Amount'] = (df['Amount'].str.replace(',', ''))
78.     df = df.drop(columns=['Validity'])
79.     df['paid up to'] = pd.to_datetime(df['paid up to'], format='%d.
    %m.%Y')
80.     df['Start'] = pd.to_datetime(df['Start'], format='%d.%m.%Y')
81.     df['End'] = pd.to_datetime(df['End'], format='%d.%m.%Y')
82.     df = df[df['paid up to'] >= MsHDate]
83.     return df
84.
85. def report_func_msh(file_names, MsHDate):
86.     xml_2_list = []
87.     for x in file_names:
88.         file_name = os.path.join(app.config['UPLOAD_FOLDER'],
    x)
```

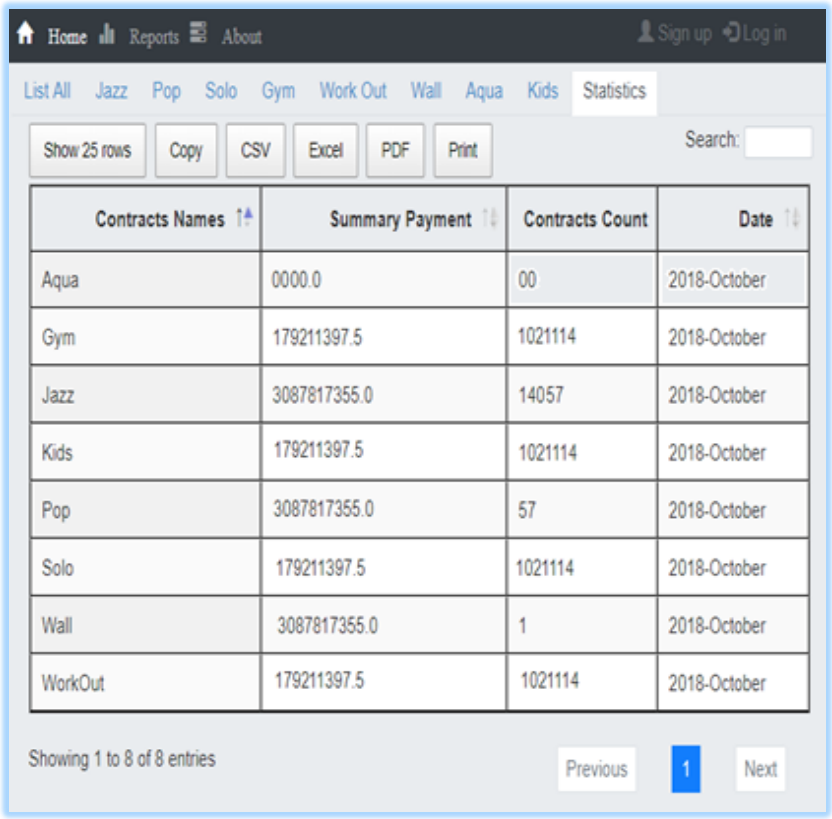
```
89.         xml_2_list = read_excel_xml_sheet(file_name)
90.         if 'Contract Data - Members contracts' in xml_2_list[0]:
91.             break
92.
93.     if 'Contract Data - Members contracts' not in xml_2_list[0]:
94.         return None
95.
96.     list_2_df = list_2_data_frame(xml_2_list, MsHDate)
97.
98.     #  kids
99.     df_kids = list_2_df[list_2_df['Position'].str.contains('Kid', case
    =False)]
100.    #  Aqua
101.    df_aqua = list_2_df[list_2_df['Position'].str.contains('Aqua', ca
    se=False)]
102.    #  WorkOut
103.    df_workOut = list_2_df[list_2_df['Position'].str.contains('Wor
    k', case=False)]
104.    #  Jazz
105.    df_jazz = list_2_df[list_2_df['Position'].str.contains('JAZZ', cas
    e=False)]
106.    #  Pop
107.    df_pop = list_2_df[list_2_df['Position'].str.contains('POP', case
    =False)]
108.    #  Solo
109.    df_solo = list_2_df[list_2_df['Position'].str.contains('SOLO', ca
    se=False)]
110.    #  Wall
111.    df_wall = list_2_df[list_2_df['Position'].str.contains('Wall', cas
    e=False)]
112.    #  ListAll
```

```
    df_listall = list_2_df
    #  GYM
    df_Gym = list_2_df[list_2_df['Position'].str.contains('GYM', case=False)]

    #  Statistics
    dfs = {'Kids': df_kids, 'Aqua': df_aqua, 'WorkOut': df_workOut, 'Jazz': df_jazz, 'Pop': df_pop, 'Solo': df_solo,
           'Wall': df_wall, 'Gym': df_Gym}
    df_statistics = pd.DataFrame(df_statistics_one_month(dfs), columns=['Contracts Names', 'Summary Payment', 'Contracts Count', 'Date'])


    return {'ListAll': df_listall,
            'Kids': df_kids,
            'Aqua': df_aqua,
            'WorkOut': df_workOut,
            'Jazz': df_jazz,
            'Pop': df_pop,
            'Solo': df_solo,
            'Wall': df_wall,
            'Statistics': df_statistics,
            'Gym': df_Gym}
```

პროგრამული კოდის შესრულების შედეგად მივიღეთ პროგრამული პროდუქტი, რომელიც მომხმარებელს აწვდის სტატისტიკას (ნახ.4.17).



ნახ.4.17. შედეგი

როგორც ნახაზზე არის ნაჩვენები, ლოგებიდან მიღებული ინფორმაცია შეგვიძლია გადავიტანოთ სხვადასხვა ფორმატში და გავაგრძელოთ მისი შემდგომი დამუშვება.

ამგვარად, მონაცემების შენახვის ნაწილში შემუშავებულ იქნა Linux ოპერაციული სისტემის მახასიათებლები: შეირჩა ამოცანისათვის უმჯობესი დისტრიბუცია, განისაზღვრა ვირტუალური მანქანის პარამეტრები, სისტემური ფაილებისა და მონაცემების შესანახი ადგილის ტიპი და ფორმატი.

ტესტირებისა და კრიტიკული ანალიზის საფუძველზე გამოვლენილ და განსაზღვრულ იქნა ოპერაციული სისტემისთვის მინიმალური მოთხოვნები: თუ რა პარამეტრებია აუცილებელი, რომელი პროგრამული მოდულები უნდა იყოს გამართული ოპერაციულ სისტემაზე, ამავე თავში განისაზღვრა ქსელური ბარათისა და „Firewall“ უსაფრთხოების მექანიზმის პარამეტრები.

შემდეგ დაპროექტებულ იქნა მონაცემთა ბაზის ინფრასტრუქტურა: განისაზღვრა მონაცემთა ბაზის პარამეტრები, სერვერების ტიპები დანიშნულების მიხედვით, შემუშავებულ იქნა მონაცემთა ბაზის ეკოსისტემაში მონაწილე სერვისების პარამეტრები და განისაზღვრა სერვისების მდებარეობა ფუნქციონალის მიხედვით (რომელი სერვისი რემელ სერვერზე უნდა იყოს გაშვებული).

წიგნში დეტალურად არის აღწერილი მონაცემთა შენახვა-დამუშავების ეკოსისტემის პროექტირება და გაშვება, რაც გულისხმობს მონაცემთა ბაზისა და ღრუბლოვანი ინფრასტრუქტურის გამართვასა და მონაცემთა შენახვა დამუშვების განაწილებული ეკოსისტემის მიღებას.

აღსანიშნავია, რომ ძვირადღირებული ალტერნატივებისგან განსხვავებით, წარმოდგენილი ნაშრომის ფარგლებში შექმნილი სისტემა შემუშავებულია დაბალი ღირებულების ტექნოლოგიების გამოყენებით.

მონაცემების შენახვა-დამუშვების ეკოსისტემის გამართვის შემდეგ შეიქმნა პროგრამული უზრუნველყოფა, რომელიც გაშვებულია პროექტის ფარგლებში შემუშავებულ ღრუბლოვან ინრასტრუქტურაში. პროგრამული უზრუნველყოფა კითხულობს მონაცემებს ბაზიდან, ამუშავებს წინასწარ განსაზღვრული მოთხოვნების მიხედვით და დამუშავებულ ინფორმაციას ასახავს რეპორტის სახით.

V თავი. გარე ღრუბლოვანი სერვისები

5.1 ზოგადი მიმოხილვა

გარე ღრუბლოვანი სერვისები უმთავრეს ღირსებას, როგორც აღინიშნა, დაბალი ინფრასტრუქტურული და ადმინისტრირების ხარჯები, ხოლო ნაკლოვანებას კორპორაციული ინფორმაციის დამოუკიდებლობლობის ხარისხის შემცირება წარმოადგენს.

ბოლო წლებში გარე ღრუბლოვანი სერვისების გამოყენების ინტენსივობამ მკვეთრად იმატა, განსაკუთრებით საოფისე და მონაცემთა ბაზების სფეროში. პრინციპი „ვანაზღაურებთ არა პროგრამული უზრუნველყოფის შეძენის, არამედ მისი გამოყენების („ქირაობის“) ღირებულებას“ ძალზე მიმზიდველია იმ შემთხვევაში, როცა პროგრამულ უზრუნველყოფაზე მოთხოვნა დროებითი ხასიათისაა [64].

ღრუბლოვანი სერვისების რამდენიმე მნიშვნელოვანი პლატფორმაა:

• Amazon Web Services • Kamatera • Microsoft Azure • Google Cloud Platform • VMWare • IBM Cloud • Rackspace • Red Hat • Salesforce	• Oracle Cloud • SAP • Verizon Cloud • Navisite • Dropbox, Egnyte • Adobe LiveCycle • iCloud • Fabasoft

უმრავლესი მათგანი უკვე ეტაბლირებული სოფტვეარ-გიგანტის შექმნილი პროდუქტია, თუმცა ზოგიერთი წარმატებულ სტარტაპსაც წარმოადგენს.

ქვემოთ ყველაზე პოპულარული (Top-3) ღრუბლოვანი სერვისების მიმოხილვას შევასრულებთ.

5.2. Amazon-ის ვებ-სერვისები

Amazon Web Services (AWS) არის მსოფლიოში ყველაზე ფართოდ გავრცელებული ღრუბლოვანი ტექნოლოგიების მწარმოებელი კომპანია, რომელიც ბევრ უნიკალურ პროდუქტს გვთავაზობს [65]. სერვისს გააჩნია მსოფლიოს მასშტაბით თანაბრად განაწილებული მონაცემთა ბაზები, რომლებზე ღრუბლოვანი ტექნოლოგიების პროდუქტები მუშაობს.

AWS მომხმარებელს სთავაზობს ისეთ ღრუბლოვან პროდუქტებს, როგორიცაა:

- Compute – გამოთვლითი სიმძლავრეები;
- Storage – გარე მეხსიერება;
- Databases – მონაცემთა ბაზები;
- Analytics – ანალიტიკური სერვისები;
- Networking – ქსელები;
- Mobile – მობილური სერვისები;
- Developer tools – სერვისები პროგრამული უზრუნველყოფის დამმუშავებლებისთვის;
- Management tools – სერვისები მენეჯერთათვის;
- IoT – „საგანთა ინტერნეტის“ (Internet of Things) სერვისები;
- Security - უსაფრთხოების სერვისები;
- Enterprise applications - წარმოების მართვის აპლიკაციები.

განვიხილოთ ზოგიერთი სერვისი უფრო დეტალურად:

Amazon EC2 (Amazon Elastic Compute Cloud) PaaS-სერვისია, რომელიც უზრუნველყოფს უსაფრთხო, ცვლადი მოცულობის გამოთვლებს ღრუბელში. იგი განკუთვნილი დეველოპერებისთვის, რათა მათ მარტივად შეძლონ ვებ-სერვისის განვითარება მისი მოცულობის ზრდასთან თუ შემცირებასთან ერთად. Amazon EC2 სერვერის ინსტალაცია კომპიუტერში ფიზიკური სერვერის გამართვასთან შედარებით ძალიან სწრაფად ხორციელდება და შემდეგი დადებითი თვისებებით გამოირჩევა: მოქნილობა,

მართვადობა, ინტეგრირებულობა, საიმედობა, უსაფრთხოება, სიიაფე და სიმარტივე.

Amazon S3 (Amazon Simple Storage Service) წარმოადგენს მუდმივი მეხსიერების მიწოდების სერვისს [66,67]. იგი სთავაზობს მომხმარებელს მოქნილობას, მონაცემთა ხელმისაწვდომობას და უსაფრთხოებას. სერვისის ღირსებათაგან შეიძლება გამოვყოთ:

- სასურველი მოცულობის არჩევის მოქნილი სქემა;
- დაშვების დონეების მართვა;
- უსაფრთხოების უზრუნველყოფა და მონაცემებთან წვდომის აუდიტის განხორციელების შესაძლებლობა.

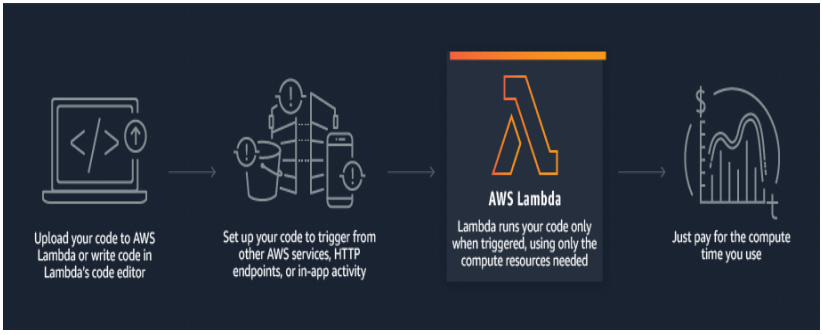
Amazon Aurora – MySQL და PostgreSQL სისტემებთან თავსებადი მონაცემთა ბაზების მართვის სისტემაა, რომელიც აერთიანებს ტრადიციულ მონაცემთა ბაზებზე მუშაობის შესაძლებლობას და მარტივად და ხარჯ-ეფექტურად ღია მონაცემებზე ხელმისაწვდომობას [68]. მწარმოებლის ინფორმაციით, Amazon Aurora არის ხუთჯერ უფრო სწრაფი, ვიდრე სტანდარტული MySQL და სამჯერ უფრო სწრაფი ვიდრე სტანდარტული PostgreSQL მონაცემთა ბაზები. იგი უზრუნველყოფს მონაცემთა ბაზების უსაფრთხოებას, ხელმისაწვდომობას და საიმედობას ძალიან დაბალ ფასად. Amazon Aurora სრულიად იმართება Amazon RDS (Relational Database Service) -ის მიერ, რომელიც ავტომატურად მართავს შრომატევად ადმინისტრაციულ ამოცანებს, როგორიცაა:

- აპარატურების უზრუნველყოფის მართვა;
- მონაცემთა ბაზების ინსტალაცია;
- სარეზერვო ასლების მართვა Amazon S3-სთან „თანამშრომლობით“;
- ბაზის ავტომატური მასშტაბირება;
- თვითაღდგენის უნარი და დაზიანებების მიმართ ნაკლებად მგრძნობიაროба;
- ბაზის დროის ნებისმიერი მომენტის მდგომარეობით აღდგენის უნარი (point-in-time recovery);

- რეპლიკაცია ხელმისაწვომობის ზონების მიხედვით (Availability Zones (AZs);
- ბაზის მაქსიმალური მოცულობა 64 ტერაბაიტამდე;
- მიგრაციის მხარდაჭერა.

Amazon DynamoDB (განსხვავებით Aurora-სგან) არის არარელაციური, ძალზე მაღალი წარმადობის მონაცემთა ბაზების მართვის სისტემას (ე.წ. დოკუმენტების მონაცემთა ბაზა), რომელიც ნებისმიერი მოცულობის ინფორმაციულ მოთხოვნას წამის მეასედში, მილიწამებში ამუშავებს [69]. DynamoDB-ს შეუძლია წამში 20 მილიონზე და ყოველდღიურად 10 ტრილიონზე მეტ მოთხოვნას გაუმკლავდეს.

AWS Lambda გვაძლევს საშუალებას შევქმნათ, ვაწარმოოთ და ვამუშაოთ მოვლენებზე ორიენტირებული აპლიკაციები (ფუნქციები) სერვერული უზრუნველყოფის გარეშე [70,71]. მხოლოდ კოდის ატვირთვა საკმარისია, რომ ჩვენი აპლიკაციები ხელმისაწვდომი გახდეს. სადღეისოდ სერვისს დაპროგრამების შემდეგი ენების მხარდაჭერა გააჩნია: Node.js, Python, Java, Go და C# over .NET Core.



ნახ.5.1. AWS LAMBDA-ს მოქმედების სქემა

Amazon VPC (Virtual Private Cloud) გვაძლევს საშუალებას შევქმნათ იზოლირებული ღრუბლოვანი სივრცე, სადაც შესაძლებელია ამაზონის სერვისების ვირტუალურ ქსელში გაშვება [72].

მომხმარებელი მთლიანად აკონტროლებს ვირტუალური ქსელის გარემოს. მას შეუძლია:

- IP-მისამართების ბლოკების (პულების) შექმნა;
- ქვექსელების გენერირება;
- რუთინგის ცხრილების და ქსელის „ჭიშკრების“ მართვა;
- უსაფრთხოების ქსელური სეგმენტის აგება და მართვა.

მთელი ეს ფუნქციონალი ხელმისაწვდომია როგორც IPv4 ასევე IPv6-გარემოში [73].

AWS Lake Formation არის სერვისი [74]. იგი გვაძლევს ე.წ. „მონაცემთა ტბების“ (Data Lakes) შექმნის შესაძლებლობას ცენტრალიზებული, მონაცემთა დაცული საცავის სახით მათი შემდგომი უსაფრთხოების უზრუნველყოფისა და ანალიზისთვის (ნახ.5.2).

ნახ.5.2. AWS LAKE FORMATION-ის ადგილი და როლი ღრუბლოვან სივრცეში

5.3 Microsoft Azure და მისი ვირტუალური მანქანები

Microsoft Azure არის ღრუბლოვანი სერვისების კომპლექტი, რომელიც მომხმარებელს სთავაზობს მონაცემთა ბაზების მართვას, ფაილების შესანახ სივრცეს, სარეზერვო ასლების განთავსებას, სერვისებს მობილური თუ ვებ აპლიკაციებისთვის და ასობით სხვა მომსახურებას. Microsoft Azure ფუნქციები ხელმისაწვდომია კლიენტისათვის როგორც სერვისის (SaaS), ასევე პლატფორმის (PaaS) ან ინფრასტრუქტურის (IaaS) სახით [7,75].

Windows Azure სისტემის მუშაობის პრინციპი ეყრდნობა *ვირტუალური ძრავის* გაშვებას ცალკეული აპლიკაციისათვის, სადაც წინასწარ დგინდება გამოთვლითი და მეხსიერების რესურსების აუცილებელი მოცულობა, რომლებსაც შემდგომ პლატფორმა ავტომატურად გამოყოფს. ასევე ავტომატურად გამოიყოფა რესურსები თავდაპირველი მოთხოვნის ცვლილებისას.

Azure-ს ინფრასტრუქტურის მართვა ხორციელდება სერვისის მომწოდებლის მიერ, მომხმარებელი კი მართავს მხოლოდ ოპერაციულ სისტემას და აპლიკაციებს.

გამოვყოთ Azure-ს რამდენიმე მნიშვნელოვანი რესურსი მათი დეტალური აღწერის გარეშე:

- ვირტუალური მანქანები და აპლიკაციები:
  - Azure Virtual Machines (Linux, Windows);
  - App Services (Web Apps, Mobile Apps, Logic Apps, API Apps, and Function Apps;
  - Azure Batch (for large-scale parallel and batch compute jobs);
  - Azure RemoteApp;
  - Azure Service Fabric;
  - Azure Container Service.
- მონაცემთა მართვის სერვისები:

- Azure Storage (comprises the Azure Blob, Queue, Table, and File services);
- Azure SQL Database;
- Azure DocumentDB;
- Microsoft Azure StorSimple;
- Azure Redis Cache.

➢ სისტემური სერვისები:
- Azure Active Directory (Azure AD);
- Azure Service Bus for connecting distributed systems;
- Azure HDInsight for processing big data;
- Azure Scheduler;
- Azure Media Services.

➢ ქსელის სერვისები:
- Azure Virtual Network;
- Azure ExpressRoute;
- Azure-provided DNS;
- Azure Traffic Manager;
- Azure Content Delivery Network.

შევეხოთ *Azure-ს ვირტუალური მანქანებს,* რომლებიც ღრუბლოვან სივრცეში, ალბათ, ყველაზე პოპულარულ სერვისს წარმოადგენს. სერვისი მომხმარებელს სთავაზობს Windows ან Linux ვირტუალური მანქანების გამოყენებას Microsoft Azure-ს მონაცემთა „ღრუბლიდან“.

მომხმარებელს შეუძლია შექმნას ნებისმიერი ზომის და პროცესორული, ოპერატიული და გარე მეხსიერების მქონე ვირტუალური მანქანა. ქვემოთ მოცემულია ვირტუალური მანქანების საკონფიგურაციო სქემები Windows და Linux-ოპერაციულ სისტემებზე მომუშავე ვირტუალური მანქანებისთვის (ცხრ.5.1, 5.2).

Windows- ბაზირებული ვირტუალური მანქანების საკონფიგურაციო ვარიანტები ცხრ.5.1

Type	Sizes	Description
General purpose	B, Dsv3, Dv3, DSv2, Dv2, Av2, DC	Balanced CPU-to-memory ratio. Ideal for testing and development, small to medium databases, and low to medium traffic web servers.
Compute optimized	Fsv2, Fs, F	High CPU-to-memory ratio. Good for medium traffic web servers, network appliances, batch processes, and application servers.
Memory optimized	Esv3, Ev3, M, GS, G, DSv2, Dv2	High memory-to-CPU ratio. Great for relational database servers, medium to large caches, and in-memory analytics.
Storage optimized	Lsv2, Ls	High disk throughput and IO ideal for Big Data, SQL, NoSQL databases, data warehousing and large transactional databases.
GPU	NV, NVv2, NC, NCv2, NCv3, ND, NDv2 (Preview)	Specialized virtual machines targeted for heavy graphic rendering and video editing, as well as model training and inferencing (ND) with deep learning. Available with single or multiple GPUs.
High performance compute	H	Our fastest and most powerful CPU virtual machines with optional high-throughput network interfaces (RDMA).

Linux - ბაზირებული ვირტუალური მანქანების საკონფიგურაციო ვარიანტები ცხრ.5.2

Type	Sizes	Description
General purpose	B, Dsv3, Dv3, DSv2, Dv2, Av2, DC	Balanced CPU-to-memory ratio. Ideal for testing and development, small to medium databases, and low to medium traffic web servers.
Compute optimized	Fsv2, Fs, F	High CPU-to-memory ratio. Good for medium traffic web servers, network appliances, batch processes, and application servers.
Memory optimized	Esv3, Ev3, M, GS, G, DSv2, Dv2	High memory-to-CPU ratio. Great for relational database servers, medium to large caches, and in-memory analytics.
Storage optimized	Lsv2, Ls	High disk throughput and IO ideal for Big Data, SQL, NoSQL databases, data warehousing and large transactional databases.
GPU	NV, NVv2, NC, NCv2, NCv3, ND, NDv2 (Preview)	Specialized virtual machines targeted for heavy graphic rendering and video editing, as well as model training and inferencing (ND) with deep learning. Available with single or multiple GPUs.
High performance compute	H	Our fastest and most powerful CPU virtual machines with optional high-throughput network interfaces (RDMA).

მნიშვნელოვანია აღნიშნულ ვირტუალურ მანქანებზე წვდომის კონტროლის და, ზოგადად, უსაფრთხოების საკითხები, რადგან ღრუბლოვანი სერვისების ხელმისაწვდომობა, პრინციპში, მთელი ინტერნეტ-სივრციდან უნდა იყოს უზრუნველყოფილი.

NSG (Network Security Group) ტექნოლოგია უზრუნველყოფს აღნიშნული ამოცანის შესრულებას, შესაბამისი ქსელური ინტერფეისის და პორტების გახსნის გზით [76].

აღსანიშნავია ასევე *დატვირთვათა განაწილების* (Load Balancing) ფუნქცია, რომელიც Azure-ს ვირტუალურ მანქანებიდან სრულფასოვანი სერვერული ინფრასტრუქტურის (ფერმის) აგების საშუალებას გვაძლევს [77].

ღრუბლოვანი სერვისების გამოყენების ერთ-ერთი მთავარი წინაპირობა *სერვისის გამოწერაა,* რომელიც სხვადასხვა მომწოდებელთან ნაწილობრივ თანხვედრილი, მაგრამ ნაწილობრივ განსხვავებული ბიზნეს-მოდელებით ხორციელდება. განვიხილოთ Azure-ში სერვისის გამოწერის მოდელი, რომლის რამდენიმე ვარიანტი არსებობს:

- **უფასო ანგარიშები –** რეგისტრაციისას მომხმარებელს ეძლევა 200 დოლარის დეპოზიტი 30 დღის განმავლობაში Azure-ს სერვისების გამოსაცდელად, ხოლო კრედიტის ამოწურვის შემდეგ სერვისი ჩერდება. მისი შემდგომი გამოყენება მხოლოდ ფასიან პაკეტზე გადასვლის შემდეგ არის შესაძლებელი;
- **MSDN გამომწერები –** Microsoft კომპანიის დეველოპერები Azure-სერვისების გამოწერისას სპეციალური ტარიფებით სარგებლობენ. მაგალითად, Microsoft Visual Studio Enterprise-ის გამოწერისას ავტომატურად იღებთ საჩუქრად 150 დოლარის დეპოზიტს Azure-პლატფორმაზე. ლიმიტის ამოწურვის შემდეგ სერვისების მიწოდება მომდევნო თვემდე ჩერდება, მაგრამ მომხმარებელს შეუძლია თავისი საკრედიტო ბარათი მიაბას პლატფორმას და არ შეწყვიტოს სერვისებით სარგებლობა. ასეთ შემთხვევაშიც MSDN-გამომწერები სპეციალური ფასდაკლებებით სარგებლობენ;

- **BizSpark – ანგარიშები** სტარტაპებისათვის უზრუნველყოფს ფასდაკლებებს. ყველაზე დიდი უპირატესობაა ის, რომ ანგარიშების მფლობელებს წვდომა აქვთ მაიკროსოფთის ყველა სერვისზე სატესტოდ და დეველოპმენტისთვის. გარდა ამისა, ისინი სარგებლობენ ხუთი MSDN-ანგარიშით, რომლებზეც ყოველთვიურად იღებენ $150 დეპოზიტს და დამატებით ფასდაკლებებს;
- **Pay-as-you-go –** ამ ტიპის გამომწერები პლატფორმას უკავშირებენ საკრედიტო ბარათს და გამოყენებული სერვისის ექვივალენტურ თანხას იხდიან;
- **კორპორატიული პაკეტებით** ორგანიზაციები სარგებლობენ. ისინი ათანხმებს სასურველ სერვისებს და იხდის 1 წლის მომსახურების საფასურს. ლიმიტის დროზე ადრე ამოწურვის შემთხვევაში ხდება მისი გაზრდა და ამ შემთხვევაში ორგანიზაცია ფასდაკლებებით სარგებლობს.

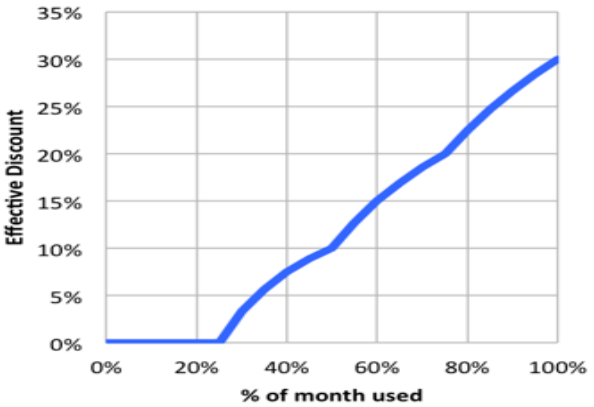
5.4 . Google Cloud-ია სერვისი – ვირტუალური სერვერი

Google Cloud არის ერთ-ერთი უმსხვილესი ღრუბლოვანი სერვისების მწარმოებელი კომპანია მსოფლიოში, რომელიც მრავალ სერვისს სთავაზობს მომხმარებელს და ბაზარზე წარმოადგენს ერთ-ერთ უმსხვილეს მომწოდებელს (ვენდორს). შევეხოთ მის ორ ძირითად სერვისს: *ვირტუალურ სერვერს* (compute engine) და *ფუნქციებს* (cloud functions) [78,79].

ვირტალური სერვერი „ინფრასტრუქტურა, როგორც სერვისი – IAAS“ ტიპის სერვისია, რომელიც მაღალი ხარისხის, მასშტაბირებად ვირტუალურ მოწყობილობებს (high performance – scalable VMs) გვთავაზობს [78]. ისინი გუგლის ინოვაციურ მონაცემთა საცავებში ინახება და მუშაობს. ორიგინალური ტექნიკური გადაწყვეტის გამო სერვერის ვირტუალური მოწყობილობები სწრაფი ჩატვირთვით გამოირჩევა. აღნიშნულს ხელს უწყობს ასევე SSD (solid state drive)-დისკების მასობრივი გამოყენება (რომელიც ტრადიციულ HDD-დისკებზე ბევრად სწრაფი, მაგრამ შედარებით ძვირია) [80,81].

Google-ს ვირტუალური სერვერები მისაწვდომია მრავალფეროვანი კონფიგურაციებით, რომლებიც ოპტიმიზებული და მორგებულია მომხმარებლის კონკრეტულ მოთხოვნებზე. მოხერხებული ფასწარმოქმნა და ფასდაკლებები ასევე კომპანია გუგლის ბაზარზე დომინირებას განაპირობებს.

გუგლი იყენებს მეორე დონის გადახდის სისტემებს (pay per use) - ანუ იხდით იმდენს, რამდენისაც გამოიყენეთ [82]. გუგლის ფასდაკლების სისტემა განსაკუთრებით ეფექტურია რესურსების ხანგრძლივი გამოყენების შემთხვევაში (ნახ.5.3).



ნახ.5.3. Google-ს ღრუბლოვანი სერვისების ეფექტური ფასდაკლების ნიმუში

მნიშვნელოვანი კომპიუტერული კლასტერების შექმნის საშუალება, რომელიც მდგრადი, ფიზიკურად განაწილებული კომპიუტერული ინფრასტრუქტურის შექმნას განაპირობებს. ამგვარი სტრუქტურის მდგრადობას ასევე მკვეთრად ამაღლებს გუგლის *კერძო საერთაშორისო ოპტიკურბოჭკოვანი ქსელი* (Google's private global fiber network), რომელიც კომპანიის

სხვადასხვა მონაცემთა ცენტრებს შორის სწრაფი და ეფექტური კავშირისთვის იდეალურ გარემოს ქმნის [83,84].

არ შეიძლება გამოვტოვოთ კომპანიის წვლილი „გარემოსთან მეგობრობის“ საქმეშიც. 21-ე საუკუნეში გარემოზე ზრუნვა პრიორიტეტულია, ამიტომ მოწინავე კომპანიები ცდილობს ე.წ. „მწვანე ეკონომიკის“ მოდელი დანერგოს. გუგლის შემთხვევაშიც ღრუბლოვანი ინფრასტრუქტურა გარემოს დაცვაზეა ორიენტირებული, მისი მონაცემთა ცენტრები მოიხმარს 50%-ით ნაკლებ ენერგიას, ვიდრე სხვა სტანდარტული მონაცემთა ცენტრები. გუგლი იძენს დიდი ოდენობით განახლებად ენერგიას თავისი მონაცემთა ცენტრებისთვის და ამითაც დიდი წვლილი შეაქვს გარემოს დაცვის საქმეში [85].

გამოყენებული ლიტერატურა:

1. სურგულაძე გ., წაწიშვილი დ. ვირტუალური რეალობა და თანამედროვე საინფორმაციო ტექნოლოგიები. დამხ.სახ., ISBN 978-9941-8-0626-1. სტუ. „IT-კონსალტ. სამეცნ.ცენტრი“. თბ., -112 გვ.

2. Virtualization . https://en.wikipedia.org/wiki/Virtualization

3. სურგულაძე გ., გულუა დ., თ. დოლიძე, ე. თურქია. ვირტუალური სისტემების მოდელირება კორპორაციულ ქსელებში.*სტუ-ს შრ.კრ. „მართვის ავტომატიზებული სისტემები“*. 2009, N2(7), გვ. 67-70

4. სურგულაძე გ., გულუა დ., მაისურაძე გ. კორპორატიული ქსელების „ღრუბლოვანი" სერვისების მონაცემთა საცავების დაპროექტების მეთოდები. *სტუ-ს შრ.კრ. „მართვის ავტომატი-ზებული სისტემები“*. 2011, N2(11), გვ. 89-92.

5. https://www.etymonline.com/word/virtual

6. https://ru.wikipedia.org/wiki/Виртуальность

7. სურგულაძე გ., გულუა დ. ქსელური არქიტექტურები ბიზნესისათვის. დამხ.სახ., ISBN 978-9941-0-9842-0. სტუ. „IT-კონსალტ. სამეცნ.ცენტრი“. თბ., -269 გვ.

8. სურგულაძე გ., კახელი ბ., მაისურაძე გ. მონაცემთა შენახვა-დამუშვების განაწილებული სისტემები. *სტუ-ს შრ.კრ. „მართვის ავტომატიზებული სისტემები“*. 2017, N1(23), თბ., გვ.79-83.

9. Surguladze G., Kiviladze G., Kakheli B. Formation of Big Data Research Center. *Transactions GTU Automated Control Systems*. 2018, N2(26). Tbilisi, pp.189–192.

10. Kiviladze G., Saralidze T,, Kiviladze T., Kakheli B. Emergency Medical Service: Difficulties and Improvements. *Transactions GTU Automated Control Systems*. 2018, N2(26). Tbilisi, pp.228–231.

11. Dern D.P. How Virtualization Improves Software Development. 2009. https://www.cio.com/article/2430777/how-virtualization-improves-soft-ware-development.html

12. Bobko PK. Linux and general public licenses: can copyright keep Open Source software Free. *AIPLA QJ*. 2000, 28, 81.

13. https://www.reddit.com/r/linuxmasterrace/comments/7diwwi/linux_distro_timeline, გადამოწმ. - 01.12.2018.

14. Lu L, Arpaci-Dusseau AC, Arpaci-Dusseau RH, et al. A study of Linux file system evolution. *ACM Transactions on Storage (TOS)*. 2014, 1, 10, 3.

15. Aghayev A, Theodore Y, Gibson G, et al. Evolving Ext4 for Shingled Disks. InFAST. 2017, 105-120.

16. https://www.cyberciti.biz/tips/the-importance-of-linux-partitions.html, გადამოწმ. - 01.12.2018.

17. Sweeney A, Doucette D, Hu W, et al. Scalability in the XFS File System. InUSENIX Annual Technical Conference. 1996, 15.

18. Kay T. Linux swap space. *Linux Journal*. 2011, 201, 5.

19. Blansit BD. Readily Available Methods for Testing Linux. *Journal of Electronic Resources in Medical Libraries*. 2009, 2, 6, 174-83.

20. Negus C. Red Hat Linux 9 Bible. *John Wiley & Sons, Inc*. 2003.

21. Yurcik W, Woolam C, Hellings G, et al. Scrub-tcpdump: A multi-level packet anonymizer demonstrating privacy/analysis tradeoffs. InSecurity and Privacy in Communications Networks and the Workshops. Third International Conference on 2007, 49-56.

22. Newham C, Rosenblatt B. Learning the bash shell: Unix shell programming. O'Reilly Media, Inc., 2005.

23. Yang G, Chen S.Y. Research on Linux firewall based on Netfilter/Iptables [J]. *Computer Engineering and Design*. 2007,17,022.

24. Dubrawsky I. Firewall evolution-deep packet inspection. *Security Focus*. 2003, 29.

25. Capone JM, Immaneni P, inventors. Protocol and system for firewall and NAT traversal for TCP connections. United States patent US 7,646,775. Filed: 12.01.2010.

26. Bhargava B, Riedl J. The Raid distributed database system. *IEEE Transactions on Software Engineering*. 1989. 1, 6, 726-36.

27. Elle K. RAID Explained. Linux Academy Blog. 2016 https://linuxacademy.com/blog/linux/raid-explained,

28. Hitz D, Malcolm M, Lau J, et al. Method for allocating files in a file system integrated with a RAID disk sub-system. United States patent US 6,038,570. Filed: 14.3.2000.

29. Corbett P, English B, Goel A, et al. Row-diagonal parity for double disk failure correction. InProceedings of the 3rd USENIX Conference on File and Storage Technologies.2004, 1-14.

30. Choy M, Leong HV, Wong MH. Disaster recovery techniques for database systems. Communications of the ACM. 2000, 1, 43, 6.

31. RAID Types and Levels. http://www.raid-calculator.com/raid-types-reference.aspx, გადამოწმ. - 10.06.2019

32. ჩოგოვაძე გ., ფრანგიშვილი ა., კივილაძე გ., სურგულაძე გ., ნარეშელაშვილი გ. ინფორმაციული საზოგადოება, მონაცემთა მენეჯმენტის ახალი ტექნოლოგიები და ექსტრემალური სიტუაციების მართვის სისტემები. *სტუ-ს შრ.კრ. „მართვის ავტომატიზებული სისტემები“*. 2018, 1, 25, 7 - 16.

33. Kedia P, Nagpal R, Singh TP. A survey on virtualization service providers, security issues, tools and future trends. *International Journal of Computer Applications*. 2013, 4, 69.

34. Singer W. NAS and iSCSI technology overview. SNIA Technical Tutorials. 2007.

35. FreeNAS. https://www.freenas.org, გადამოწმ. - 10.06.2019.

36. https://arstechnica.com/information-technology/2014/06/the-ars-nas-distribution-shootout-freenas-vs-nas4free, გადამოწმ. - 10.06.2019.

37. https://www.openmediavault.org, გადამოწმ. - 01.12.2018.

38. Zou L, Rui X, Nguyen TA, et al. A Scalable Network Area Storage with Virtualization: Modelling and Evaluation using Stochastic

Reward Nets. *InProceedings of the 2018 International Conference on Information Science and System*, 2018, 225-233.

39. Castro J.D. Task-Oriented Distros. Introducing Linux Distros 2016, pp. 345-355.

40. Gibson GA, Van Meter R. Network attached storage architecture. Communications of the ACM. 2000, 11, 43, 37-45.

41. მარტიაშვილი გ., კახელი ბ., კაშიბაძე მ. ტელეფონია და SMS, *სტუ-ს შრ.კრ. „მართვის ავტომატიზებული სისტემები“*. 2017, 1, 23, 211-215.,

42. Knud Lasse-Lueth. https://iot-analytics.com/state-of-the-iot-update-q1-q2-2018-number-of-iot-devices-now-7b

43. Kiviladze G. Semi automated management of defining the case priority in the flu epidemic season for Emergency Medical Service. GESJ:Computer Sciences and Telecommunications. 2018, 1, 53, 101-105

44. სურგულაძე გ., პეტრიაშვილი ლ. მონაცემთა საცავის აგების ტექნოლოგია ინტერნეტული ბიზნესის სისტემებისათვის. ISBN 99940-40-36-7. სტუ. „ტექნიკური უნივერსიტეტი“, 2005, 200 გვ.

45. პეტრიაშვილი ლ., სურგულაძე გ. მონაცემთა მენეჯმენტის თანამედროვე ტექნოლოგიები (Oracle, MySQL, MongoDB,Hadoop). ISBN 978-9941-27-176-2. სტუ, „ITკონსალტინგის სამეცნ.ცენტრი", თბ., 2017, -202 გვ.

46. fabric8. integrated development platform. Documentation for version 2.2.101. http://fabric8.io/docs/index.html, გადამოწ. 21.05.2019.

47. https://www.slideshare.net/satyajitmenon/perspective-2020-nasscom, გადამოწმ. - 01.6.2019.

48. https://www.realvnc.com/en, გადამოწმ. - 10.05.2019.

49. https://www.teamviewer.com/en, გადამოწმ. - 10.05.2019.

50. Owens M. Embedding an SQL database with SQLite. *Linux Journal*. 2003, 2, 110.

51. Aditya SK, Karn VK. Android sQLite essentials. Packt Publishing Ltd. 2014.

52. მარტიაშვილი გ., კახელი ბ. დიდი ზომის მონაცემთა შენახვა მობილურ აპლიკაციებში. VIII საერთაშორისო სამეცნიერო და პრაქტიკული კონფერენციის "ინტერნეტი და საზოგადოება" (INSO2017) ნაშრომების კრებული. 2017, 170 - 173.

53. სურგულაძე გ., თურქია ე., ბულია ი. Web-აპლიკაციების დამუშავება მონაცემთა ბაზების საფუძველზე (ASP.NET, ADO.NET, C#).სტუ-ს „IT-კონსალტინგის ცენტრი". 2014, გვ. 150 – 161.

54. სურგულაძე გ., კივილაძე ნ., კივილაძე გ. კორპორაციული აპლიკაციების აგება დაპროგრამების სერვის-ორიენტირებული ტექნოლოგიით. *სტუ-ს შრ.კრ. „მართვის ავტომატიზებული სისტემები".* 2016, 1, 21, გვ. 215-220.

55. https://www.linuxcounter.net/statistics/kernel, გადამოწმ. - 05.06.2019.

56. http://fabric8.io/docs/index.html, გადამოწმ. - 11.04.2019.

57. https://www.openshift.com, გადამოწმ. - 11.04.2019.

58. Bernstein D. Containers and cloud: From lxc to docker to kubernetes. *IEEE Cloud Computing*. 2014, 1, 3, 81-4.

59. Brewer EA. Kubernetes and the path to cloud native. *InProceedings of the Sixth ACM Symposium on Cloud Computing*. 2015, 167-167).

60. კახელი ბ., მარტიაშვილი გ. რუსეთის ფედერაციის თავდაცვითი და შემტევი კიბერ შესაძლებლობები. სტუ-ს შრ.კრ. „მართვის ავტომატიზებული სისტემები". 2017, 1, 23, 216-219

61. სურგულაძე გ., კივილაძე გ., კახელი ბ. NoSQL მონაცემთა ბაზების განვითარების პერსპექტივები და პრობლემები მართვის საინფორმაციო *სისტემებში. სტუ-ს შრ.კრ. „მართვის ავტომატიზებული სისტემები".* 2016, N2(22), თბ., გვ.230-239

62. სურგულაძე გ., კივილაძე გ. შესავალი NoSQL მონაცემთა ბაზებში (MongoDB). ISBN 978-9941-0-9642-6. სტუ. „IT-კონსალტინგის ცენტრი", თბ., 2017, 151 გვ.

63. სურგულაძე გ., დოლიძე ს. მომხმარებლის ინტერფეისის დაპროგრამება (AngularJs, ReactJs). ISBN 978-9941-8-0625-4. სტუ. „IT-კონსალტინგის ცენტრი", თბ., 2019, 106 გვ.

64. Aishwarya Srinivasana, Abdul Quadir Mdb, Vijayakumar.Vc, Era of Cloud Computing: A New Insight To Hybrid Cloud, Science Direct, Procedia Computer Science 50 (2015) 42 – 51

65. About AWS . https://aws.amazon.com/about-aws/

66. Amazon Simple Storage Service. https://docs.aws.amazon-.com/AmazonS3/latest/dev/Welcome.html

67. Amazon Simple Storage Service. Developer Guide . 2006 https://docs.aws.amazon.com/AmazonS3/latest/dev/s3-dg.pdf

68. Amazon Aurora Features: MySQL-Compatible Edition. https://aws.amazon.com/rds/aurora/details/mysql-details/

69. Amazon DynamoDB. https://aws.amazon.com/dynamodb/

70. Build a Serverless Web Application with AWS Lambda, Amazon API Gateway, Amazon S3, Amazon DynamoDB, and Amazon Cognito. https://aws.amazon.com/getting-started/projects/build-server-less-web-app-lambda-apigateway-s3-dynamodb-cognito/module-3/

71. Using AWS Lambda with Amazon DynamoDB. https://docs.aws.amazon.com/lambda/latest/dg/with-ddb.html

72. Amazon Virtual Private Cloud User Guide. Copyright © 2019 Amazon Web Services, Inc. and/or its affiliates. https://docs.aws-.amazon.com/vpc/latest/userguide/vpc-ug.pdf

73. Vangie Beal. What is The Difference Between IPv6 and IPv4 ? Updated October 2017. https://www.webopedia.com/DidYouKnow-/Internet/ipv6_ipv4_difference.html

74. AWS Lake Formation features. https://aws.amazon.com/lake-formation/features/

75. John Adams, Azure for Developers, O'Reilly, 2015, 978-1-491-92612-3. [LSI]

76. Pagliai I. Azure Network Security Groups (NSG) – Best Practices and Lessons Learned. 2016. https://blogs.msdn.microsoft-.com/igorpag/2016/05/14/azure-network-security-groups-nsg-best-practices-and-lessons-learned/

77. Fidelis E. Azure Load Balancing Solutions: A guide to help you choose the correct option. 2018. https://devblogs.microsoft.com/premier-developer/azure-load-balancing-solutions-a-guide-to-help-you-choose-the-correct-option/

78. Google Compute Engine: Scalable, High-Performance Virtual Machines. https://cloud.google.com/compute/

79. Google Cloud Functions: Event-driven serverless compute platform. https://cloud.google.com/functions/

80. SSD vs HDD: Which should I have in my PC? https://www.win-dowscentral.com/ssd-vs-hdd-which-should-i-have-my-pc

81. SSD vs HDD. https://www. .storagereview.com/ssd_vs_hdd

82. Google Cloud: Filestore pricing. https://cloud.google.com-/filestore/pricing

83. Google's Global Network Infrastructure. Sydney, NSW, Australia, 2015. http://blog.zorangagic.com/2015/08/googles-global-network-infrastructure.html

84. Virtual Private Cloud (VPC). https://cloud.google.com/vpc/

85. Renewable energy. Google data centers. 2017. https://www.google.com/about/datacenters/renewable/index.html

86. Ibrahim M, Ramanathan S, Som TK, et al. System and method to support identity theft protection as part of a distributed service oriented ecosystem. United States patent US 9,357,384. Filed: 31.05.2016.

87. Wu M.W., Lin Y.D. Open source software development: an overview. *Computer*. 2001, 1, 6, 33-8.

88. Stallman R. Free software, free society: Selected essays of Richard M. Stallman. *Lulu Com*, 2002.

89. ბოტჰე კ., სურგულაძე გ., დოლიძე თ., შონია ო., სურგულაძე გიორგი. თანამედროვე პროგრამული პლატფორმები და ენები (WindowsNT, Unix, Linux, C++, Java, XML). სტუ „ტექნიკური უნივერსიტეტი". ISBN 99940-14-11-0. თბ., 230 გვ.

90. Knud Lasse-Lueth. The top 20 Industrial IoT trends – as showcased at Hannover Messe 2019. April 1-5, 2019 in Hannover, Germany. https://iot-analytics.com/author/knud-lasse-lueth/

გადაეცა წარმოებას 20.06.2019. ხელმოწერილია დასაბეჭდად 01.07.2019. ოფსეტური ქაღალდის ზომა 60X84 1/16. პირობითი ნაბეჭდი თაბახი 6. ტირაჟი 100 ეგზ.

სტუ-ს „IT კონსალტინგის სამეცნიერო ცენტრი"
(თბილისი, მ. კოსტავას 77)

ISBN 978-9941-8-0627-8