

Admission

Students of IT faculties often have to become familiar with the configuration and management of Windows and GNU / Linux operating systems during their studies, which is related to the need to provide these people with access to operating systems from the administrator account level. For obvious reasons, physical machines, which are standard equipment of a computer lab, are not able to provide users with such support, because it would require refreshing their image after each class, which is associated with too much work.

One way to solve this problem is to use virtualization technology that allows you to create a standalone virtual machine inside a physical computer. On such a prepared machine, it is possible to install the selected operating system and access from the administrator level.

As a result, a kind of "system in the system" is created, where the physical machine on which we work is the *host* , and the *virtual systems are its guests* . Virtual machines run in an isolated environment, which allows you to safely test various types of software and the systems themselves, without harming the host machine. Moreover, they can also be easily created, duplicated, deleted, or managed without logging into the computer they are running on. Such a tool solution makes it possible to create in a relatively simple way entire virtual laboratories, which are now commonly used in education.

This engineering thesis is devoted to the creation and operation of such a virtual environment, built on the basis of QEMU / KVM technologies. For this purpose, a project was created based on virtual machines running on two physical computers located within the same internal LAN network. A web application written using Bootstrap, NodeJS, JavaScript and Perl technologies was built to manage virtual resources.

The work begins with the theoretical part, which consists of the following chapters:

1. *Theoretical basics of virtualization* ,
2. *Review of the existing types of virtualization.*

The first chapter describes the process of virtualization, presents its beginnings and explains the basic concepts related to it.

The next chapter presents the division of this process into particular types, depending on the adopted concept; explains the differences between them and introduces the virtualization software currently available. Finally, the architecture of the network environment based on QEMU / KVM was characterized.

After the theoretical part, the project implementation process is described, the stages of which are

- ›Defining the requirements and goals of the completed project; presented in the third chapter entitled "Goals and assumptions of the completed project";
- ›. characteristics of parts of the project; described in chapter four - " Characteristics of the individual elements of the project";
- ›. implementation of the selected solution, which is included in chapter five - "Project implementation".

The work ends with final conclusions, which present the advantages and disadvantages of the architecture used in comparison to other currently available solutions.

1. Theoretical foundations of virtualization

1.1. What is virtualization?

Virtualization is a broadly understood concept that can be used, *inter alia*, in the context of networks, applications, servers, mass storage or even workstations. It is a method (technique) of managing resources that allows them to be shared by processes or even computer systems by means of partitioning and aggregation.¹ In other words, virtualization is the use of software to obtain an abstraction (illusion) of resources.

In a narrower sense, the process of virtualization is understood as the use of virtual machines (*VM*) or *virtual environment* (VE) in order to implement software intended for the same hardware platform, but requiring a special execution environment.²

1.2. The beginnings of virtualization

The first works with the use of virtualization were carried out in the years 1956-1962 during work on a project created in cooperation with the University of Manchester and Ferranti and Plessey, the result of which was a computer called ATLAS. In the created machine, some of the external instructions were executed by the hardware, and some by the *supervisor* . The computer's task was to properly execute the intercepted calls, and in this case virtualization was only a side effect of the solutions used.³

The actual use of virtualization took place only in the mid-1960s in New York City as a result of the work of IBM (*International Business Machines*) and MIT (*Massachusetts Institute of Technology*) on the M44 / 44X project. This project consisted of a physical M44 machine based on the IBM 7044 hardware platform and virtual machines (then called *pseudo-machines*) 44X. The virtual machines shared the memory and the software of the M44 mother computer. The solution used was to accurately clone this computer by 44X machines. The presented project was purely scientific and has never been used commercially, but it has become a model for further work on virtualization. More virtual machines were created soon, including:

¹A, Chrobot, G. Łukawski G., *Operating Systems - Virtualization* , Department of Computer Science, Kielce University of Technology, p. 3

²Ibid, p. 4

³ *Wirtualizacja* , <https://pl.wikipedia.org/wiki/Wirtualizacja> (accessed December 07, 2015)

- CP40 (1965) - based on the IBM 860/40;
- CP67 (1965) - Based on the IBM360 / 67, it was the first 32-bit computer to support full virtualization;
- CP67 (1967) - successor to CP40, based on the IBM360 / 6 architecture;
- VM / 370 (1970) - a series of popular VMs based on CP67.

The above-mentioned machines were perfect copies of the physical hardware, operated by the virtual machine monitor, which was in direct contact with the computer system. In this way, many copies of this operating system were created, which significantly increased the CPU usage on Mainframe computers.⁴

Citrix Systems (1989) and *VMware* (1998) appeared , producing virtualization software.

Microsoft also joined the industry and in 2003 acquired *Connectix Corp* , founded in 1988 , which developed an operating system virtualization program. *Microsoft Virtual PC* , widely used today , was designed to run Windows on a *Mac OS X host* .⁵

Another significant event was the appearance in 2007 of the *Oracle Virtualbox program* , created by *Innotek GmbH* , but today issued under the Oracle logo .⁶

Currently, we can still expect popularization of virtualization and increased use of virtual machines, also in home stations.

1.3. Criteria for the realizability of a virtual machine (VM)

The theoretical basis for virtualization was formulated in 1974 by Gerald J. Popek and Robert P. Goldberg, who in the article *Formal Requirements for Virtualizable Third Generation Architectures* listed the conditions that a virtual machine must meet in order to function properly. ⁷According to their thesis, there are three such properties:⁸

⁴ *Wirtualizacja* , <https://pl.wikipedia.org/wiki/Wirtualizacja> (accessed December 07, 2015)

⁵M. Stachowicz, *Application Virtualization. Comparison of Technologies* , Silesian University of Technology: Faculty of Materials Science and Metallurgy, Katowice 2011, p. 9

⁶Ibid, p. 10

⁷JPBuzen, UO Gagliardi, *The Evolution of Virtual Machine Architecture* , AHPS Press, Montvale, NJ, pp. 291-300

⁸RP Goldberg, GJ Popek; *Formal Requirements for Virtualizable Third Generation Architectures*; Honeywell Information Systems, Harvard University; Los Angeles; p. 417

- adequacy (*equivalence*) - a program running in a virtual machine must behave in the same way as on physical hardware;
- resource control (*resource control*) - the virtual machine must have full control of all resources that are virtualized;
- performance *efficiency* - that is, most of the instructions must be executed without the virtual machine (by hardware).

In practice, all three conditions do not need to be (and usually are not) met exactly.

Popek and Goldberg also split the virtual machine instructions into three groups:⁹

- **privileged instructions** - their effect is an interrupt or system call in user mode or missing in kernel mode;
- **control sensitive instructions** - they can change the configuration of system resources;
- **execution sensitive instructions** - their operation depends on the system configuration.

Following their reasoning, the most difficult in terms of implementation is the execution of sensitive and privileged instructions, which in turn leads to the separation of the next three types of virtualization:¹⁰

- **full virtualization with the use of binary translation ;**
- **system virtualization / paravirtualization ;**
- **hardware virtualization .**

Taking into account the defined instruction sets and constraints in the form of conditions, Popek and Goldberg formulated a theorem that became the basis for the development of virtualization technologies:

⁹Ibid, p. 418

¹⁰RP Goldberg, GJPopek; *Formal Requirements for Virtualizable Third Generation Architectures*; Honeywell Information Systems, Harvard University; Los Angeles; p. 420

*For any standard third-generation computer, a virtual machine can be constructed if the sensitive instruction set is a subset of the privileged instruction set.*¹¹

It follows from the above statement that any operation that causes a virtual machine to malfunction causes an abort or a system call, and all the rest of the unprivileged instructions are simply passed to the physical hardware for performance.

However, it did not work in practice, because you can also virtualize without meeting this assumption, agreeing to a decrease in performance. This is the case, inter alia, for the x86 architecture.¹²

¹¹M.Sawicz, *Virtualization but what for?* , <http://jesien.linux.org.pl/2008/materialy/prelcje/wirtualizacja.pdf> ; p. 7 (read on December 7, 2015)

¹²P.Przybylak; *Virtual Infrastructure - A New Approach to Systems* ; [http://zeszyty-naukowe.wwsi.edu.pl/zeszyty/zeszyt4/Wirtualna Infrastruktura - Nowe Podejscie Do Systemow.pdf](http://zeszyty-naukowe.wwsi.edu.pl/zeszyty/zeszyt4/Wirtualna%20Infrastruktura%20-%20Nowe%20Podjecie%20Do%20Systemow.pdf) ; p . 13 (accessed December 7, 2015)

2. Review of the existing types of virtualization

2.1. Types of virtualization

Currently, there are many different divisions of virtualization technologies, defined depending on what and how they virtualize. Each vendor of virtualization software has a different approach to this process.

For example, VMware distinguishes the following types of virtualization:¹³

- *Hardware Assisted Virtualization* ,
virtualization using the binary translation technique ,
- *Software Assisted Virtualization or Paravirtualization* .

According to Microsoft, there are the following types of virtualization:¹⁴

- *Type 1 Hypervisor* ,
- *Type 2 Hypervisor* ,
- *Monolithic Hypervisor* ,
- *Microkernel Hypervisor* .

Professor Andrew S. Tanenbaum, creator of the operating system *MINIX* claims that virtualization should be divided into the following types:¹⁵

- type 1 hypervisor,
- type 2 hypervisor,
- paravirtualization.

¹³ *Understanding Full Virtualization, Paravirtualization, and Hardware Assist*; 2007; ss. 3–6
https://www.vmware.com/files/pdf/VMware_paravirtualization.pdf (data odczytu 17 grudnia 2015)

¹⁴ M. Tulloch; *Understanding Microsoft Virtualization Solutions, From the Desktop to the Datacenter*; Microsoft Press; Redmond Washington 2010; ss. 23–27

¹⁵ A. S. Tanenbaum, *Systemy Operacyjne*, Gliwice, Helion 2010, ss. 671–677

According to other authors, virtualization can be divided according to the type of virtualized physical resources into the following types:

- server virtualization,
- desktop virtualization,
- network virtualization,
- application virtualization,
- mass storage virtualization.

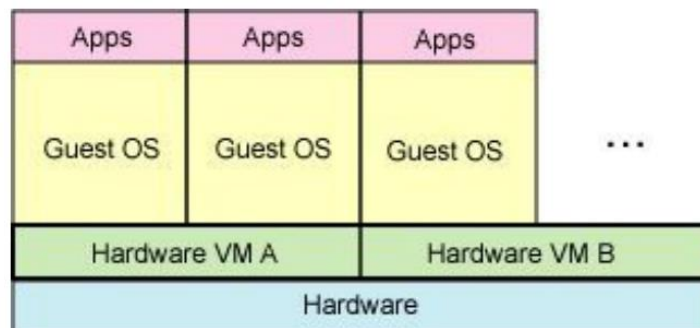
In this work, the division proposed by Wright State University, which takes into account the types most often appearing in scientific journals, was used.¹⁶

They are in turn:¹⁷

- **hardware emulation (*Emulation-based Full Virtualization*)** - it is a full emulation of hardware, that is, each element of a computer component (processor, memory, hard disk and input / output devices). This process allows you to run environments with a completely different architecture, but it is inefficient and slow.¹⁸

This type includes: Bochs, VirtualPC for Mac, QEMU.¹⁹

Fig 1. Hardware emulation scheme



Source: M. Bishopp, *Introduction to Computer Security* , Addison-Wesley, 2005, p. 649

¹⁶ Editorial Board; *International Journal of Security and Networks*, www.inderscience.com/ijns/; Wright, Ohio 2004, s.14

¹⁷ M Bishopp, *Introduction to Computer Security*, Addison-Wesley, 2005, s.645

¹⁸ *Wirtualizacja* , http://ii.uni.wroc.pl/~msq/so09/slides/2_4_wirtualizacja.pdf (accessed on December 20, 2015)

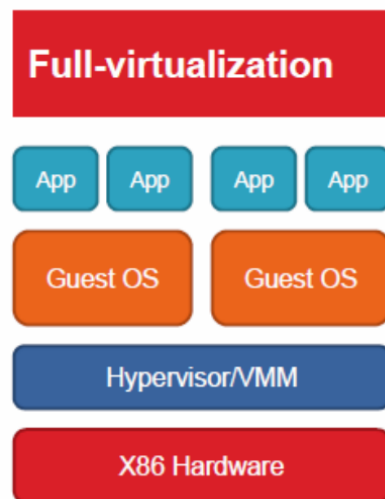
¹⁹ *Virtualization* , <http://cecs.wright.edu/~pmateti/Courses/7380/Lectures/Virtualization/Slides/vm-umass-2015.pdf> (accessed December 17, 2015)

- **Native / Full Virtualization** - **this** type of virtualization requires meeting the Popek-Goldberg criteria. It is more efficient than traditional emulation because it allows you to run an operating system that is compatible with your host system. It is therefore possible to run any operating system in the guest virtual machine.

In this process, only a fraction of the calls needed for software isolation are virtualized. The machine virtualizes only those instructions that might conflict with the activities of other environments virtualized or the host operating system. Everything else is processed by physical hardware to increase efficiency.²⁰ Privileged instructions (e.g. memory access , disk operations) are usually modified so that they only apply to the virtual environment in which they are executed. The unprivileged instructions (e.g. , I/ O , network support) are usually passed directly to the processor.

The scheme of the construction of this technology is shown in the figure below.

Fig. 2. Scheme of building full virtualization on the example of x86 architecture



Source: *Wirtualizacja* , http://ii.uni.wroc.pl/~msq/so09/slides/2_4_wirtualizacja.pdf (accessed on December 17, 2015)

undoubted advantage of native virtualization is the need to emulate each device, which makes the processor of the physical machine heavily loaded. The

²⁰ *Virtualization* , <http://cecs.wright.edu/~pmateti/Courses/7380/Lectures/Virtualization/Slides/vm-umass-2015.pdf> (accessed December 17, 2015)

emulator does everything in a loop that the actual processor of a physical machine would do, which in turn results in a reduction in computer performance - even up to 20-30% of the virtualization overhead.

On the other hand, the advantages of this solution include no need to modify the system running in a virtual environment, the ability to use any operating system in the host environment and very good isolation between virtual environments.²¹

The full virtualization technologies include: *VMware ESX Server* , *Microsoft Virtual Server* .²²

- **Hardware -assisted Virtual Machine (HVM)** . *Hardware-assisted Virtual Machine* . It is an extension of the full virtualization technique through the possibility of using the technology of new processors (Intel-VT, AMD-V) ²³, which implement hardware virtualization support . This method appeared in 2007 with the creation of the above-mentioned processors.

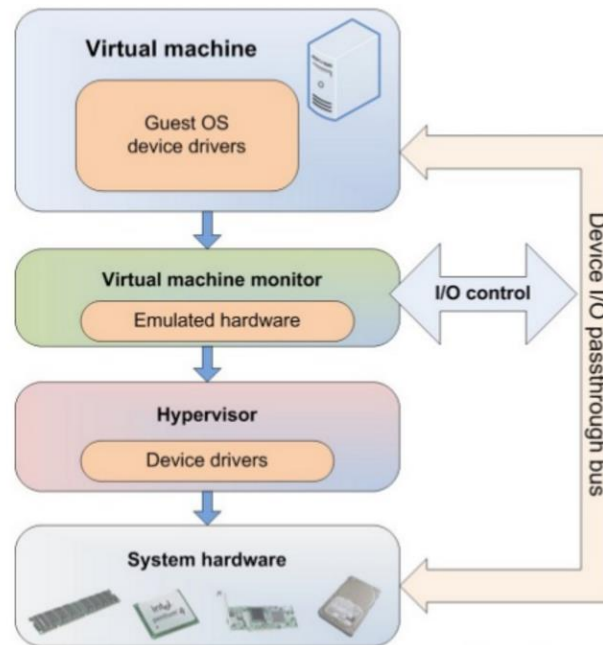
²¹M. Bishopp, *Introduction to Computer Security* , Addison-Wesley, 2005, p. 660

²²G. Terlikowski, *Introduction to Virtualization and RIA* , Institute of Informatics UPH in Siedlce, Siedlce 2015, p. 11

²³<http://bip-slaskie.pl/zamniane/18/1192108801/1299765671.pdf> , p. 6 (accessed on December 20, 2015)

The scheme of operation of this technology is shown in the figure below:

Fig. 3. Schematic diagram of hardware-supported virtualization



Source: J. Huang, *Embedded Virtualization applied in Mobile Devices*, Academia Sinica, Taiwan 2012, p. 12

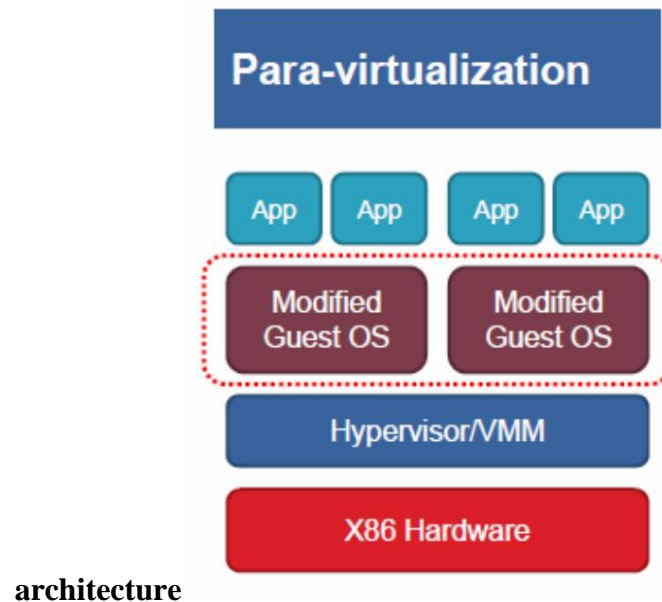
The big advantage of this type of virtualization is increased efficiency - thanks to hardware support, some tasks are performed directly by the host processor without the costly process of emulation and modification of the kernel. This technology is more efficient compared to full virtualization. In addition, it meets the same assumptions as native virtualization, i.e. the ability to run any operating system on the host, no need to modify the operating system, well-isolated virtual environments.

A significant disadvantage of this technology, however, is the need for hardware virtualization extensions in the processor. If you intend to create virtual machines on a physical dedicated server, first make sure that the processor has the appropriate extensions - *Intel Virtualization Technology* (VT) or *AMD-Virtualization* (AMD-V). Most processors on the market today meet these requirements. Solutions of this type of virtualization include: KVM, Xen HVM, VMware ESXi vSphere, MS Hyper-V.

- **paravirtualization (hypervisor)** - only supported on kernel-modifiable operating systems. The main goal is to achieve the maximum possible performance by eliminating the need for hardware emulation and appropriate kernel transformations. The virtual machine does not emulate hardware, but implements API support that can be used by suitably modified guest operating systems. The API's job is to divide physical resources between virtual machines. The software does not emulate all hardware, but only provides disjoint access to some physical components, emulating only some.²⁴

The scheme of building this type of virtualization is presented below:

Fig. 4. Schematic structure of paravirtualization on the example of x86



Source: *Wirtualizacja*, http://ii.uni.wroc.pl/~msq/so09/slides/2_4_wirtualizacja.pdf (accessed December 21, 2015)

Paravirtualization is an interesting approach to solving the problem of access to specific processor registers by many operating systems simultaneously.

²⁴ <http://winntbg.bg.agh.edu.pl/rozprawy2/10081/full10081.pdf> (accessed on December 21, 2015)

This type of update vortex uses a type 1 hypervisor. One of the guest systems is privileged, but the kernel of the guest systems must be modified to reference the hypervisor.²⁵

The advantage of paravirtualization is very good performance of guest systems with a modified kernel. The disadvantage, however, is the inability to run any operating system. Properly modified kernels have Linux and FreeBSD systems, but we will not run Windows systems in this way. Examples of a paravirtualization technology solution are: Xen PV, Parallels Workstation, Enomalism, Vmware ESX Server, and Win4Lin 9x.

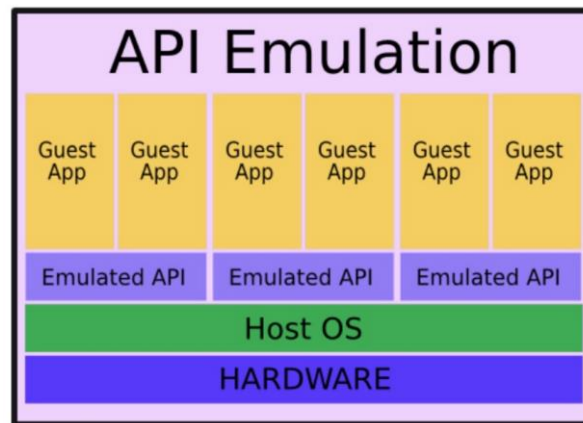
- **API emulation** - in this technology, instead of the entire hardware and operating system, only the application interface is emulated. A virtual machine provides an application running on it, which sometimes does not even need to be virtualized, virtual registries and file access. Thanks to this, conflicts between applications and between the application and the operating system are eliminated.²⁶

²⁵ <http://bip-slaskie.pl/zamniane/18/1192108801/1299765671.pdf>; p. 6 (accessed December 21, 2015)

²⁶ A.Bula; *Virtualization of Operating Systems - Comparative Analysis*, http://www.math.uni.opole.pl/dokumenty/Int/Wirtualizacja_systemow_operacyjnych.pdf, p. 9 (accessed on December 23, 2015)

The advantage of this solution is the portability of the application, easier sharing it with users and managing its licenses. The disadvantage, however, is the need to install a virtual machine on the position where the application is to be run. An example of this technology can be, among others: *Microsoft Application Virtualization* , *Citrix XenApp*, *VMware ThinApp* .²⁷

Fig. 5. Diagram of API emulation structure



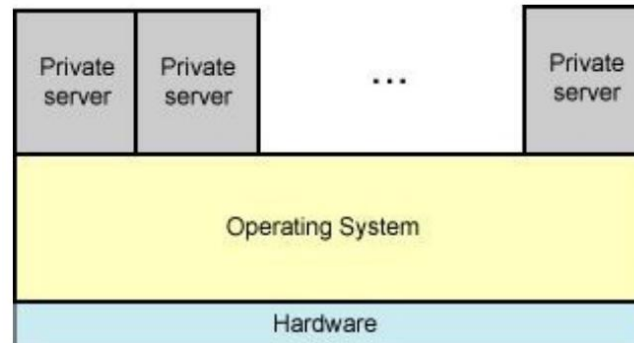
Source: S. Wójtczak, *Virtualization of Operating Systems* , University of Łódź, Łódź 2008, <http://www.strony.toya.net.pl/~vermaden/tmp/thesis.pdf> , (accessed December 23, 2015)

- **OS-level virtualization** - especially popular with VPS vendors . This type of virtualization consists in dividing one physical machine into several virtual ones, based on the same (or similar) operating system in the so-called container.

Technically, it is a way of isolating consecutive instances of the same operating system within one physical machine. The container has separate resources and isolation of processes and devices, which ensures the security of other containers and the host system. This solution is very efficient due to the lack of the need to create an entire virtual computer and emulate devices. Containers also have significantly more limitations compared to hardware virtualization. The system running in the container shares the kernel with the host system, so it is not possible to compile your own modules, update the kernel.

This type of virtualization is perfect for typical hosting applications, such as e-mail, www. Solutions of this technology are available, among others, in *OpenVZ* (open version of *Virtuozzo*) , *Parallels Virtuozzo* , *LXC* (native technology embedded in the Linux kernel).

Fig. 6. Scheme of building operating system virtualization



Source: L.Parziale, B. Gunreben, F. Miranda; *The Virtualization Cookbook for IBM* ; IBM Books 2014; p. 32

2.2. Type 1 hypervisor vs. Type 2 hypervisor

The concept of virtualization is related to the already mentioned concept of a hypervisor. Hypervisor, or *Virtual Machine Monitor (VMM)* watches over the proper course of the virtualization process. It is a computer program that allows you to run multiple operating systems on the same computer at the same time. When the virtual machine's operating system (guest) is recalled to the computer hardware, its request is intercepted and then handled by the hypervisor. Thanks to this, parallel access to computer hardware is possible through many operating systems operating simultaneously.²⁸

There are two types of hypervisors depending on where they operate.

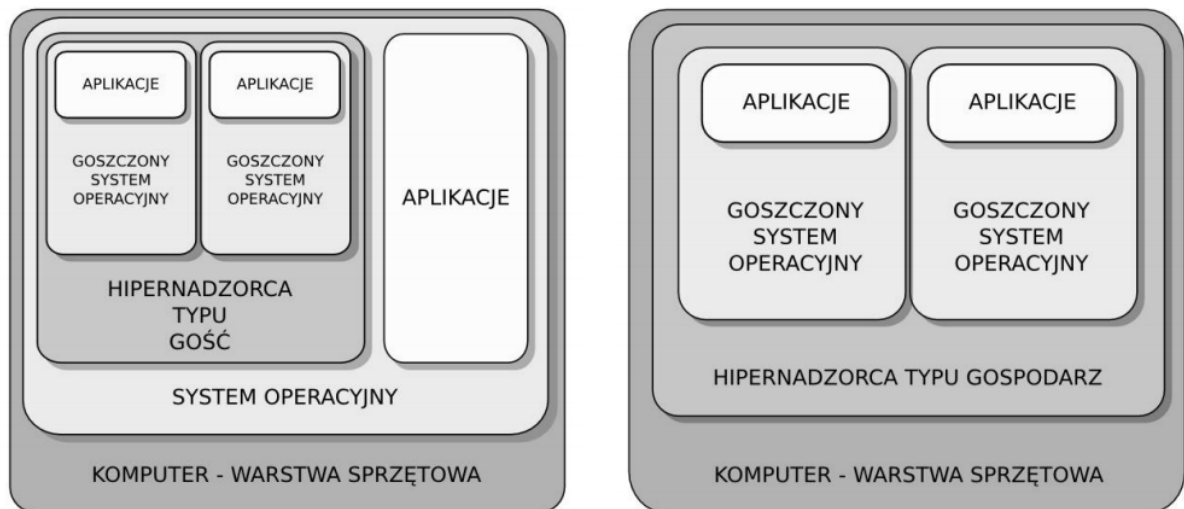
These are:

²⁸L. Kaczmarek, M. Wróbel; *Possibilities of Using Virtualization in Computer Systems; Application of Computers in Science and Technology: Scientific Papers of the Faculty of Electrical and Control Engineering*, Gdańsk University of Technology No. 30; Gdańsk 2014; p. 2

- **Type 1 hypervisor / native** (*Native Virtual Machine Monitor* , " *bare metal*") - works directly at the hardware level, having full control over it and monitoring running operating systems.
- **Type 2 hypervisor / Hosted Virtual Machine Monitor** - runs as a program running on a given operating system (host).²⁹

The differences between the two types are shown in the figure below:

Fig 7. Virtual machine operation models with guest (type2) and host (type1) hypervisor



Source: L. Kaczmarek, M. Wróbel; *Possibilities of Using Virtualization in Computer Systems. Application of Computers in Science and Technology*: Scientific Papers of the Faculty of Electrical and Control Engineering, Gdańsk University of Technology No. 30; Gdańsk 2014; p. 2

2.3. Overview of selected virtualization software

Below is a list of the most popular virtualization solutions today, commonly known as "virtual machines". Depending on various factors (e.g. guest operating system), the following software may use several different types of virtualization simultaneously:

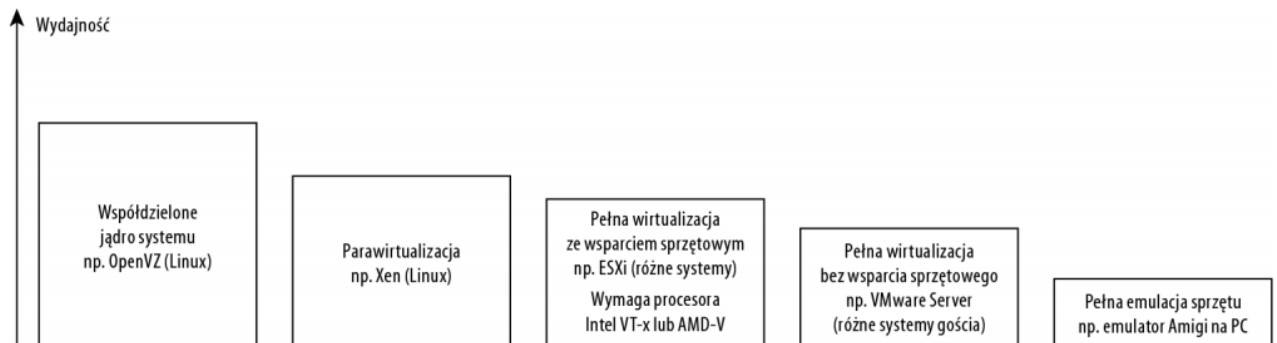
- **VMware ESXi, VMware Server** - full virtualization, virtualization with hardware support, also elements of paravirtualization;

²⁹Ibid, p. 3

- **Xen** - paravirtualization, full virtualization with hardware support, emulation;
- **Microsoft Hyper-V, Virtual PC** - full virtualization, virtualization with hardware support, emulation;
- **VirtualBox** - full virtualization, virtualization with hardware support, emulation;
- **OpenVZ** - shared kernel - highest performance, but limited to only one type of operating system (Linux).

The comparison of the performance of these technologies is shown in the figure below:

Fig. 8. Comparison of virtualization technology performance



Source: M. Serafin, *Virtualization in Practice*, Helion, Gliwice 2012, p. 16

2. 4. Architecture of virtualization based on QEMU / KVM

This paper deals in detail with virtualization based on QEMU / KVM technologies. In the previous chapter it was mentioned that KVM is an example of **Full Hardware Supported Virtualization** (HVM), while QEMU belongs to the emulation type. But how do they function within one configuration?

To better understand these issues, please refer to the term emulator.

It has already been mentioned that emulation is the process of simulating by a computer program running in a given computer system the full behavior of another computer system,

most often with a different architecture. The program that performs these simulations is called an emulator. In practice, we distinguish two types of emulators:

- **full emulators** - that is, those that work according to the specification of a specific physical computer. This group includes QEMU, as well as Bochs, UAE and Frodo
- **API emulators** - which only emulate the behavior of the user interface of a specific operating system, they are often called compatibility layers. This group includes, for example, Wine, DosBox, Dosemu.

It is therefore known that QEMU is both a virtual machine and an emulator. It is an emulator because it can run both individual programs and entire operating systems created for a different processor than the one on which it is currently working. This is done by fast translation of the target processor's instructions into the orders of the processor on which QEMU is currently working.

On the other hand, in virtualization mode (using Xen or KVM), QEMU runs on the host CPU.

As for the KVM (ang. *Kernel-based Virtual Machine* - it is a solution that allows for virtualization using a hypervisor acting as a Linux kernel module. The KVM component appeared in the Linux kernel with version 2.6.20 (released January 2007). KVM is a virtualization platform for Linux on x86 hardware that includes virtualization support extensions (Intel VT or AMD-V). It consists of a kernel module (named `kvm.ko`) that provides the basic virtualization infrastructure, CPU-specific modules (`kvm-intel.ko` or `kvm-amd.ko`), and a user space component (a modified version of QEMU). Accordingly, the virtual machine is implemented as a normal Linux process so that KVM uses all the kernel features, incl. provides the standard Linux security model and resource control. A modified version of QEMU is responsible for device emulation, which provides an emulated BIOS, PCI bus, USB bus and a standard set of devices, such as IDE and SCSI disk controllers, network cards, etc. Additionally, QEMU in such architecture supports two operating modes: emulator and hypervisor with full virtualization support (no support for virtualization hardware extensions).

Regarding memory management, KVM inherits it from Linux. Virtual machine memory is stored like the memory of any process, so it can be swapped, and the sharing of memory pages is additionally supported by KSM (*Kernel Same-Page Merging*). KSM looks through the memory of each virtual machine and if it finds identical pages of memory, it sticks them together into one shared page. If a visitor tries to make modifications, they will get a copy of their own.

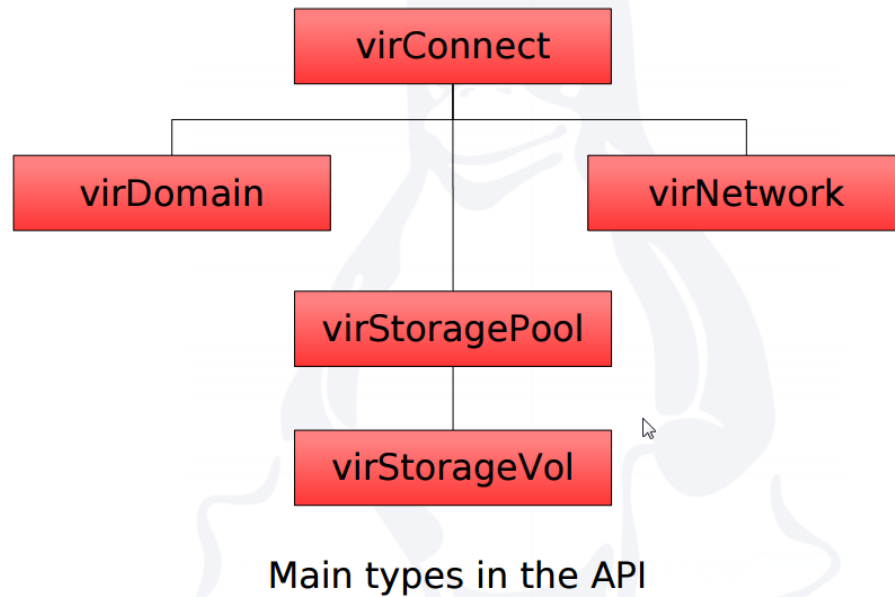
To manage virtual machines in this technology, the Libvirt API library was used, which is a universal and open API for operating and managing virtual machines in GNU / Linux systems. It also enables communication with various hypervisors - locally and remotely. Currently, Libvirt API supports the following technologies, and its most intense development can be seen under RHEL / CentOS and Fedora:

- KVM / QEMU
- OpenVZ
- Xen
- VirtualBox
- LXC
- VMware ESX
- MS Hyper-V
- IBM PowerVM

The undoubted advantage of Libvirt API is the diversity in terms of managed functionalities of virtual machines. The most important of them are presented in the figure below:

Fig 9. Variation in the functionality of Libvirt API

Libvirt API



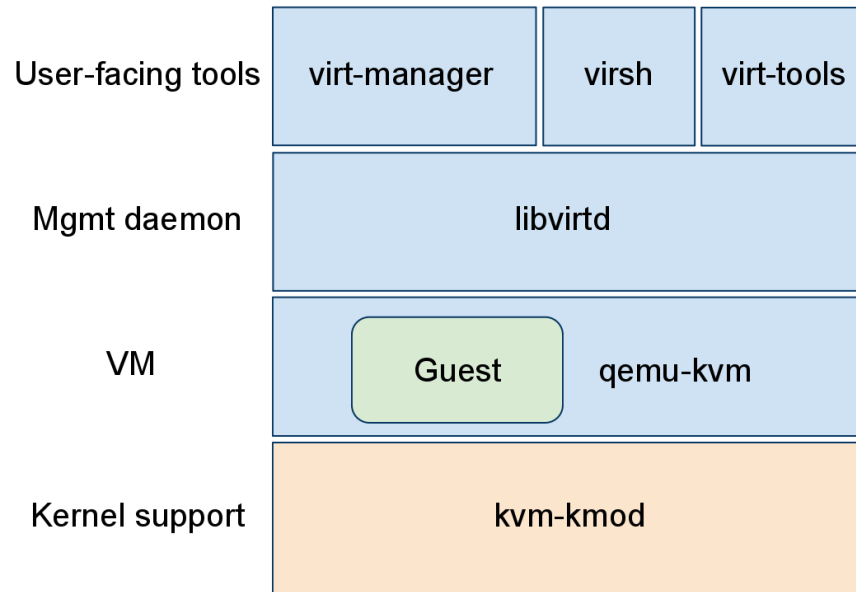
Source: <http://www.ibm.com/developerworks/library/l-libvirt/> (accessed February 21, 2016)

Above The API is responsible for configuring almost all aspects of virtualization. As part of its operation, the universal *virsh console is available* , allowing, among others, the creation and configuration of virtual machines, storage configuration and network management. Remote access is also possible using the following protocols: SSH, TLS, TCP.

Of course, the Libvirt API alone in combination with KVM / QEMU is not enough to be able to effectively manage virtual environments. A set of tools is still needed to implement its functions. In this work , the *Virtual Machine Manager* graphical application was used for this purpose , which manages virtualization through the libvirt library.

The whole architecture used is presented in the following diagram:

Fig. 9. Scheme of building the QEMU / KVM architecture



Source:

https://www.ibm.com/developerworks/community/blogs/ibmvirtualization/entry/kvm_architecture_the_key_components_of_open_virtualization_with_kvm2?lang=en (accessed February 21, 2016)

3. Goals and assumptions of the completed project

3.1. Project description and goals

The overall goal of the project carried out for the purposes of the project was to create virtual environments based on QEMU / KVM technology and to build an application that enables remote virtual management. resources.

The implemented system was made on the basis of three physical machines, successively named comp1, comp2 and comp3, which are located within the same LAN local network.

One of the machines serves as an HTTP server, with a web application for remote management of virtual and physical machines. Only the network administrator has access to it, because the application runs only on his physical machine and is configured only on that computer.

The other two physical computers have virtual machines with installed operating systems: Slitaz40 and Damn Small Linux.

The infrastructure created in this way facilitates and accelerates access to individual operating systems on the same physical computer.

This is an undoubted advantage both in terms of research and teaching, as well as economic. Thanks to this, you can reduce the costs associated with the purchase of software licenses and the costs of access to individual applications.

It is also important to use resources efficiently and to reduce the susceptibility of systems to failures and overloads. The project allows you to safely run application tests on various operating systems. You can also easily save the configuration state and return to it at another, more convenient time. This leads to the consolidation of IT systems, which is confirmed by the ever-growing popularity of virtualized solutions.

3.2. Hardware requirements and assumptions

The basic hardware requirement of the infrastructure described in the following chapters is a properly functioning local network with Internet access. The TP-Link TL-WR1043ND router was used to configure it in the project.

Because a bridge was used when configuring virtual machines, it does not work with interfaces of the type wireless , the local network must be *wired* .

Before you start working with KVM, you should check whether the processor of your physical machine has support for VT (*Virtualization Technology*). To verify this information, go to the / **proc** / **cpuinfo** file and check for the **vmc** (for Intel processors) or **svm** (for AMD processors) flags. Processors supporting virtualization have appropriate hardware support - for AMD processors it is AMD-V technology, and for Intel processors - VT-x technology.

The list of currently available processors supporting virtualization is presented below:

Intel:

- Pentium 4 662 and 672, Extreme Edition 955 and 965 (not including Pentium 4 Extreme Edition with HT).
- Pentium D 920 to 960, omitting 945, 925 and 915;
- Core Duo T2300, T2400, T2500, T2600, T2700, L2000 and U2000;
- Core 2 Solo;
- Core 2 Duo except E6540, E8190, E7xxx, E4xxx, T5200-T5550 and T5750;
- Core 2 Quad except Q8200;
- Core 2 Extreme Duo and Quad;
- Xeon 3000 series;
- Xeon 5000 series;
- Xeon 7000 series.

AMD:

- Athlon 64 and Athlon 64 X2 "F" and "G" series for AM2 sockets (except 939);
- Turion 64 X2;
- Opteron 2nd and 3rd generation;
- Phenom.

Physical machines must also have enough memory so that you can create three virtual machines (and their copies) on each of them.

Additionally, when using KVM, it is recommended that the computer has a dual-core or quad-core CPU.

As for QEMU, this technology supports the following architectures: x86, x86-64, PowerPC, SPARC; however, it has incomplete support for Windows.³⁰ Additionally, the use of QEMU is possible only from version 2.6.2.3 of the Linux kernel.

³⁰E. Łukasik, M. Skublewska-Paszkowska, J. Smółka J; *Android and iOS - creating mobile applications* ; Lublin University of Technology; Lublin 2014; p. 70

Three physical machines (two laptops and one desktop type) with the following technical parameters were used to implement the project presented in the paper:

Table 1. List of selected technical parameters of the equipment

computer name	computer type	computer model	operating system	technical parameters
comp1	laptop	Acer Aspire AS4830T- 6678	Xubuntu14.02	<ul style="list-style-type: none"> - CPU: Intel® Core™ i3-2370M - graphics card: Intel HD Graphics 3000 - Motherboard: Mobile Intel® HM65 Express chipset. - network card: 802.11b / g / n Wi-Fi CERTIFIED™, 10/100/1000 Gigabit Ethernet LAN Sound card: Dolby® Advanced Home Theater® v4 Audio Enhancement, -storage: SATA 3 - 6Gb / s -RAM: 4GB -hard drive: HDD, 320 GB, SATA

comp2	desktop	Dell OptiPlex 790	Xubuntu14.02	<ul style="list-style-type: none"> - CPU: Intel Core i5 (2nd Gen) 2400 / 3.1 GHz - graphics card: Integrated Intel® HD Graphics 2000 - Motherboard: Intel® H65 Express Chipset -Network card: Intel® 82579LM Gigabit1 Ethernet LAN 10/100/1000 - Sound card: Realtek ALC269Q High Definition Audio Codec -storage: Serial ATA-600 -RAM: 1333 MHz -hard drive: HDD, SATA 6Gb / s
-------	---------	-------------------------	--------------	---

comp3	laptop	Dell Inspiron 17 5758	Windows7 Professional	- CPU: Intel (R) Core (TM) i7-4800MQ CPU @ 2.70GHz (8 CPUs) - graphics card: NVIDIA GeForce 920M - motherboard: 1000 GB HDD -Network card: Intel® 82579LM Gigabit1 Ethernet LAN 10/100/1000 - Sound card: Realtek ALC269Q High Definition Audio Codec -storage: Serial ATA-600 -RAM: 8GB - hard disk: 1000 GB HDD
-------	--------	-----------------------------	--------------------------	--

Source: own study

As you can see, all of the above processors, except the Intel (R) Core (TM) i7-4800MQ CPU @ 2.70GHz (8 CPUs), support virtualization.

3.3. Program requirements and assumptions

The main software requirement when creating and configuring virtual machines using QEMU / KVM is the need to install additional extensions and tools on the host machines for the configuration of virtual machines and the network running on them. Installation can be done from the distribution's repository.

The auxiliary tools that have been installed are mainly:

- bridge-utils - network bridge building tool;
- QEMU / QEMU-KVM– QEMU with KVM support;
- libvirt - daemon libvirt;
- virt-manager - a graphics application that creates and manages virtual machines by libvirt.

Libvirt API is the default KVM / QEMU machine management interface on most modern GNU / Linux distributions. Down machine operation uses the built-in *virsh console* and the previously mentioned virt-manager application.

In addition to the above-mentioned additional configuration tools, four different operating systems were used to complete the project. Xubuntu14.02 is installed on host computers (comp1 and comp2). With the turn of the guests; i.e. virt1, virt2, virt3 virtual machines; small-capacity distributions Slitaz40 and Damn Small Linux were used.

The last physical machine, comp3, runs on the *Windows7 Professional operating system* and serves as an HTTP server with a resource management web application written in JavaScript using the Bootstrap platform.

Xubuntu system configuration 14.02. on the hosts is identical for both comp1 and comp2.

The software installed and configured on this system is the standard distribution software and the aforementioned additional tools.

Standard software includes :

Graphic environments:

- XFCE4;
- GIMP 2.8.10;

Development tools:

- GCC 4.9.0;
- Eclipse 4.4;

Additional software:

- Python-2.7.5;
- Vim 7.4.52;
- Mozilla Firefox 28.0;
- Libreoffice 3.6;
- WinRAR 5.11;
- Wine 1.7;
- Gnumeric 1.12.9;
- Libvirt 1.2.2;
- Apache v2;
- bridge-utils;
- qemu-kvm;
- Virtual Machine Manager.

As for the computer with Windows7 Professional it has the following software :

- Adobe Reader X1;
- Firefox 19.0.2;
- Java SE Development 7u17;
- JDK 7u17 with NetBeans 7.3;
- NetBeans IDE 7.3;
- Eclipse 4.4;

- Notepad ++ v6.3.1;
- Silverlight 5;
- VNC-5.0.5 Viewer;
- VLC media player 2.0.5;
- Visual Studio 2012;
- 7-zip 9.20;
- PUTTY 0.67;
- PuttyGen;
- Google Chrome (46.0);
- Apache v 2.

Due to the fact that the computer comp3 runs an application based on the Bootstrap framework, the following components are also installed on this computer:

- Microsoft .NET Framework 1.1,
- Internet Explorer 6.0 SP1 or higher,
- jQuery version1;
- Bootstrap v3.3.5;
- AngularJS 1.4.1;
- Node Express 4.13.0.

4. Characteristics of individual elements of the project

4.1. Characteristics of local and virtual networks

All three physical machines used in the project are on the same LAN and have static IP addresses, assigned manually. Each IP is assigned a specific physical computer name - comp1, comp2, comp3.

A laptop with the Windows7 Professional (comp3) operating system runs a web application for managing a pool of virtual machines on the other two computers.

All three physical machines have an open ssh port for communication between them.

The network described in the project is characterized by the following parameters:

- **IP range for hosts: 153.65.206.15– 153.65.206.17**

Table 2. List of host IP addresses

computer name	Static IP
comp1	153.65.206.15
comp2	153.65.206.16
comp3	153.65.206.17

Source: own study

- **IP range for created virtual machines: 153.65.206.18– 153.65.206.30**

Table 3. List of virtual machine IP addresses

the name of the physical computer	virtual machine name	Virtual Machine IP
comp1	virt1	153.65.206.18– 153.65.206.23
	virt2	
	virt3	
comp2	virt1	
	virt2	
	virt3	

Source: own study

- **subnet mask: 255.255.255.0**
- **default gate: 153.65.206.1**
- **broadcast: 153.65.206.255**

As for virtual machines, they were created on computers comp1 and comp2 respectively. Each of these physical machines has three virtual machines named virt1, virt2, and virt3, respectively.

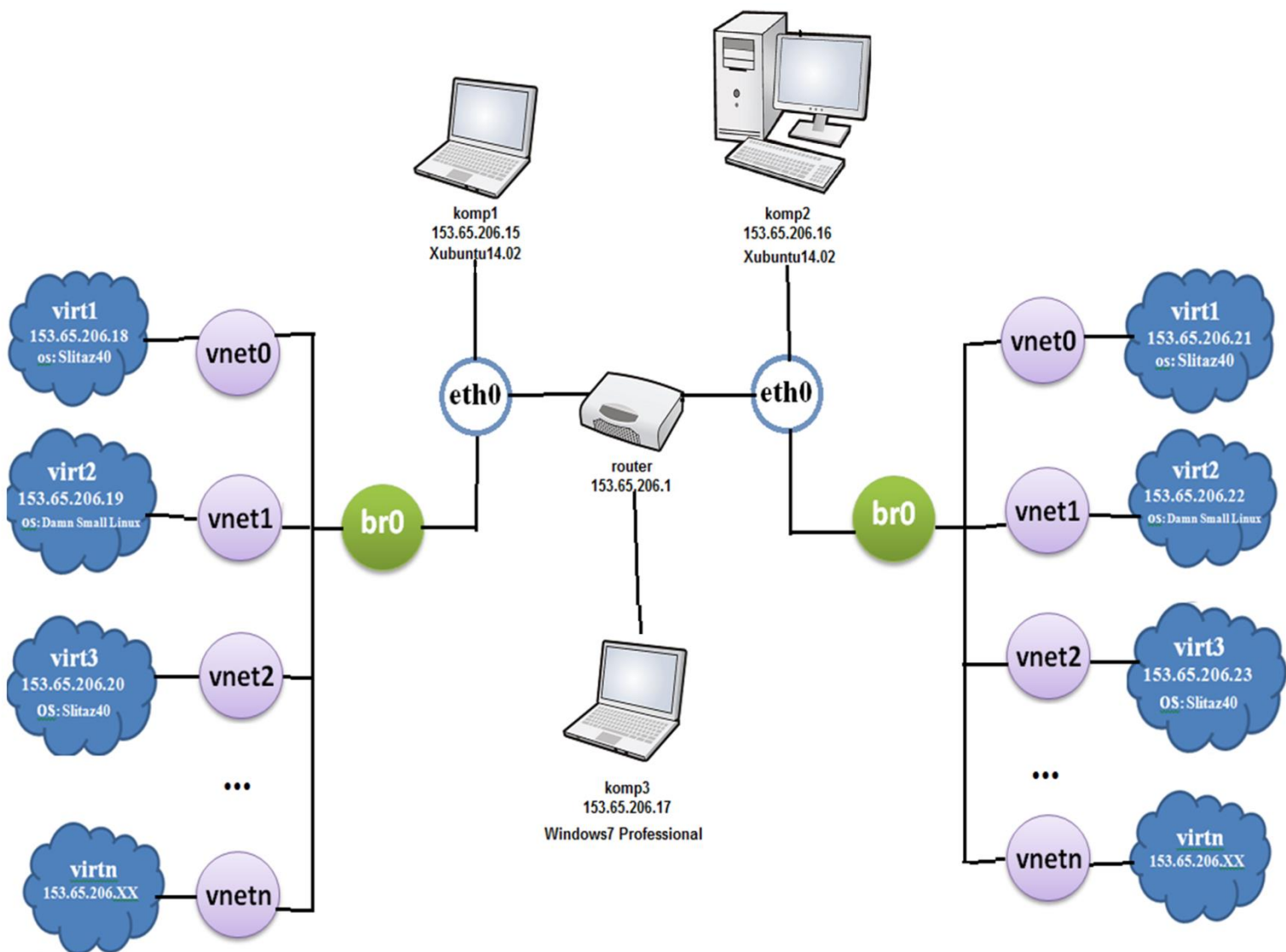
A bridge named br0 has been placed on the host's physical interface (eth0). The network bridge takes over the configuration of the host network. For each virtual machine, a vnet0, vnet1 and vnet2 virtual interface is created that is visible on the host system. The bridge (br0) connects the host's physical interface to the virtual guest interfaces so that virtual machines are able to send and receive packets over the host interface, but are unable to communicate with each other. In addition, the bridge has been properly configured so that it starts up at system startup.

Each virtual machine has a unique, automatically generated MAC address of a virtual network adapter. IP addresses are also assigned automatically from the available pool of addresses by the router's DHCP server. Each virtual machine is allocated a reservation for its

MAC address, thanks to which it receives a predetermined IP address. Specific names of virtual machines are assigned to the IP addresses - DNS and revDNS are configured.

The diagram of the network structure implemented in the project is presented in the figure below :

Fig 10. Scheme of building a project network



Source: own study.

On any virtual machine; virt1, virt2 and virt3; Slitaz40 and Damn Small Linux distribution systems were installed in small capacity .

Individual virtual machines are assigned images accordingly:

- virt1- Slitaz40,
- virt2- Damn Small Linux,
- virt3 - Slitaz40.

Virtual Manager was used to create virtual machines on hosts (you can also use virt-install and qemu-img create) .

Images of available systems are found in / **var / lib / libvirt / images /** . Virtual machine disks are saved in the qcow2 (*copy-on-write*) file format due to the fact that this format only stores changes in relation to the base image, thus saving disk space. In addition, it makes it easier to restore the image to a specific state, thus being an alternative to a snapshot.

The configuration of mounted virtual machines is stored in the domain **XML file, in the / etc / libvirt / qemu directory**. Both directories are owned by root, so only root can modify them.

Each virtual machine was created with the same parameters, they only differ in the type of operating system installed. A sample configuration file for a virt1 machine looks like this:

```
<domain type = 'kvm'>
  <name> virt1 </name>
  <uuid> (SNIP) </uuid>
  <memory> 524288 </memory>
  <currentMemory> 524288 </currentMemory>
  <vcpu> 1 </vcpu>
  <os>
    <type arch = 'i686' machine = ' pc-i440fx-trusty ' > hvm </type>
    <boot dev = 'hd' />
  </os>
  <features>
    <acpi />
    <apic/>
```

```

    <pae/>
</features>
<clock offset='utc'/>
<on_poweroff>destroy</on_poweroff>
<on_reboot>restart</on_reboot>
<on_crash>restart</on_crash>
<devices>
  <emulator>/usr/libexec/qemu-kvm</emulator>
  <disk type='file' device='disk'>
    <driver name='qemu' type='qcow2' cache='none'/>
    <source file='/var/lib/libvirt/images/virt1.img'/>
    <target dev='hda' bus='ide'/>
    <address type='drive' controller='0' bus='0' unit='0'/>
  </disk>
  <disk type='file' device='cdrom'>
    <driver name='qemu' type='qcow2'/>
    <source file='/var/lib/libvirt/images/virt1.img'/>
    <target dev='hdc' bus='ide'/>
    <readonly/>
    <address type='drive' controller='0' bus='1' unit='0'/>
  </disk>
  <controller type='ide' index='0'>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x01'
function='0x1'/>
  </controller>
  <interface type='network'>
    <mac address='(SNIP)'/>
    <source network='default'/>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x03'
function='0x0'/>
  </interface>
  <serial type='pty'>
    <target port='0'/>
  </serial>
  <console type='pty'>
    <target type='serial' port='0'/>
  </console>
  <input type='mouse' bus='ps2'/>
  <graphics type='vnc' port='-1' autoport='yes'/>
  <sound model='ac97'>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x04'
function='0x0'/>
  </sound>
  <video>
    <model type='cirrus' vram='9216' heads='1'/>

```

```
        <address type='pci' domain='0x0000' bus='0x00' slot='0x02'  
function='0x0' />  
    </video>  
    <memballoon model='virtio'>  
        <address type='pci' domain='0x0000' bus='0x00' slot='0x05'  
function='0x0' />  
    </memballoon>  
</devices>  
</domain>
```

The user can log into the system using an administrator-user account with appropriate privileges (username: admin, password: admin) or as an ordinary user (username: guest, password: guest).

4.2. The architecture of an application for managing virtual resources

And the file for remote management of virtual resources runs on the computer comp3. It is divided into two parts, each of them is presented in a separate page tab.

The first part is called *Computer Management* and is responsible for providing information about virtual machines running on selected physical machines.

The appearance of this part is shown below:

Fig 11. The layout of the HTML page of the *Computer Management* tab

Project name

Dashboard

Computer management

Virtual management

Komp1

-- Please change --

Komp1

Komp2

Komp3

Komp4

153.65.206.15

Turn off

Reboot

		Mac Address	Status	VM MaM Size	VM MaM Available Size	Last login	CPU	VM Last change	Actions
Komp1	vi	153.65.206.19	running	10GB	5GB	15:28 08.06.2015	50%	15:28 08.06.2015	Snapshot Clone Start Stop delete
Komp2	p			10GB	5GB				Snapshot Clone Start Stop delete
Komp3	p			10GB	5GB				Snapshot Clone Start Stop delete
Komp4	p			10GB	5GB				Snapshot Clone Start Stop delete

Reboot all physical machines

Shutdown all physical machines

Start selected on all

option

Stop selected on all

option

delete selected on all

option

Source: own study

The user is able to check what virtual machines are available on a given physical computer and what are currently running. The drop-down list next to the physical computer name contains the names of all available virtual computers on the machine. After selecting the name of the virtual machine, the rest of the available fields will be filled in automatically.

Information on:

- **IP** address of the virtual machine;
- **MAC** address of the virtual machine;
- virtual machine **status** - there are six types of possible virtual machine states, they are:
 - **running** - when the virtual machine is working properly and is turned on;
 - **blocked** - when the virtual machine is sleeping or waiting for I / O;
 - **paused** - when the virtual machine is paused by the administrator, it does not consume CPU resources;
 - **shutdown** - when the virtual machine is turned off;
 - **dying** - when the virtual machine is in the process of shutting down, or when it has almost stopped working due to a crash;
 - **crashed** - when the virtual machine is not available due to system crash.

It is worth noting that in order for the *crashed state to be* visible in response to the `virsh list --all` command, a given virtual machine must have settings in the XML configuration file, stating that in case of damage to the machine, it will restart.

The fragment defined by this parameter looks like this:

```
...
</features>
<clock offset = 'utc' />
<on_poweroff> destroy </on_poweroff>
<on_reboot> restart </on_reboot>
<on_crash> restart </on_crash>
  <devices>
...
```

- the current size of the virtual machine memory;
- available virtual machine memory size;
- date of the last login to the virtual machine system;
- CPU size of the virtual machine;
- date of the last modification of the virtual machine configuration.

There are also groups of *Actions buttons* , responsible for performing individual actions on a virtual machine. They are in turn:

- snapshot,
- clone,

- start,
- stop,
- delete.

There are also two buttons at the bottom of the page to control virtual machines:

- ***Start selected on all*** - after selecting the name of a virtual machine from the drop-down list located next to it and then pressing the button, the virtual machine will be started on all hosts on which it is available;
- ***Stop selected on all*** - after selecting the name of a virtual machine from the drop-down list located next to it and then pressing the button, the virtual machine will be turned off on all hosts where it is available;
- ***Delete selected on all*** - after selecting the name of a virtual machine from the drop-down list located next to it and then pressing the button, the given virtual machine will be removed from all hosts where it is available.

From this part of the application it is also possible to perform certain activities on physical machines. The buttons are used for this:

■ ***Reboot all physical machines*** - restart all physical machines;

■ ***Shutdown all physical machines*** - shutdown all physical machines.

and ***Turn off*** and ***Reboot*** , available when choosing the name of the physical machine. The first one is responsible for turning off the selected physical machine, the second one restarts it.

The second part of the application ; called ***Virtual Management*** ; allows you to check what virtual machine images are running on a given physical computer. For clarity, it is placed in a separate page tab:

Fig 12. The layout of the HTML page of the *Virtual Management* tab

Comp name	Status	Last Login	IP	CPU
Comp 1	Running	15:28 08.06.2015	192.192.192.192	50%

Source: own study

When you select a virtual machine image name from the drop-down list, the rest of the available fields are filled in automatically, providing data regarding:

- list of physical machines on which the image is available;
- the status of a given image on a specific physical machine at any given time;
- date of the last login to the operating system of the virtual machine on the given physical computer;
- Virtual Machine IP;
- CPU size of the virtual machine.

As for the graphical interface of the application, it was created using the Bootstrap v3 front-end framework using JavaScript, and the back-end part of the application was made using the AngularJS v1 framework, NodeJS and a script written in Perl, responsible for data consolidation and connecting them with the appropriate view elements .

In addition, the following technologies were used to create the application:

- Node v.0.12.7
- Node Express 4.13.0
- Node-ssh-exec 0.1.1

Regarding the application directory structure, the root of the *Virtual Dashboard project* has the following structure:

Virtual Dashboard

```
├── .bowerrc
├── .gitignore
├── bower.json
├── package.json
├── server.json
├── app /
│   ├── management.js
│   ├── pc-list.js
│   └── ssh.js
├── public /
│   ├── app /
│   │   ├── controllers /
│   │   │   ├── dashboardCtrl.js
│   │   │   └── mainController.js
│   │   ├── services/
│   │   │   ├── computersService.js
│   │   │   └── HttpService.js
│   │   ├── views/
│   │   │   ├── index.html
│   │   │   └── new_index.html
│   │   ├── app.js
│   │   └── app.routes.js
│   ├── assets/
│   │   ├── css/
│   │   │   └── style.css
│   ├── vendors/
│   │   ├── angular/
│   │   │   ├── .bower.json
│   │   │   ├── angular.js
│   │   │   ├── angular.min.js
│   │   │   ├── angular.min.js.gz
│   │   │   ├── angular.min.js.map
│   │   │   ├── angular-csp.css
│   │   │   ├── bower.json
│   │   │   ├── index.js
│   │   │   └── package.json
│   │   └── angular-route/
│   │       ├── .bower.json
│   │       └── angular-route.js
```

```
angular-route.min.js
angular-route.min.js.map
bower.json
index.js
package.json
```

```
└─bootstrap/
  └─dist /
    └─css /
      bootstrap.css
      bootstrap.css.map
      bootstrap.min.css
      bootstrap-theme.css
      bootstrap-theme.css.map
      bootstrap-theme.min.css
    └─fonts /
      glyphsicons-halflings-regular.eot
      glyphsicons-halflings-regular.svg
      glyphsicons-halflings-regular.ttf
      glyphsicons-halflings-regular.woff
      glyphsicons-halflings-regular.woff2
    └─js /
      bootstrap.js
      bootstrap.min.js
      npm.js
  └─fonts /
    glyphsicons-halflings-regular.eot
    glyphsicons-halflings-regular.svg
    glyphsicons-halflings-regular.ttf
    glyphsicons-halflings-regular.woff
    glyphsicons-halflings-regular.woff2
  └─grunt /
    bs-commonjs-generator.js
    bs-glympicons-generator.js
    bs-lessdoc-parser.js
    bs-raw-files-generator.js
    configBridge.js
    sauce_browser.yml
  └─js /
    affix.js
    alert.js
    button.js
    carousel.js
    collapse.js
    dropdown.js
    modal.js
    popover.js
    scrollspy.js
    tab.js
    sauce_browser.yml
  └─less /
    alerts.less
    badges.less
    bootstrap.less
    breadcrumbs.less
    button-groups.less
    buttons.less
    carousel.less
    close.less
```

```
code.less
component-animations.less
dropdowns.less
forms.less
glyphicons.less
grid.less
input-groups.less
jumbotron.less
labels.less
list-group.less
media.less
mixins.less
modals.less
navbar.less
navs.less
normalize.less
pager.less
pagination.less
panels.less
popovers.less
print.less
progress-bars.less
responsive-embed.less
responsive-utilities.less
scaffolding.less
tables.less
theme.less
thumbnails.less
tooltip.less
type.less
utilities.less
variables.less
wells.less
└── Gruntfile.js
    package.js
    package.json
```

The most important files responsible for the correct operation of the application can be divided into:

API :

- **app / management.js** - contains methods for managing virtual machines;
 - shutdown - shutdown ;
 - getVirts - download a list of machines ;
 - findComputerbyIP - based on the list of computers, get their IP;
- **app / pc-list.js** - hard-coded computer IP list;
- **app / ssh.js** - ssh connection authentication.

AngularJS :

- **/public/app/app.js** - module declaration;
- **/public/app/app.routes.js** - page routing declaration;
- **/public/app/controllers/dashboardCtrl.js** - home screen controller;
- **/ public / app / services / computersService.js** - service to be downloaded from the api list of computers;
- **/ public / app / views / index.html** - **HTML page**, *Computer Management* tab ;
- **/ public / app / views / index_new.html** - **HTML page**, *Virtual Management* tab .

5. Project implementation

5.1. Host configuration

The chapter below describes how to configure the physical machines hosting virtual machines. The installation of the required components and the specification of the network configuration were described, in this case the network bridge was used.

5.1.1. Preparation of the environment and installation of additional tools

Before starting the creation of virtual machines, you should check whether the physical machine supports virtualization. This verification can be done using the following command:

```
egrep '^ flags. * (vmx | svm)' / proc / cpuinfo
```

In the event that the computer has such support, the result should be similar to the following:

```
flags: fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca
cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe nx
lm constant_tsc arch_perfmon pebs bts pni dtes64 monitor pts_cpl
xf3 vm3 est cd lx lm3
flags: fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca
cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe nx
lm constant_tsc arch_perfmon pebs bts pni dtes64 monitor pts_cpl
xf3 vm3 est cd lx lm3
```

When we are sure that the machine has such support, but the command did not print any flags after execution, check if virtualization is not turned off in the BIOS.

The next step is to install the appropriate packages and tools from the distribution.

To do this, we execute the command:

```
sudo apt-get install qemu-kvm libvirt-bin bridge-utils virt-  
manager
```

After they are successfully installed, the user should be added to the libvirtd and kvm groups, to do this, execute the command:

```
sudo adduser [user_id] libvirtd  
sudo adduser [user_id] kvm
```

and run daemon libvirt and add it to the autostart:

```
sudo service libvirt-bin start  
sudo update-rc.d libvirt-bin defaults
```

5.1.2. Create a network bridge

The network bridge is created on the host with the following command:

```
sudo brctl addbr br0
```

Then we proceed to its configuration so that it is launched at system startup. To do this, disable Network Manager and edit the file named / **etc** / **network** / **interfaces** accordingly .

In the case of network configuration in the project, i.e. with the use of DHCP, the file after editing should look like this:

```
auto lo
iface lo inet loopback

iface eth0 inet manual

iface br0 inet dhcp
    bridge_ports eth0
```

As for static configuration , the file would look like this:

```
auto lo
iface lo inet loopback

iface eth0 inet manual

auto br0
iface br0 inet static
bridge_ports eth0
address 153.65.206.15
network 153.65.206.0
netmask 255.255.255.0
broadcast 153.65.206.255
gateway 153.65.206.1
dns-nameservers 10.10.24.5 8.8.8.8
dns-search example.com
    bridge_ports eth0
```

After configuring the bridge, turn off the NetworkManager and restart the network.
We issue commands in sequence:

```
sudo service network-manager stop
sudo update-rc.d network-manager remove
sudo service networking start
```

```
sudo update-rc.d networking defaults
```

The correctness of the bridge configuration can be checked using:

```
sudo brctl show
```

The command should return the following response:

```
bridge name bridge id STP enabled interfaces  
br0 8000.0200c0a80091 no eth0
```

5.2. Guest configuration

The following chapter describes how host virtual machines are built and what tools you can use to manage these machines.

At the beginning of the project, three virtual machines, named virt1, virt2, and virt3, were created on each of the two available physical machines (comp1 and comp2). As we delved into the topic of virtualization and discovered the complexities of it, it became necessary to create more machines in order to be able to save the configurations used to test the new properties on them. So there was the problem of storing images of new machines and the problem of quick installation of the same images. Therefore, it was decided to expand the pool of available machines, using the QEMU property, which is the derivative of images.

Initial Virtual Machines; virt1, virt2 and virt3; they have therefore become *base images* with a pure operating system. They were saved in the qcow2 format.

From each base image, an image was created derived from it using the `qemu-img create` command. An example of this command for a `virt2` machine:

```
qemu-img create -b /etc/libvirt/qemu/virt2.qcow2 \  
-f qcow2 / etc / libvirt / qemu / virt2-s0 . qcow2  
Formatting '/etc/libvirt/qemu/virt2-s0.qcow2', fmt = qcow2 size =  
5368709120 backing_file = '/ etc / libvirt / qemu / virt2.qcow2'  
encryption = off cluster_size = 65536 lazy_refcounts = off
```

The created machine `virt2-s0.qcow2` has the same parameters as the base machine. The properties of the base machine can be checked with the command:

```
qemu-img info virt2.qcow2  
image: virt2.qcow2  
file format: qcow2  
virtual size: 5.0G (5368709120 bytes)  
disk size: 5.0G  
cluster_size: 65536
```

We can also check the condition of the derived image:

```
qemu-img info virt2-s 0.qcow 2  
image: /etc/libvirt/qemu/virt2-s0.qcow2  
file format: qcow2  
virtual size: 5.0G (5368709120 bytes)  
disk size: 196K  
cluster_size: 65536
```

```
backing file: /etc/libvirt/qemu/virt2.qcow2
```

Then, to start a derived machine, first copy the base machine's XML configuration file, edit it, changing the name, uuid, disk path, MAC address, and then use the `virsh` command to define and start the machine:

```
virsh define virt2-s0.xml
virsh start virt2-s0 --console
virsh list
Id Name State
-----
9 virt2-s0 running
```

After starting, we can check the disk size of the newly created machine:

```
ls -lash virt2-s0.qcow2
14M -rw-y - y--. 1 root root 14M Oct 4 06:30 virt2-s0.qcow2
```

The size is only 14Mb, which means that only changes compared to the base machine will be written to the new disk.

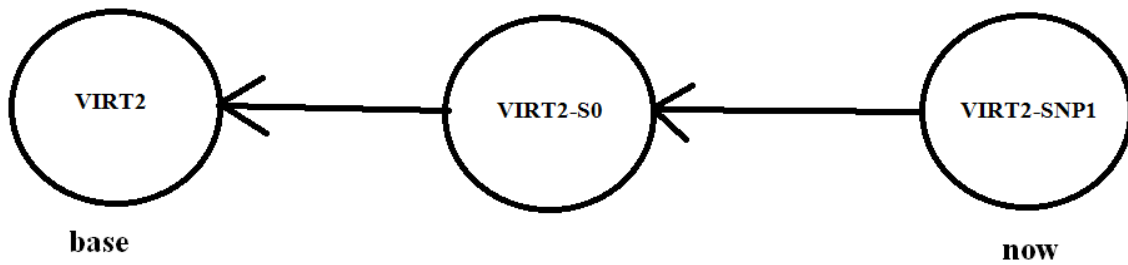
As a result, the so-called *delta image was created* , storing only changes in relation to the image on which it is based. This size will increase as you save new changes to it.

The `virt2.qcow2` image is our base image. From now on, when testing new solutions, we never start this machine again, but always run `virt2-s0.qcow2` as the starting machine.

If we would like to make any changes to the currently used image `virt2-s0.qcow2`, we save these changes by creating a snapshot of the image used, the command: **`qemu-img snapshot -c <snapshot-name> <imagename>`**.

After its creation, the dependence of our images will look like this:

Fig. 13. The relationship of derived images in QEMU



Source: own study

The virt2 image is the base for the virt2-s0 image, and the virt2-s0 image is the base for virt2-snp1. The current image is virt2-snp1, and each next change we want to make to the current image, we will need to create a new snapshot based on the current image. The new image will store the differences from the previous image.

We can use the qemu-img and virsh tools to manage snapshots.

The most important operations of these tools are presented in the table below:

Table 4. Shared image management with qemu-img and virsh.

QEMU-IMG		VIRSH	
snapshot	creating / managing the so-called <i>offline disk snapshots</i>	snapshot-create-as	creating the so-called <i>internal / external snapshots</i>
create	creating the so-called <i>offline external snapshots</i>	blockcopy	copying a chain of pictures
commit	applying changes from the earliest derived image to the base image	blockcommit	merge the chain from the earliest changes to the base image
rebase	copying the contents of the base image to a derived image	blockpull	merge the chain from the base image to the derived image

Source: own study

5.2.1. Create a virtual machine

A virtual machine on the host operating system can also be created using Virtual Manager and from the command line using libvirt and qemu-img create.

In both cases, configuration should begin by downloading the system image and copying it to the appropriate folder on the host, usually / **var / lib / libvirt / images /**.

Next, we add a virtual disk on the host:

```
qemu-img create -f qcow2 -o preallocation = metadata  
/var/lib/libvirt/images/virt1.qcow2 10G
```

This command creates an empty qcow2 image disk of 10G capacity. The advantage of using the .qcow2 format is that the disk is not created at full size initially, but grows as the data is retrieved.

Once we have the disk, we install the virtual machine with the command below:

```
virt-install -n virt1 --virt-type kvm --description "Slitaz40" --  
ram = 1536 --vcpus = 1 --cpu host --os-variant = ubuntuoneiric --  
accelerate -hvm --network bridge: br 0, model = virtio --graphics  
vnc, listen = 0.0.0.0 --noautoconsole -disk path = / var / lib /  
libvirt / images / virt1.qcow2, format = qcow2, bus = virtio,  
cache = none --cdrom / var / lib / libvirt / iso / slt40.iso
```

Virt-install is the libvirt API tool for creating a virtual machine.

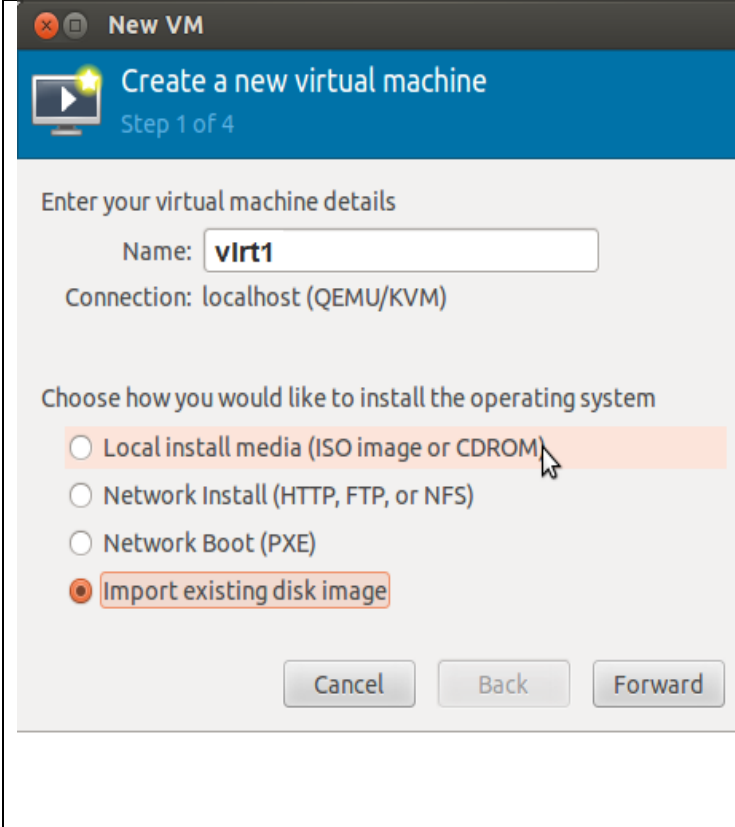
This tool takes the following parameters:

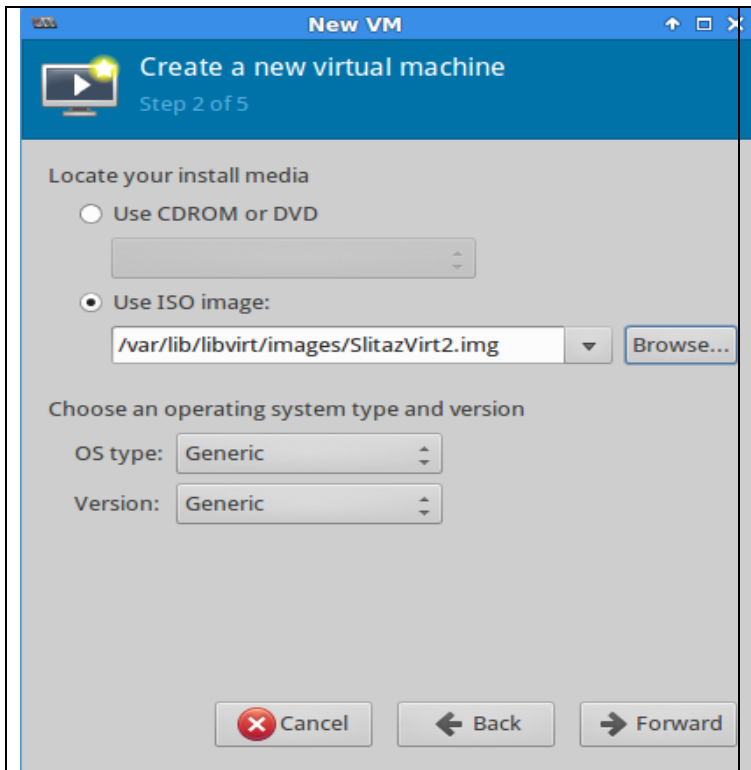
- r** - characterizes the RAM size (in MB);
- accelerate** -**indicates** to use *hardware acceleration* ;
- n**- defines the name of the virtual machine;
- f**- indicates the full path to the machine's disk image;
- cdrom** - is the path to the ISO (ISO image name) that you downloaded. This parameter is only needed during the installation, it can be removed after installation;
- vcpus** = **N** - specifies an SMP guest with N virtual CPUs;
- description** - points to a virtual machine description that will be placed in its XML file;
- l** - used when installing the machine from an image to a URL;
- disk** - specifies machine disk format, e.g.: qcow2 or raw;
- **ac97 soundw** - create a guest with AC'97 virtual sound card; without this option, no sound card will be assigned

Another libvirt tool that we can use to create virtual machines is **virt-image** . This command creates a new machine based on its configuration saved in the XML file.

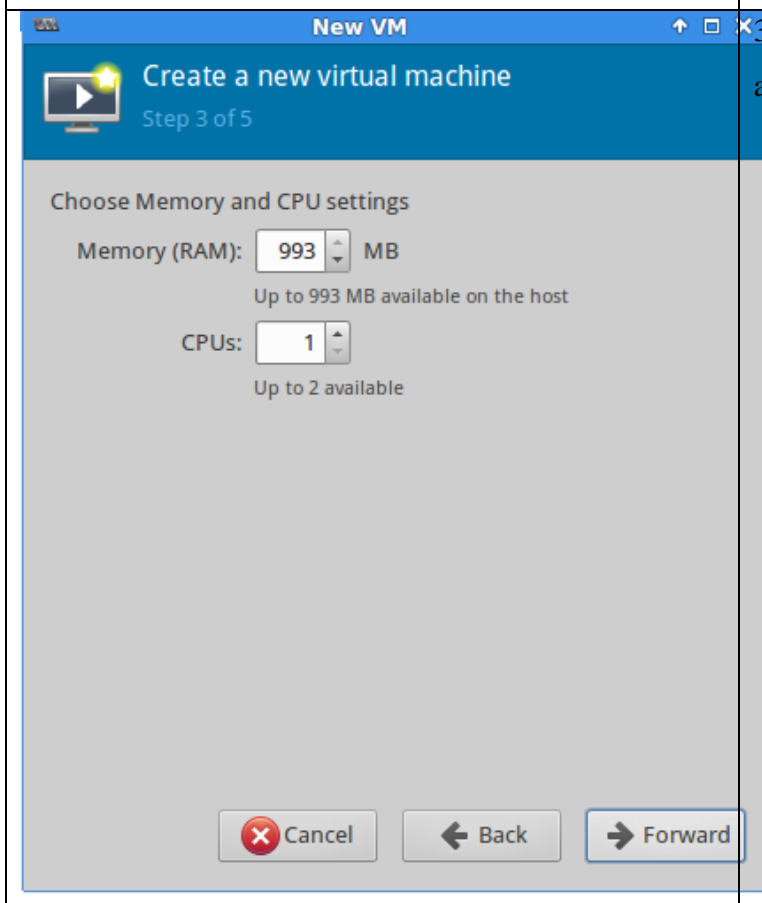
When it comes to creating virtual machines with Virtual Manager, this method is much easier because the program has a graphical interface. Creating a virtual machine with the use of Virtual Manager is shown in the screenshots below:

Fig 14. Creating a Virtual Machine Using Virtual Manager






	<p>1. We set the name of the VM and the location of the system image (choose <i>Local Install Media</i>).</p>
--	---

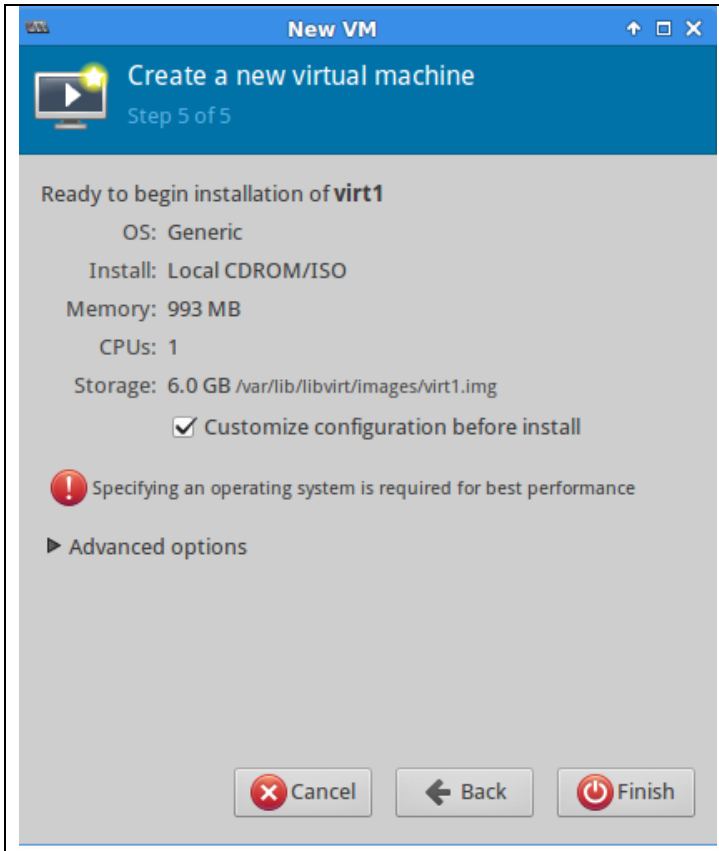
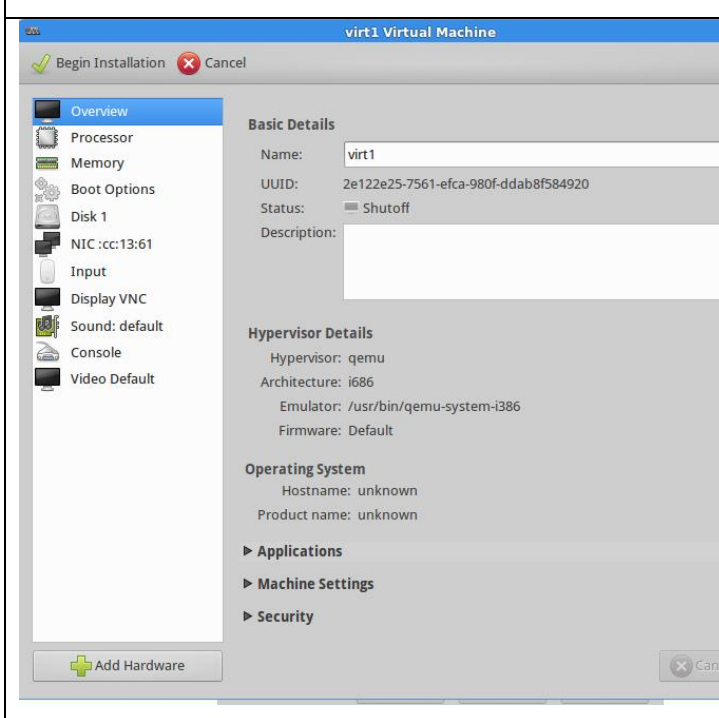


2. We indicate the location of the operating system that we want to install on the machine.



3. We set the size of RAM and the amount of CPU.

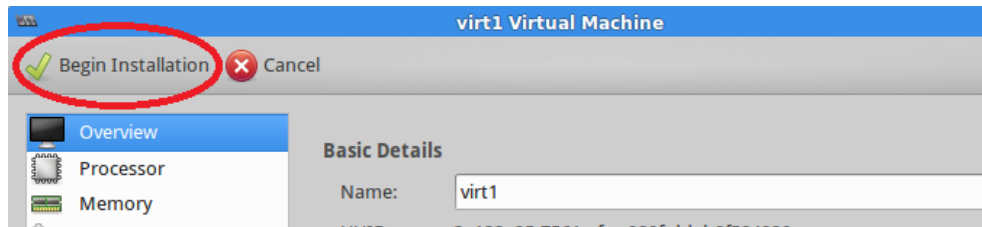
New VM	
<div data-bbox="277 241 984 346"> Create a new virtual machine Step 4 of 5</div> <div data-bbox="302 373 984 703"><p><input checked="" type="checkbox"/> Enable storage for this virtual machine</p><p><input type="radio"/> Create a disk image on the computer's hard drive</p><div data-bbox="370 464 516 506"><input type="text" value="6.0"/> GB</div><p>19.0 Gb available in the default location</p><p><input checked="" type="checkbox"/> Allocate entire disk now </p><p><input type="radio"/> Select managed or other existing storage</p><div data-bbox="331 653 984 695"><input type="button" value="Browse..."/> <input type="text"/></div></div> <div data-bbox="467 989 984 1041"><div> Cancel</div><div> Back</div><div> Forward</div></div>	<p>4. Determine the size of the virtual machine disk.</p>

	<p>5. We go to the advanced configuration of the machine.</p>
	<p>6. We set the detailed parameters of the virtual machine hardware.</p>

Source: own study

After selecting all the parameters of the virtual machine, the final step is to start installing the system. To do this, click the ***Begin Installation button*** , which is in the top left log of the configurator:

Fig. 15 Installing a virtual machine with the help of Virtual Manager



Source: own study

Installation is automatic, without the need for user intervention.

5.2.2. Guest virtual machine management

Besides Virtal Manager, many other tools can be used to manage the guest virtual machine. Currently, the most popular among them are:

- › Libvirt API,
- › QEMU,
- › Libguestfs,
- › Virt-v2v,
- › SVirt,
- › Virt-*,
- › Boxgrinder,
- › Spice.

libvirt library , written in C, is an interesting and complex project that allows you to manage virtual machines. It allows you to control processes such as: starting a virtual machine, cloning, creating network interfaces, increasing RAM, CPU.

An additional advantage of the library is the fact that its API is open, and thus provides a uniform interface for programmers of various languages, thus offering the

opportunity to contribute to the development of API. The languages currently supported are:

- ♦ C # - <http://libvirt.org/csharp.html>
- ♦ Java - <http://libvirt.org/java.html>
- ♦ OCaml - <http://libvirt.org/ocaml/>
- ♦ Perl - <http://search.cpan.org/dist/Sys-Virt/>
- ♦ PHP - <http://libvirt.org/php>
- ♦ Python - <http://libvirt.org/python.html>
- ♦ Ruby - <http://libvirt.org/ruby/>

The downside to using libvirt is that the library does not support all types of monitors. It currently offers support for the following types:

- Hypervisor Xen on Linux and Solaris hosts;
- QEMU;
- KVM ;
- LXC;
- OpenVZ;
- VirtualBox ;
- VMware ESX and GSX;
- VMware Workstation / Player;
- User Mode Linux paravirtualized kernel ;
- Storage on IDE / SCSI / USB, FibreChannel, LVM, iSCSI and NFS disks .

Another disadvantage is the lack of automation; management can only be done from one node and not from many as in the case of load balancing.

The libvirt utilities are mostly command-line commands.

One of the libvirt commands is the **virsh** command , which is able to manage the configuration of the following virtual machine parameters:

→ increase / decrease virtual machine memory (RAM)

To manage the memory of a virtual machine, we can use two commands of the virsh command, which are presented in the table below:

Table 5. Virsh commands to edit VM memory state

command to edit the state of the VM memory	command function	command to return the memory status of a given VM
• virsh edit <VM_name, id_VM>	allows you to edit the machine's XML configuration file	• virsh dominfo <VM_name, id_VM> grep 'memory' Max memory: 4194304 KiB Used memory: 2097152 KiB
• virsh setmem <VM_name, id_VM> <count_in_kilobytes>	defines the size of the memory used	
• virsh setmaxmem <VM_name, id_VM> <count_in_kilobytes>	sets the maximum available memory limit for the supervisor	

Source: own study

To increase the memory of a specific virtual machine, please do the following:

- shut down the virtual machine with the command **virsh shutdown** <VM_name>:

```
virsh shutdown virt1
Domain virt1 is being shutdown
```

- edit virtual machine parameters with the **virsh edit** <VM_name> command:

```
virsh edit virt1
```

- in the displayed XML configuration file, find the entry referring to the virtual machine memory size and edit it freely:

```
<memory unit = 'KiB'> 4194304 </ memory>
```

- save the changes made, again using **virsh edit** <VM_name>:

```
virsh edit virt1
Domain virt1 XML configuration edited.
```

- all that's finished is restarting the virtual machine with the **virsh reboot** <VM_name> command:

```
virsh restart virt1
```

→ increase / decrease the size of the virtual machine processor (CPU)

The following commands are responsible for managing the virtual machine processor:

Table 6. Virsh commands to edit VM CPU size

command to edit the size of the VM processor	command function	command to return the CPU status of a given VM
• virsh edit <VM_name, id_VM>	allows you to edit the machine's XML configuration file	• virsh domininfo <VM_name, id_VM> grep -i cpu CPU (s): 4 CPU time: 22.1s • virsh vcpuinfo <VM_name, id_VM> VCPU: 1 CPU: 4
• virsh vcpus <VM_name, id_VM> <number_vcpus>	defines the amount of vcpus for a given machine	

		State: running CPU time: 22192.9s CPU Affinity: yyyyyyyy
--	--	--

Source: own study

In the case of editing the CPU of a virtual machine by changing the configuration of the XML file with the **virsh edit command**, we proceed in the same way as in the previous example. The only change is the fact that we are editing the fragment of the file responsible for the vCPU configuration:

```
<vcpu placement = 'static'> 4 </vcpu>
```

→ modifying the disk space of a virtual machine

To add a new virtual disk to the machine, you must first create such a disk and then add it to the virtual machine's disk space.

A new drive is usually created in a folder **/ var / lib / libvirt / images /** using the command **qemu-img create** :

```
cd / var / lib / libvirt / images /
qemu-img create -f raw virt1-disk2.img 7G
Formatting 'virt1-disk2.img', fmt = raw size = 7516192768
```

Then add it to an existing machine using the **virsh attach-disk command** :

```
virsh attach-disk virt1 --source /var/lib/libvirt/images/virt1-
disk2.img --target / dev / vdb --persistent
Disk attached successfully
```

The above command uses the following parameters:

- virt1** = **<vm_name>** - virtual machine name;
- source** - it is after it full path to the attached disk;
- target-** place where we want to mount this disk;
- persistent-** describes how the disk is to be added to the virtual machine, it can also take the *cache parameter* .

In order to remove a disk from a virtual machine, we issue the **virsh detach-disk** command :

```
virsh detach-disk virt1 vdb  
Disk detached successfully
```

You can verify the correctness of the configuration using:

```
fdisk -l | grep 'vd'
```

→ adding a virtual CD-ROM / DVD to a virtual machine

We use the **virsh attach-disk** command with the appropriate parameters:

```
virsh attach-disk <VM_name> < source> <target> --driver file --  
type cdrom --mode readonly
```

→ adding devices defined in the XML file to the virtual machine

With the virsh command, we can also add devices defined by the XML configuration file to the guest virtual machine.

The following example shows adding an ISO file as HDC to a running virt1 guest machine. For this purpose, the device was first defined with the XML configuration file:

```
cat virt1iso.xml  
<disk type = "file" device = "disk">  
  <driver name = "file" />  
  <source file = " / var / lib / libvirt / virt1drive. iso" />  
  <target dev = "hdc" />  
  <readonly />  
</disk>
```

Then , with the command **virsh attach-device** we add the created file to guest comp1:

```
virsh attach-device virt 1 virt1iso.xml
```

→ creating snapshots of a virtual machine

As mentioned before, VM snapshots reflect the entire environment in which the virtual machine is running - they include the processor (CPU) state, memory (RAM) state, emulated devices, and all writable disks states.

There are several types of virtual machine snapshots:

1.internal _____ - a single qcow2 image stores both the reference image and the changes made to it. It is a single image that contains all information about the virtual machine at any given time. These types of snapshots take quite a long time (especially when the virtual machine is turned on) and only support the qcow2 file format, so they are not updated by QEMU.

There are two subgroups of internal snapshots:

► **internal** ' *disk snapshot* ' - {live / offline}

It reflects the state of the virtual disk at the moment. Both the output image and the changes made are stored in the same qcow2 file. The screenshot can be made with the guest machine turned on or off.

► **internal dump type 'checkpoint'** - (*internal 'system checkpoint' snapshot*) - {live}

It is executed when the virtual machine is turned on. The state of the virtual machine and disks are saved, while the state of the virtual machine consists of the state of its memory and emulated devices (without the state of its disk).

Before creating a virtual machine dump, you should prepare an xml file for this dump:

```
cat /var/tmp/virt1-snp1.xml
<domainsnapshot>
<name> virt1-snp1 </name>
<description> test snapshot virt1 </description>
</domainsnapshot>
```

Then you can proceed to take a screenshot when the machine is turned on:

```
virsh snapshot-create virt1 /var/tmp/virt1-snp1.xml
Domain snapshot virt1-snp1 pki created from '/var/tmp/snap1-
komp1.xml'
```

At the time of the snapshot, the guest machine will be paused and restarted after the process is complete.

To check if the snapshot was taken correctly, we check if it appears on the list of available snapshots for a given machine:

```
virsh snapshot-list virt1
Name Creation Time State
-----
virt1-snp1pki 2011-10-04 19:04:00 +0530 running
```

For this purpose, we can also use the **qemu-img info** command , which will also display information about the resulting screenshots for a given guest:

```
qemu-img info virt1
image: /etc/libvirt/qemu/virt1.qcow2
file format: qcow2
virtual size: 8.0G (8589934592 bytes)
disk size: 3.2G
cluster_size: 65536
Snapshot list:
ID TAG VM SIZE DATE VM CLOCK
1 virt1-snp1 1.7G 2011-10-04 19:04:00 32: 06: 34.974
2 1317757628 0 2011-10-05 01:17:08 00: 00: 00.000
```

To retrieve a snapshot taken, use the command **virsh snapshot-revert <domain snapshotname>**.

2. external - when the *template* file is in the "read-only" format and the new generated file is a delta of changes. Such a snapshot only stores changes from the reference image. This type is especially useful for backups. It does not require turning off the guest machine. The external dumps support all kinds of disk formats - raw, qcow2, etc.

As with internal discharges, we also distinguish between two sub-types:

► ***external 'disk snapshot' - {live / offline}***

The disk snapshot is saved in one file and the changes are made in a new qcow2.

► ***external 'system checkpoint' snapshot - {live}***

A virtual machine's disk image is saved in one file, and its RAM and the state of emulated devices in another.

3 . virtual machine state - occurs when we save the state of the virtual machine at a given moment to a file. Only the VM state is saved, not the disk state.

The virtual machine configuration can be saved by copying its XML configuration file. This is done with the **virsh dumpxml** command :

```
virsh dumpxml virt1> VIRT1.xml
```

```
ls
virt1.xml
VIRT1.xml
```

From the resulting configuration file (here VIRT1) you can create an image of an already existing machine:

```
virsh create VIRT1.xml
```

The full list of available virsh command options used in snapshots is presented in the table below:

Table 6. Virsh commands for snapshots.

recommendation	description
snapshot-create <domain> <xml_file>	creates a snapshot
snapshot-current <domain>	shows the XML file of the current snapshot for a given virtual machine
snapshot-delete	removes the snapshot
snapshot-dumpxml <domain> <snapshot_name>	shows the XML snapshot file with the given name for the selected domain
snapshot-list <domain>	shows available snapshots
snapshot-revert <domain> <snapshot_name>	restores the domain to the given snapshot
snapshot-create-as	creates a snapshot with the specified parameters
snapshot-parent <snapshot_name>	shows the name of the snapshot master machine

Source: own study.

virsh commands :

Table 7. Virsh commands for virtualization.

recommendation	role
Domain Management	
virsh list -- all	list of all virtual machines with status

virsh list -- inactive	list of inactive virtual machines
virsh start <vmname, vmid>	start a virtual machine
virsh shutdown <vmname, vmid>	virtual machine stop, shutdown
virsh reboot <vmname, vmid>	restart the virtual machine
virsh destroy <VMName ,VMID>	forced stop of the virtual machine (force stop)
virsh suspend <vmname, vmid>	stopping (pausing) the virtual machine
virsh resume <vmname, vmid>	virtual machine resumption
virsh dumpxml <vmname, vmid>	displaying the virtual machine configuration file
virsh define <vmname, vmid>	creates an xml configuration file for a virtual machine
virsh save <VM_name, VMID> <filename>	saving the state of the virtual machine to a file
virsh restore <filename>	restore a previously saved virtual machine
Domain Monitoring	
dombldinfo	information about the size of domain block devices
dombldlist	domain block list
dombldstat	displays statistics for domain block devices
domif-getlink	displays the state of the virtual interface
dominfo	displays information about domain
dommemstat	displays memory statistics for a domain
domstate	displays the status of the domain
letter	displays a list of domains
Host and Hypervisor	
hostname	displays the name of the hypervisor
nodecpustats	displays the cpu statistics for a given node
nodememstats	displays memory statistics for a given node

sysinfo	displays information about the system hypervisor
nodeinfo	displays information about the node
Interface	
iface-start	starts the host's physical interface (activates it / "if-up")
iface-destroy	deactivates physical host interface (deactivates it / "if-down")
iface-undefine	removes the host's physical interface (removes it from configuration)
iface-list	displays a list of physical interfaces
iface-bridge	creates a bridge and attaches it to an existing network device
iface-unbridge	removes the sternum
Networking	
net-create	creating a virtual interface
net-define	creating a given virtual network interface
net-destroy	erasing the virtual interface
net-dumpxml	displaying a specific virtual network configuration file
net-info	display information about a given virtual network
net-list --all	display information about all virtual networks
net-start	activation of a given virtual network interface
net-undefine	deactivation of a given virtual network interface

Besides the virsh command, we can also use other libvirt tools to manage the virtual machine. The most commonly used are listed below:

► **virt-clone, virt auto-clone, virt-clone-prompt**

It is a virtual machine image cloning tool. Virt-clone copies the virtual disk image of the guest machine and uses it to define a new one with the same resources as the reference machine. Only the following attributes that define the virtual machine are changed:

- computer name;
- path to the disk;
- machine IP;
- UUID;
- MAC address.

► **virt-top**

The tool collects data on virtual machines on a given guest, such as status, CPU usage, RAM, and presents it in a descending form, starting with those that burden the host system the most.

► **virt-what;**

► **virt-df**

Shows information about virtual machine disk usage:

```
virt-df -a virt1.img
Filesystem 1K-blocks Used Available Use%
virt1.img: / dev / sda1 99 099 1551 92432 2%
```

As for QEMU, in addition to the qemu-img tool already mentioned, the following commands are also available:

Table 8. The most important QEMU commands

recommendation	role
<u>QEMU-IMG:</u>	
-check	checks the image and if it finds errors in it, it displays them, if there are no errors, it displays nothing
-convert	converts the format of the images
-create	creates an image
-snapshot	creates the so-called <i>offline disk snapshot</i>
-info	displays information about the image of the machine in question
<u>QEMU-INFO:</u>	
-block	displays information about block devices like cdrom, hard disk
-blockstats	displays statistics on block devices
-cpus	displays information about the CPU
-cpustats	displays CPU usage statistics
-mem	displays information about the memory of virtual machines
-mmtree	displays the memory tree
-tlb	displays information about virtual and physical memory on a given host

-usb	lists usb devices on virtual USB port
-usbhost	displays a list of USB devices on a specific virtual machine
-history	displays the history of console commands
-irq	displays statistics on interrupts
-network	displays information about VLANs
-pci	displays the PCI devices that are emulated
-snapshots	displays available snapshots
-qtree	displays the device tree
-roms	displays information about available rom devices
-migrate	displays information about the migration status
-snapshot	<p>snapshots record information about the entire virtual machine - CPU state, RAM, state of its devices, disk content. To use snapshots, you must have at least one non-removable and writable block device that uses the qcow2 format as a disk image.</p>

Source: own study

5.3. Configuration of the management application

The chapter below describes in detail how the web application for managing virtual resources on hosts has been configured. Code implementation examples are given and the role of the most important configuration files is explained.

5.3.1. User authentication

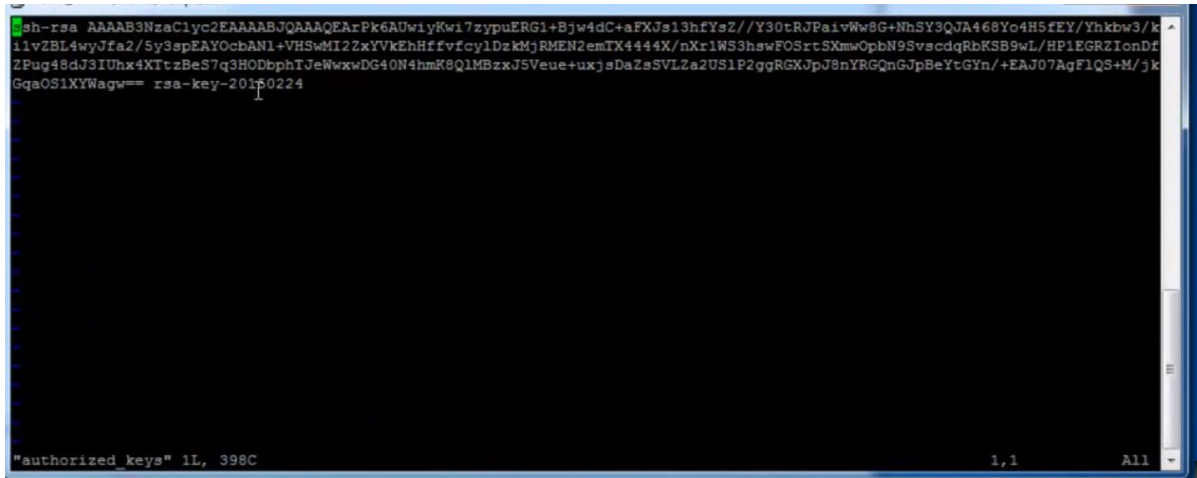
The created application runs on a physical computer with the Windows7 operating system. In principle, only one user has access to the application, who is responsible for managing the built infrastructure. For this reason, the application works only on one computer of a specific user (on localhost: 7070) and as only one person will have access to it, there is no user identification module or database for storing their data. If, however, there was a need to expose the application to the outside world, then the most sensible method of user authorization would be to implement a responsive Bootstrap login form and connect it to the MySQL database.

As for the authorization of connections of the machine running the application (comp3) with hosts (com1, comp2), it is done via SSH using a public key. It is a passwordless login method that works according to the *challenge-response model*. During the authorization process, the server with the public key creates the so-called challenge, which only the user with the private key of the same pair is able to decode and thus provide the correct *response*.

To configure the authorization with the use of SSH keys, on the side of the computer from which we connect remotely, i.e. in our case comp3, we generate a pair of public and private keys. Since the computer on which the application is running is running a Windows operating system that does not have an SSH server installed, two additional programs had to be installed to generate the keys - PuTTY and PuTTYgen.

Then the key pair was created with PuTTYgen. We save the keys in the following formats: private as `id_rsa`, public - `id_rsa.pub`. The resulting private key was encrypted with a password and saved on the computer's disk, while the public key was copied to the machines it connects to (com1 and comp2). On each Xubuntu machine, open the following **HOME / .ssh / authorized_keys file** and paste the contents of the public key copied from the PuTTYgen frame into it. This key will look like this:

Fig 16. HOME / .ssh / authorized_keys



Source: own study

Set the appropriate access to files with the following commands:

```
chmod 755 ~ / .ssh / authorized_keys
chmod 755 ~ / .ssh
```

In the next step, you need to modify the SSH- / **etc / ssh / sshd_config configuration file** to accept public key authorizations. The lines responsible for this are below:

```
# enable public-key authentication
RSAAuthentication no
PubkeyAuthentication yes
```

The file must also specify the path to the public key.

The created application references SSH in the file `/app/ssh.js`. This file contains the path to the private key saved on this computer and the user name.

5.3.2. Designing the front-end of the application

The front-end, i.e. the appearance of the application, was created using the Bootstrap framework, provided with appropriate additional libraries. The website design template can be found in the index.html (for the first tab) and newindex.html (for the next). The basic version of the template looks like this:

```
<! DOCTYPE html>
<html lang = "en">
<head>
<meta charset = "utf-8">
<meta http-equiv = "X-UA-Compatible" content = "IE = edge">
    <meta name = "viewport" content = "width = device-width,
initial-scale = 1">
<base href = "/" />
    <!-- The above 3 meta tags *must* come first in the head;
any other head content must come *after* these tags -->
    <title>Virt</title>
    <!-- Bootstrap -->
    <link
href="/vendors/bootstrap/dist/css/bootstrap.min.css"
rel="stylesheet">
    <link href="/vendors/bootstrap/dist/css/bootstrap-
theme.min.css" rel="stylesheet">
    <link href="/assets/css/style.css" rel="stylesheet">
    <!-- HTML5 shim and Respond.js for IE8 support of HTML5
elements and media queries -->
    <!-- WARNING: Respond.js doesn't work if you view the
page via file:// -->
    <!--[if lt IE 9]>
```

```

    <script
src="https://oss.maxcdn.com/html5shiv/3.7.2/html5shiv.min.js"
></script>

    <script
src="https://oss.maxcdn.com/respond/1.4.2/respond.min.js"></s
cript>

    <![endif]-->
</head>

```

...

//dołączone biblioteki

```

<!-- libs -->
<script src="/vendors/jquery/dist/jquery.min.js"></script>
<script
src="/vendors/bootstrap/dist/js/bootstrap.min.js"></script>
<!-- angular modules -->
<script src = "/ vendors / angular / angular.min.js">
</script>
<script src = "/ vendors / angular-route / angular-
route.min.js"> </script>
<!-- modules -->
<script src = "/ app / app.js"> </script>
<script src = "/ app / app.routes.js"> </script>
<script src = "/ app / controllers / mainController.js">
</script>
<script src = "/ app / services / HttpServices.js"> </script>
</body>

```

```
</html>
```

It is written in simple HTML, while the fragment **<meta name = "viewport" is** responsible for the scalability of the created website on mobile devices.

Before the last closing **<body>** tag, CSS and JS files from the downloaded library and a link to the jQuery library were attached to the template.

Then, a part responsible for the menu and navigation on the site was attached to the template. This snippet looks like this (in the case of index.html):

```
<nav class = "navbar navbar-inverse navbar-fixed-top">
<div class = "container">
<div class = "navbar-header">
<button type = "button" class = "navbar-toggle collapsed"
data-toggle = "collapse" data-target = "# navbar" aria-
expanded = "false" aria-controls = "navbar">
<span class = "sr-only"> Toggle navigation </span>
    <span class="icon-bar"></span>
    <span class="icon-bar"></span>
    <span class="icon-bar"></span>
</button>
    <a class="navbar-brand" href="#">Project name</a>
</div>
<div id="navbar" class="collapse navbar-collapse">
    <ul class="nav navbar-nav">
        <li class="active"><a
href="/">Dashboard</a></li>
    </ul>
</div><!--/.nav-collapse -->
</div>
```

```
</nav>
```

Angular.js is responsible for the dynamics of the website, which in its syntax has become an adaptation and extension of the HTML language. AngularJS promotes a declarative style of application writing (as opposed to imperative HTML). It is an example of an application development model called MVC, or *model-view-controller*. This pattern implies the division of the application structure into three modules:

- model - the part responsible for the data model - here it is a set of objects and primitives that can be referenced from the Scope object (\$ scope).
- view - responsible for the method of data presentation;
- controller - a module responsible for communication between the user and the system; accepts requests, passes processed data to the model and administers views - here it is the JavaScript class attached to the scope of a given view.

In this layout, the view updates the model that is changed in the controller immediately, and vice versa, when the controller refreshes the model, the view will update immediately in the view. This is called *two-way data-binding*.

The dynamics of the strings are governed by directives that can be used in several ways:

- as an element,
- as an attribute,
- as a CSS class,
- as a comment.

AngularJS has built-in directives that the application developer can use. The most important of them are :

- ng-app - declares an element as the main element of the application;
- ng-init - code execution at the beginning,
- ng-bind - automatically changes the text of an HTML element to the value of a given expression,
- ng-model - it is used for communication between the user and the model,

- ng-controller - defines the JavaScript controller class, which is to process the data returned to HTML,
- ng-show - when the value will be truly the container will be displayed (adds the ng-hide class - visibility through CSS),
- ng-if - removes or adds a given fragment, supports if expressions that allow showing a given element if the conditions are met,
- ng-repeat - iterates through the array,
- ng-list - when presenting an array as a string, treats that string as an array

Examples of Angular directives used in the code are provided in the table below:

Table 9. Examples of AngularJS directives in application code.

code snippet	name of the directive	role
<pre> <table class = "table table-striped"> <tr> <th> Machine IP </th> <th> VMname </th> <th>IP</th> <th>Mac Address</th> <th>State</th> <th>Status</th> <th>VM Disk Size</th> <th>VM Available Size</th> <th>Last login</th> <th>CPU</th> <th>VM Last change</th> <th>Actions</th> </tr> <tr ng-repeat="computer in dashboard.computers" ng-init="computerIndex = \$index"> <td>{{ computer.ip }}</td> <td> <select name="virtuals" class="form- control" ng-model="virtIndex" ng- change="dashboard.updateStatus(computerIndex,virtInd ex)"> <option value="">Please select -- </option> <option ng-repeat="virt in computer.virts.concat().sort()" ng- value="{{ \$index }}">{{ virt.name }}</option> </pre>	<ul style="list-style-type: none"> • ng-repeat • ng-init 	<p>- is responsible for iterating through the collection of items and displaying and / or filtering them</p> <p>- is responsible for examining data in a specific scope; in this case it aliases the special values of the ng-repeat directive; holds the index of the</p>

<pre> </select> </td> <td> {{computer.virts [computer.current_machine_index] .ip}} </td> <td> {{computer.virts [computer.current_machine_index] .mac}} </td> <td> </pre>	<ul style="list-style-type: none"> • ng-value 	<p>first index element in computerIndex</p> <p>- it is used when dynamically generating a list, in this case virtual machines</p> <p>- \$ index- is a service that is an object, what we define in this object will be immediately available for the view (directive).</p>
<pre> <div class = "row table-top"> <select ng-change = "dashboard.updateCurrentComputer (computerIndex)" class = "pull-left form-control virt- selector pull-left" name = "" ng-model = "computerIndex"> <option value="">Please select --</option> <option value="{{ \$index }}" ng- repeat="computer in dashboard.computers">{{computer.ip}}</option> </select> <a ng- click="dashboard.shutdown(dashboard.current_comput er.ip)" class="btn btn-primary">Turn off <a ng- click="dashboard.reboot(dashboard.current_computer.ip)" class="btn btn-primary">Reboot <a ng-click="dashboard.refreshData()" class="pull-right btn btn-primary">Refresh data </div> </pre>	<ul style="list-style-type: none"> • ng-change • ng-model 	<p>-directive called on select change, in this case loaded virtual machines are changed</p> <p>- assigns the contents of the field to the computerIndex variable, assigns the value of the</p>

		input to this model
--	--	------------------------

Source: own study

5.3.3. Designing the back-end of the application

The backend layer of the created application is NodeJS, a script written in Perl that runs on the side of a remote computer, and scripts that run on a remote computer.

AngularJS is responsible for communication with the remote computer, which connects to it via the SSH port and runs the script written in Perlcomm.pl. This script in turn collects data from the given computer and returns it to standard *output* .

The comm.pl script in some cases runs remotely auxiliary scripts written in the shell, which provide it with data about virtual machines; other times it issues commands by itself, using commands such as virsh, qemu-img info, etc.

Sample code fragments with references to the HTML page view of the application are presented in the table below:

		file from where it is downloaded.	
3	It displays the name of the physical computer from which the data is downloaded, the machine will not appear on the list if it is offline.	Computer names are found in the \ app \ pc-list.js file.	We refer to the names of computers from the pc-list.js file in the \ app \ management.js file, if the connection with a given computer is not possible, it will not be displayed on the list.
4	The name of the virtual machine available on the given physical machine, and disabled physical machines are also displayed.	This information is retrieved by a Perl script called comm.pl , which is run on a given physical machine. This script uses the virsh --all command to extract the data.	<p>A fragment of the comm.pl script that is responsible for obtaining the data:</p> <pre> sub getVirtualMachines { my @virt_lines = getDataLines('list --all'); my @virts; for my \$index (0..\$#virt_lines) { \$virts[\$index] = {name => \$virt_lines[\$index]->[1] status => \$virt_lines[\$index]->[2]}; } return @virts; } </pre> <p>getDataLines:</p> <pre> sub getDataLines { my \$command = shift; my \$common = 'virsh -r -c qemu:///system'; my \$virsh_stream = qx(\$common \$command); my @raw_lines = split('\n', \$virsh_stream); my @output_array; for my \$index (0..\$#raw_lines) { if (\$index > 1) { my @array = split ' ', \$raw_lines[\$index]; \$output_array[\$index-2] = \@array; } } return @output_array; } </pre> <p>The comm.pl script uses the command: virsh list --all</p> <p>which displays the names of all virtual machines on a given computer.</p> <p>Sample command output:</p>

			<pre> Id Name State ----- 1 virt1 running 2 virt2 running 3 virt3 running </pre> <p>When displaying the output, the Perl script only shows the second column, starting with the second line; a fragment of the getDataLines script is responsible for this.</p> <p>Sample output:</p> <pre> virt1 virt2 virt3 </pre> <p>You can achieve the same effect by using awk and sed:</p> <pre>virsh list --all awk '{print \$ 2}' sed 1d</pre>
5	Virtual Machine IP	The information is downloaded by the comm.pl script, which runs the getIPAddress.sh script on the host machine, and then reads its results.	<p>A fragment of the comm.pl script that is responsible for obtaining the data:</p> <pre> sub getVirtIP { my \$addresses = `./getIPAddress.sh`; my %data_lines = map { split ' ', \$_ } split ('\n', \$addresses); my @keys = keys %data_lines; for my \$virt (@virts) { if (\$virt->{mac} ~~ @keys) { \$virt->{ip} = \$data_lines{\$virt->{mac}}; } else { \$virt->{ip} = "IP missing"; } } } </pre> <p>The comm.pl script runs the getIPAddress.sh script, which obtains the IP addresses of the aforementioned virtual machines stored in the virts variable.</p> <p>The getIPAddress.sh script first obtains the MAC address of the virtual machine using libvirt, a code snippet:</p> <pre>MAC_ADDRESS = \$ (\$ virsh dumpxml "\$ VM_NAME" \$ grep "mac address" \$ sed "s /. * '\ (. * \)'. * / \ 1 / g")</pre> <p>and then refer to arp to look up the IP corresponding to the given MAC address, snippet of code:</p> <pre>\$ arp -an \$ grep "\$ MAC_ADDRESS"</pre>
6	MAC address of the given virtual machine		A fragment of the comm.pl script:

			<pre> sub getMac { my \$virt = shift; my \$command = 'dumpxml ' . \$virt; my @mac_lines = getDataLines(\$command); return @mac_lines[0]->[4]; } sub getMacs { for my \$virt (@virts) { \$virt->{mac} = getMac(\$virt->{name}); } } </pre> <p>Perl script uses the command: <code>virsh dumpxml <VM-NAME-HERE> grep -i '<mac'</code> which is performed for each of the virtual machines.</p> <p>Sample output of the virsh dumpxml virt1 command grep -i :</p> <pre><mac address = '52: 54: 00: 4c: 40: 1c ' /></pre>
7	state virtual machine: -running, -blocked, -paused, -shutdown, -dying, -crashed.	The information is downloaded from the comm.pl script	<p>script gets this information using the <code>virsh -all</code> command that is run on each host machine.</p> <p>If a virtual machine is configured so that if it is damaged, the system will be reinitialized, when the fault occurs, it falls into the so-called <i>endless loop</i> . This configuration entry can be found in the XML file of each machine, see the fragment: <code><on_crash> restart </on_crash></code></p> <p>In case the virtual machine has this option in the XML file, the virsh -all command in the State field has six possible answers.</p> <p>Alternatively, to know if the image of the machine is working; you can use the <code>qemu-img check</code> command.</p>
8	the size of the virtual machine memory currently in use	The information is downloaded from the comm.pl script	<p>script gets this information by running the appropriate script on each host machine.</p> <p>Command referenced by Perl script: <code>virsh dumpxml <VM-NAME-HERE> grep -i memo</code></p> <p>Sample command output for virt1:</p> <pre>< memory unit = 'KiB' > 16777216 </ memory > < currentMemory unit = 'KiB' > 1048576 </ currentMemory ></pre> <p>Alternatively, you can also refer to the virsh dominfo command. An example output of this command for machine virt1 looks like this:</p> <pre> Id : 1 Name : virt1 UUID: 4f610a1f-7539-47cf-8299-9534500b340d OS Type: hvm State: shut off CPU(s): 1 Max memory:16777216 kB Used memory:1048576 kB Persistent: yes Autostart: disable Managed save : no </pre>

9	maximum virtual machine memory size	The information is downloaded from the comm.pl script	It uses the same command as before.												
10	date of the last login to the virtual machine	The information is downloaded from the comm.pl script	The information comes from the virtual machine log, which is located in the /var/log/libvirt/qemu/virtual_machine.log folder												
11	date of the last change to the virtual machine image	The information is downloaded from the comm.pl script	<p>Generating this information is related to the mechanism of nested <i>images</i> offered by virtualization. In this case, the changes in the virtual machine image are the created snapshots, which are the so-called <i>delta images</i> , that is, they only store changes in relation to the base image.</p> <p>Therefore, the date of the last modification made to the base image will be the date when the last snapshot of that image was created.</p> <p>Information about available snapshots for a given virtual machine can be obtained with the command <code>virsh-snapshot –list <domain></code></p> <p>Sample output:</p> <table><tr><th>Name</th><th>Creation Time</th><th>State</th></tr><tr><td>-----</td><td>-----</td><td>-----</td></tr><tr><td>1295973577</td><td>2011-01-25 17:39:37 +0100</td><td>running</td></tr><tr><td>1295978837</td><td>2011-01-25 19:07:17 +0100</td><td>running</td></tr></table>	Name	Creation Time	State	-----	-----	-----	1295973577	2011-01-25 17:39:37 +0100	running	1295978837	2011-01-25 19:07:17 +0100	running
Name	Creation Time	State													
-----	-----	-----													
1295973577	2011-01-25 17:39:37 +0100	running													
1295978837	2011-01-25 19:07:17 +0100	running													

The application uses the Node.js environment for communication with physical machines via the node-ssh-exec module.

Sample request to a physical machine (shutdown):

```
exports.shutdown = function(req, res) {

    var comp = findComputerbyIP(req.params.ip);

    var config = {
        host: req.params.ip,
        username: comp.username,
        password: comp.password
    };
};
```

```
    var command = 'echo ' + comp.password + ' | sudo -S shutdown -  
h now';  
  
    exec(config, command, function(error, response) {  
        if (error) {  
            console.log(error);  
        }  
        res.status (200) .json (response);  
    });  
}
```

The application has an API issued for use by the client (website). Each application request is intercepted and the appropriate request is performed based on its type, address and parameters.

5.3.4. Launching the application and reading the results

For the application to work, first update the list of physical computers in the `/app/pc-list.js` file and edit the `/app/ssh.js` file, which contains the SSH connection configuration. The next step is to place the public key on each remote computer to which we want to connect and check if the SSH port on the machine is properly configured.

After verifying the correctness of the remote connection, it will be possible to start the application. To do this, go to the main folder and run the **npm install command from it** , which will install the application with its extensions.

Next, we start the server with the command: **node server.js**. As soon as we receive a positive response with the following text: “ **Server listening on port 7070** ”, it will be possible to use from the application at **http: // localhost: 7070** .