**Wstęp**

Studenci kierunków informatycznych niejednokrotnie podczas trwania studiów muszą zapoznać się z konfiguracją i zarządzaniem systemami operacyjnymi z rodziny Windows i GNU/Linux.Wiąże się to z koniecznością zapewnienia tym osobom dostępu do systemów operacyjnych z poziomu konta administratora. Z oczywistych względów, maszyny fizyczne, stanowiące standardowe wyposażenie pracowni komputerowej nie są w stanie zapewnić użytkownikom takiego wsparcia, gdyż wymagaloby to odswieżania ich obrazu po każdych zajęciach, co wiąże się ze zbyt dużym nakładem pracy.

Jednym ze sposobów rozwiązania tego problemu jest wykorzystanie technologii wirtualizacji, która umożliwia stworzenie autonomicznej maszyny wirtualnej wewnątrz fizycznego komputera. Na tak przygotowanej maszynie możliwe jest zainstalowanie wybranego systemu operacyjnego oraz dostęp z poziomu administratora.

W efekcie tworzy się swego rodzaju „system w systemie”, gdzie maszyna fizyczna, na której pracujemy to gospodarz (ang. *host*), zaś systemy wirtualne to jego goście (ang. *guest*). Maszyny wirtualne uruchamiane są w izolowanym środowisku, co pozwala na bezpieczne testowanie różnego rodzaju oprogramowania oraz samych systemów, bez szkody dla maszyny gospodarza. Co wiecej, można je też łatwo tworzyć, powielać, usuwać bądź zarządzać nimi bez konieczności logowania się do komputera, na którym działają. Takie rozwiązanie narzędziowe umożliwia tworzenie w stosunkowo prosty sposób całych wirtualnych laboratoriów, które obecnie są powszechnie stosowane w edukacji.

Niniejsza praca inżynierska poświęcona jest tworzeniu i obsłudze takiego wirtualnego środowiska, zbuwanego w oparciu o technologie QEMU/KVM. Stworzono w tym celu projekt oparty na maszynach wirtualnych, działających na dwóch komputerach fizycznych, znajdujących się w obrębie tej samej sieci wewnętrznej LAN. Do zarządzania zasobami wirtualnymi zbudowano aplikację webową napisaną przy użyciu technologii Bootstrap, NodeJS, JavaScript i Perl.

Praca rozpoczyna się częścią teoretyczną,którą tworzą następujące rozdziały:

1.*Teoretyczne podstawy wirtualizacji*,

2. *Przegląd istniejących typów wirtualizacji.*

Pierwszy rozdział opisuje na czym polega proces wirtualizacji, przedstawia jego początki oraz wyjaśnia podstawowe pojęcia z nim związane.

Następny rozdział przedstawia podział tego procesu na poszczególne typy, w zależności od przyjętej koncepcji; wyjaśnia róznice między nimi oraz przedstawia dostępne obecnie oprogramowanie do wirtualizacji. Na koniec scharakteryzowano architekturę środowiska sieciowego opartego na QEMU/KVM.

Po części teoretycznej, opisano proces realizacji projektu, którego etapami są

› określenie wymagań i celów wykonanego projektu; przedstawione w rozdziale trzecim zatytułowanym „Cele i założenia wykonanego projektu”;

›. charakterystyka części projektu; opisane w rozdziale czwartym- **„**Charakterystyka poszczególnych elementów projektu”;

›. realizacja wybranego rozwiązania, która zawiera się w rozdziale piątym- „Implementacja projektu”.

Pracę zamykają wnioski końcowe, gdzie przedstawiono wady i zalety użytej architektury na tle innych dostępnych obecnie rozwiązań.

**1. Teoretyczne podstawy wirtualizacji**

* 1. **Czym jest wirtualizacja?**

Wirtualizacja jest szeroko rozumianym pojęciem, którego można używać między innymi w kontekście sieci, aplikacji, serwerów, pamięci masowych czy nawet stacji roboczych. Jest to metoda (technika) zarządzania zasobami, która pozwala na ich współdzielenie przez procesy lub nawet systemy komputerowe za pomocą podziału i agregacji.[[1]](#footnote-1) Innymi słowami, wirtualizacja polega na użyciu oprogramowania w celu uzyskania abstrakcji (iluzji) posiadanych zasobów.

W ujęciu węższym, proces wirtualizacji rozumiany jest jako zastosowanie maszyn wirtualnych (ang.*virtual machine*- VM) lub środowiska wirtualnego (ang.*virtual environment*- VE), w celu wykonania oprogramowania przeznaczonego dla tej samej platformy sprzętowej, ale wymagającego specjalnego środowiska wykonawczego.[[2]](#footnote-2)

**1.2. Początki wirtualizacji**

Pierwsze prace z użyciem wirtualizacji przeprowadzone zostały w latach 1956 – 1962 podczas pracy nad projektem tworzonym we współudziale Uniwersytetu w Manchesterze i firm Ferranti oraz Plessey, którego efektem był komputer o nazwie ATLAS. W stworzonej maszynie część instrukcji zewnętrznych wykonywana była przez sprzęt, a część przez przez program nadzorujący (ang. *supervisor*). Komputer miał za zadanie w odpowiedni sposób wykonywać przechwycone wywołania, a wirtualizacja była w tym przypadku jedynie efektem ubocznym zastosowanych rozwiązań.[[3]](#footnote-3)

Faktyczne użycie wirtualizacji miało miejsce dopiero w połowie lat 60. w Nowym Jorku jako owoc pracy firmy IBM (ang. *International Business Machines*) i MIT (ang.*Massachusetts Institute of Technology*) nad projektem M44/44X. Projekt ten składał się z fizycznej maszyny M44, opartej o o platformę sprzętową IBM 7044 oraz maszyn wirtualnych (nazywanych wówczas ang. *pseudo-machines*) 44X. Maszyny wirtualne współdzieliły pamięć oraz oprogramowanie komputera matki M44. Zastosowane rozwiązanie polegało na dokładnym sklonowaniu tego komputera przez maszyny 44X. Przedstawiony projekt był czysto naukowy i nigdy nie został zastosowany komercyjnie, stał się jednak wzorem dla kolejnych prac nad wirtualizacją. Wkrótce stworzono kolejne maszyny wirtualne,wśród nich m.in.:

•  CP40 (1965r.) – oparta na IBM 860/40;

• CP67 (1965r.) – oparta na IBM360/67, był to pierwszy 32-bitowy komputer, który wspierał pełną wirtualizację;

• CP67 (1967r.)- następca CP40, oparty na architekturze IBM360/6;

• VM/370 (1970r.) – seria popularnych VM bazujących na CP67.

Wyżej wymienione maszyny były idealnymi kopiami fizycznego sprzętu, obsługiwanego przez monitor maszyn wirtualnych, który pozostawał w bezpośrednim kontakcie z systemem komputerowym. W ten sposób powstawało wiele kopii tego systemu operacyjnego, przez co wykorzystanie procesora na komputerach klasy Mainframe uległo znacznemu zwiększeniu.[[4]](#footnote-4)

Dość gwałtowny rozwój technologii wirtualnych miał miejsce w latach 90. W tym okresie stworzony został język Java, korzystający z maszyn wirtualnych, a w 1998 roku pojawiła się firma *Citrix Systems* (1989r.) oraz *VMware* (1998r.), produkujące oprogramowanie do wirtualizacji.

Firma *Microsoft* także dołączyła się do branży i w 2003 roku przejęła powstałą w 1988 firmę *Connectix Corp*, która stworzyła program umożliwiający wirtualizację systemów operacyjnych. Powszechnie dziś używany *Microsoft Virtual PC*, miał za zadanie uruchomienie systemu Windows na hoście z systemem *Mac OS X*.[[5]](#footnote-5)

Kolejnym znaczącym wydarzeniem było pojawienie się w 2007 roku programu *Oracle Virtualbox*, stworzonego przez firmę *Innotek GmbH*, dzisiaj jednak wydawanego pod logiem firmy *Oracle*.[[6]](#footnote-6)

Obecnie, w dalszym ciągu można oczekiwać popularyzacji wirtualizacji i zwiększenia zastosowania maszyn wirtualnych, także w domowych stanowiskach.

**1.3. Kryteria realizowalności maszyny wirtualnej (VM)**

Teoretyczne podstawy wirtualizacji zostały sformułowane w 1974 roku przez Geralda J. Popek i Roberta P. Goldberg, którzy w artykule pt.: *Formal Requirements for Virtualizable Third Generation Architectures*  wymienili warunki, jakie musi spełniać maszyna wirtualna, aby mogła działać poprawnie.[[7]](#footnote-7) Zgodnie z ich tezą, są trzy takie własności :[[8]](#footnote-8)

* odpowiedniość(ang.*equivalence*) – program działający na maszynie wirtualnej musi zachowywać się w taki sam sposób, jak na sprzęcie fizycznym;
* kontrola zasobów(ang.*resource control*) - wirtualna maszyna musi w pełni kontrolować wszystkie zasoby, które są wirtualizowane;
* wydajność(ang.*efficiency*) – czyli większa część instrukcji musi być wykonywana bez udziału maszyny wirtualnej (przez sprzęt).

W praktyce, wszystkie trzy warunki nie muszą być (i zazwyczaj nie są) dokładnie spełniane.

Popek i Goldberg podzielili także instrukcje maszyny wirtualnej na trzy grupy: [[9]](#footnote-9)

* **instrukcje uprzywilejowane** - ich efektem jest przerwanie lub wywołanie systemowe w trybie użytkownika lub ich brak w trybie jądra;
* **instrukcje wrażliwe ze względu na kontrolę** – mogą zmieniać konfigurację zasobów systemowych;
* **instrukcje wrażliwe ze względu na wykonanie**- ich działanie zależy od konfiguracji systemu.

Idąc dalej tokiem ich rozumowania, najtrudniejsze pod wzgledem implementacji jest wykonanie instrukcji wrażliwych i uprzywilejowanych, co w konsekwencji prowadzi do wyodrębnienia kolejnych trzech rodzajów wirtualizacji:[[10]](#footnote-10)

* **pełnej wirtualizacji z wykorzystaniem translacji binarnej**;
* **wirtualizacji systemowej/parawirtualizacji**;
* **wirtualizacji sprzętowej**.

Biorąc pod uwagę zdefiniowane zbiory instrukcji i ograniczenia w postaci warunków, Popek i Goldberg sformułowali twierdzenie, które stało się podstawą rozwoju technologii związanych z wirtualizacją:

*Dla każdego standardowego komputera trzeciej generacji wirtualna maszyna może zostać skonstruowana, jeśli zbiór instrukcji wrażliwych jest podzbiorem zbioru instrukcji uprzywilejowanych.[[11]](#footnote-11)*

Z powyższego twierdzenia wynika, iż każda operacja powodująca błędne działanie maszyny wirtualnej powoduje przerwanie lub wywołanie systemowe, a cała reszta instrukcji nieuprzywilejowanych jest przekazywana po prostu do fizycznego sprzętu, w celu uzyskania odpowiedniej wydajności.

Nie sprawdziło się to jednak w praktyce, gdyż można także wirtualizować bez spełnienia tego założenia, godząc się na spadek wydajności. Dzieje się tak m.in. w przypadku architektury x86.[[12]](#footnote-12)

**2. Przegląd istniejących typów wirtualizacji**

**2.1. Rodzaje wirtualizacji**

Obecnie istnieje wiele różnych podziałów technologii wirtualizacji, definiowanych w zależności od tego co i w jaki sposób wirtualizują. Każdy producent oprogramowania do wirtualizacji podchodzi w inny sposób do tego procesu.

Tak np. firma VMware rozróżnia następujące rodzaje wirtualizacji:[[13]](#footnote-13)

• wirtualizacja wspomagana sprzętowo (ang. *Hardware Assisted Virtualization*),

• pełna wirtualizacja z użyciem techniki translacji binarnej (ang. *Full Virtualization using Binary Translation*),

• wirtualizacja wspomagana programowo lub parawirtualizacja (ang. *OS Assisted Virtualization or Paravirtualization*).

Według firmy Microsoft istnieją następujące rodzaje wirtualizacji:[[14]](#footnote-14)

• hipernadzorca typu 1 (ang. *Type 1 Hypervisor*),

• hipernadzorca typu 2 (ang. *Type 2 Hypervisor*),

• hipernadzorca monolityczny (ang. *Monolithic Hypervisor*),

• hipernadzorca typu mikrojądro (ang. *Microkernel Hypervisor*).

Profesor Andrew S.Tanenbauma, twórca systemu operacyjnego *MINIX*, twierdzi, iż wirtualizację powinno się dzielić na poniższe typy:[[15]](#footnote-15)

• hipernadzorca typu 1,

• hipernadzorca typu 2,

• parawirtualizacja.

Zdaniem innych autorów, wirtualizację można podzielić ze względu na rodzaj wirtualizowanych zasobów fizycznych na poniższe typy:

• wirtualizacja serwerów,

• wirtualizacja desktopów,

• wirtualizacja sieci,

• wirtualizacja aplikacji,

• wirtualizacja pamięci masowych.

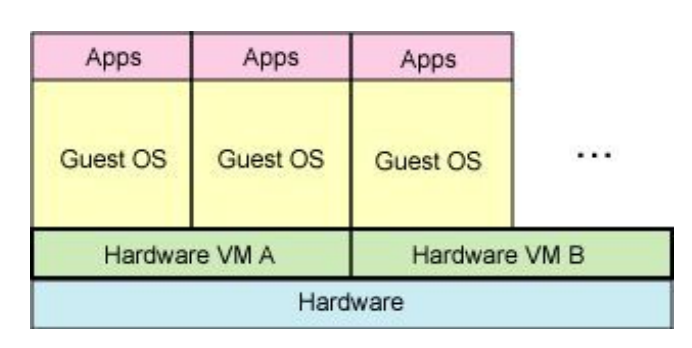
W niniejszej pracy zastosowano podział zaproponowany przez Wright State University, który uwzględnia typy najczęściej pojawiające się w pismach naukowych.[[16]](#footnote-16)

Są to kolejno:[[17]](#footnote-17)

* **emulacja sprzętowa (ang. *Emulation-based Full Virtualization*)** - jest to pełna emulacja sprzętu, czyli każdego elementu podzespołu komputera (procesora, pamięci, dysku twardego oraz urządzeń wejścia/wyjścia). Proces ten pozwala na uruchomienie środowisk o zupełnie odmiennej architekturze, jest jednak mało wydajny i wolny.[[18]](#footnote-18)

Do tego rodzaju zaliczane są m.in: Bochs, VirtualPC for Mac, QEMU.[[19]](#footnote-19)

**Rys1. Schemat emulacji sprzętowej**

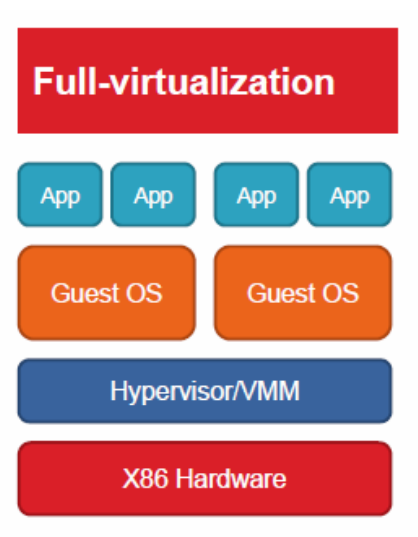


Źródło: M.Bishopp, *Introduction to Computer Security*, Addison-Wesley, 2005, s.649

* **pełna wirtualizacja/wirtualizacja natywna (ang. *Native / Full Virtualization*)** – ten rodzaj wirtualizacji wymaga spełnienia zalożeń kryteriów Popka-Goldberga. Jest bardziej wydajna niż tradycyjna emulacja, gdyż umożliwia uruchamianie systemu operacyjnego kompatybilnego z posiadanym systemem gospodarza. Możliwe zatem jest uruchomienie dowolnego systemu operacyjnego na maszynie wirtualnej gościa.

W tym procesie, wirtualizowana jest tylko część wywołań potrzebnych do izolacji oprogramowania. Maszyna wirtualizuje tylko te instrukcje, które mogłyby kolidować z działalnością innych środowisk wirtualizowanych lub systemu operacyjnego gospodarza. Cała reszta jest przetwarzana przez fizyczny sprzęt, celem zwiększania wydajności. [[20]](#footnote-20)Uprzywilejowane instrukcje (np.: dostęp do pamięci, operacje dyskowe)są zazwyczaj modyfikowane, tak aby odnosiły się jedynie do wirtualnego środowiska, w którym zostały wykonane. Nieuprzywilejowane instrukcje (np.:wejścia/wyjścia, obsługa sieci***)*** są zazwyczaj przekazywane wprost do procesora.

Schemat budowy tej technologii przedstawia rysunek zamieszczony poniżej.

**Rys 2. Schemat budowy pełnej wirtualizacji na przykładzie architektury x86**

Źródło: *Wirtualizacja*, <http://ii.uni.wroc.pl/~msq/so09/slides/2_4_wirtualizacja.pdf> (data odczytu 17 grudnia 2015)

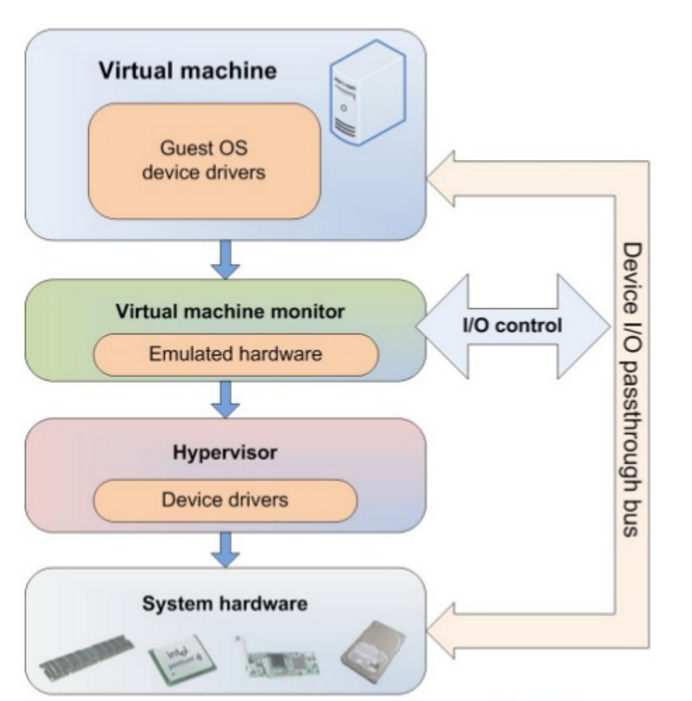
Niewątpliwą wadą wirtualizacji natywnej jest konieczność emulacji każdego urządzenia, przez co procesor maszyny fizycznej jest mocno obciążony. Emulator wykonuje w pętli wszystko to, co robiłby rzeczywisty procesor maszyny fizycznej, co w efekcie skutkuje zmniejszeniem wydajności pracy komputera- nawet do 20-30% narzutu na obsługę wirtualizacji.

Z drugiej strony, do zalet tego rozwiązania należy brak konieczności modyfikacji systemu uruchamianego w wirtualnym środowisku, możliwość użycia dowolnego systemu operacyjnego w środowisku hosta oraz bardzo dobre odizolowanie między wirtualnymi środowiskami.[[21]](#footnote-21)

Wśród technologii pełnej wirtualizacji znajdują sie m.in: *VMware ESX* *Server*, *Microsoft Virtual Server*.[[22]](#footnote-22)

* **wirtualizacja wspierana sprzętowo** – nazywana jest takze **wirtualizacja pełna ze wsparciem sprzętowym (**ang. HVM – *Hardware-assisted Virtual Machine*). Jest rozszerzeniem techniki pełnej wirtualizacji poprzez możliwość użycia technologii nowych procesorów (Intel-VT, AMD-V)[[23]](#footnote-23), które realizują sprzętowe wsparcie wirtualizacji. Metoda ta pojawiła sie w roku 2007, wraz z powstaniem ww. procesorów.

Schemat działania tej technologii przedstawia poniższy rysunek:

**Rys 3. Schemat budowy wirtualizacji wspieranej sprzętowo** 

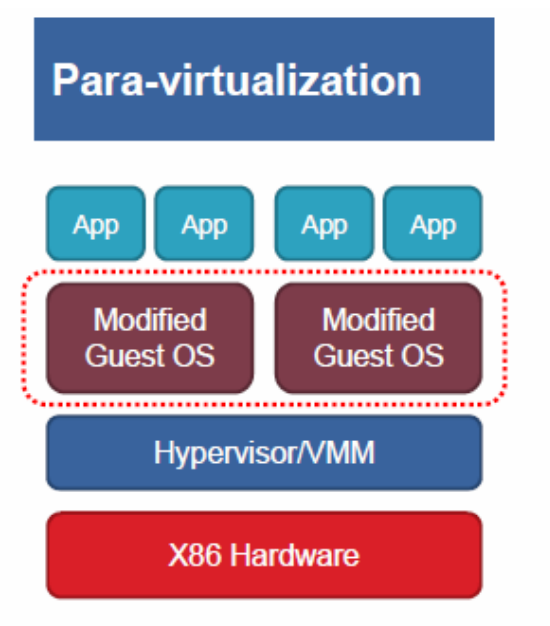
Źródło: J. Huang, *Embedded Virtualization applied in Mobile Devices,* Academia Sinica, Tajwan 2012, s.12

Dużą zaletą tego typu wirtualizacji jest zwiększona wydajność - dzięki wsparciu sprzętowemu część zadań realizowana jest bezpośrednio przez procesor hosta bez kosztownego procesu emulacji i modyfikacji jądra.Technologia ta jest bardziej wydajna, w porównaniu do pełnej wirtualizacji. Ponadto, spełnia te same założenia, co wirtualizacja natywna, czyli możliwość uruchomienia dowolnego systemu operacyjnego na hoście, brak konieczności modyfikacji systemu operacyjnego, dobrze odizolowane środowiska wirtualne.

Istotną wadą tej technologii jest jednak konieczność posiadania sprzętowych rozszerzeń wirtualizacji w procesorze. Jeżeli zamierzamy na fizycznym serwerze dedykowanym stworzyć maszyny wirtualne, to najpierw należy się upewnić, czy dany procesor posiada odpowiednie rozszerzenia- *Intel Virtualization Technolog* (VT) lub *AMD-Virtualization* (AMD-V ). Większość dostępnych obecnie na rynku procesorów spełnia te wymagania. Rozwiązaniami tego rodzaju wirtualizacji są m.in: KVM, Xen HVM, VMware ESXi vSphere, MS Hyper-V.

* **parawirtualizacja (hipernadzorca)** – jest wspierana jedynie przez systemy operacyjne umożliwiające modyfikację jądra. Głównym celem jest osiągnięcie maksymalnej możliwej wydajności poprzez wyeliminowanie potrzeby emulacji sprzętu oraz odpowiednie przekształcenia jądra. Wirtualna maszyna nie emuluje sprzętu , lecz implementuje obsługę API, z którego mogą korzystać odpowiednio zmodyfikowane systemy operacyjne gości. Zadaniem API jest podział fizycznych zasobów pomiędzy maszyny wirtualne. Oprogramowanie nie emuluje całości sprzętu, a jedynie zapewnia rozłączny dostęp do niektórych fizycznych komponentów, emulując tylko niektóre.[[24]](#footnote-24)

Schemat budowy tego typu wirtualizacji przedstawiono poniżej:

**Rys 4. Schemat budowy parawirtualizacji na przykładzie architektury x86**

Źródło: *Wirtualizacja*, http://ii.uni.wroc.pl/~msq/so09/slides/2\_4\_wirtualizacja.pdf (data odczytu 21 grudnia 2015)

Parawirtualizacja jest ciekawym podejściem do rozwiązania problemu dostępu do określonych rejestrów procesora przez wiele systemów operacyjnych jednocześnie.

Ten rodzaj wirtualizacji wykorzystuje hypervisor typu1. Jeden z systemów gościa jest uprzywilejowany, przy czym zachodzi konieczność modyfikacji jądra systemów gości celem odwołań do hyperwisora.[[25]](#footnote-25)

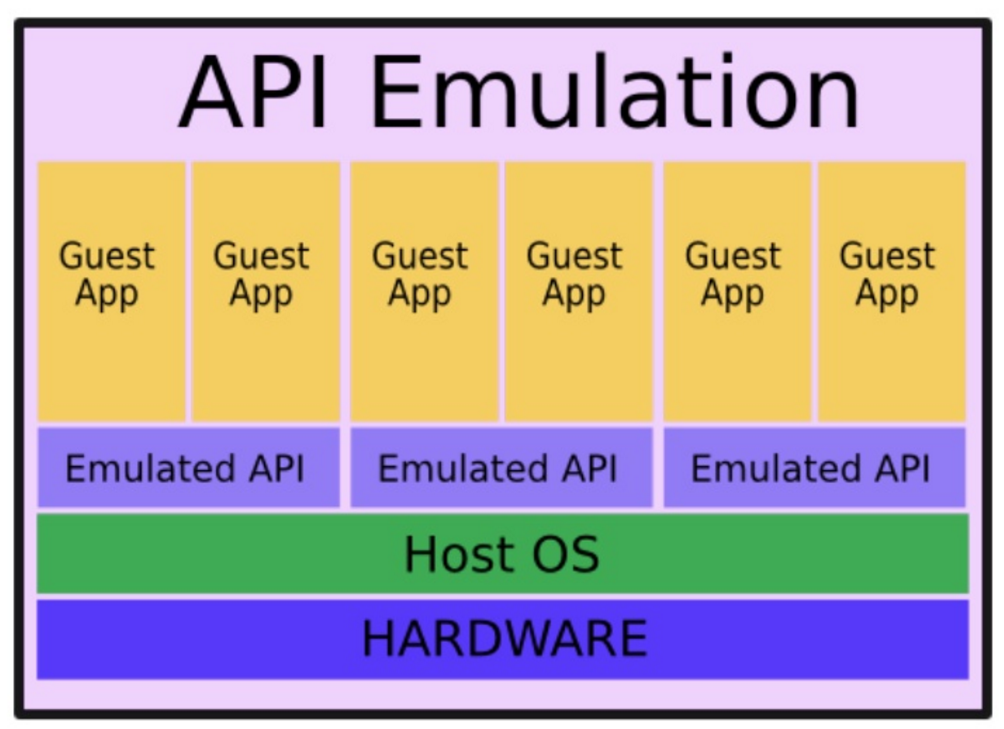
Zaletą parawirtualizacji jest bardzo dobra wydajność systemów gości, przy zmodyfikowanym jądrze. Wadą natomiast jest brak możliwości uruchomienia dowolnego systemu operacyjnego. Odpowiednio zmodyfikowane jądra posiadają systemy Linux oraz FreeBSD, nie uruchomimy tak natomiast systemów z rodziny Windows. Przykładem rozwiązania technologii parawirtualizacji są: Xen PV, Parralells Workstation, Enomalism, Vmware ESX Server i Win4Lin 9x .

* **emulacja API** – w tej technologii zamiast całego sprzętu i systemu operacyjnego emulowany jest tylko interfejs aplikacji. Wirtualna maszyna zapewnia działajacej na niej aplikacji, która czasem nie musi być nawet wirtualizowana, wirtualne rejestry i dostęp do plików. Dzięki temu eliminowane są konflikty między aplikacjami oraz na linii aplikacja – system operacyjny.[[26]](#footnote-26)

Zaletą tego rozwiązania jest przenośność aplikacji,łatwiejsze udostępnianie jej użytkownikom i zarządzanie jej licencjami. Wadą natomiast jest konieczność instalacji wirtualnej maszyny na stanowisku, gdzie aplikacja ma być uruchomiona.

Przykładem tej technologii może być m.in.: *Microsoft Application Virtualization*, *Citrix XenApp,VMware ThinApp*.[[27]](#footnote-27)

**Rys 5. Schemat budowy emulacji API**



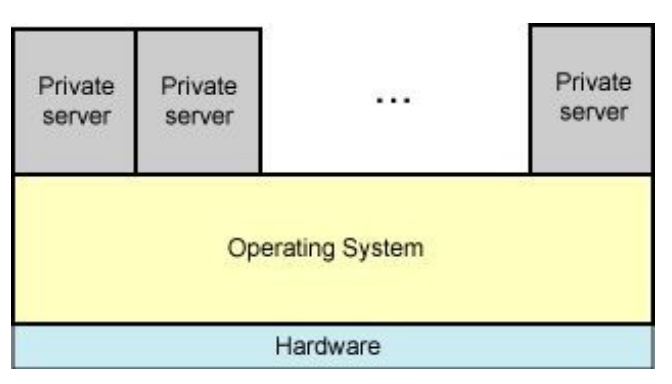
Źródło: S.Wójtczak, *Wirtualizacja Systemów Operacyjnych*, Uniwersytet Łódzki, Łódź 2008, <http://www.strony.toya.net.pl/~vermaden/tmp/thesis.pdf>, (data odczytu 23 grudnia 2015)

* **wirtualizacja na poziomie systemu operacyjnego**- szczególnie popularna wśród dostawców VPS. Ten rodzaj wirtualizacji polega na podziale jednej fizycznej maszyny na kilka wirtualnych, opartych o ten sam (lub zbliżony) system operacyjny w tzw. kontenerze.

Z technicznego punktu widzenia, jest to pewien sposób izolacji kolejnych instancji tego samego systemu operacyjnego w ramach jednej maszyny fizycznej. Kontener posiada wydzielone zasoby oraz izolację procesów i urządzeń, przez co zapewnia bezpieczeństwo innym kontenerom, oraz systemowi hosta. Takie rozwiązanie jest bardzo wydajne ze względu na brak potrzeby tworzenia całego wirtualnego komputera i emulacji urządzeń. Z kontenerami wiąże się również znacznie więcej ograniczeń w porównaniu z wirtualizacją sprzętową. System uruchomiony w kontenerze współdzieli jądro z systemem hostem, przez co nie ma możliwości kompilacji własnych modułów, aktualizacji jądra.

Tego rodzaju wirtualizacja doskonale nadaje sie do zastosowań typowo hostingowych, typu poczta, www. Rozwiązania tej technologii występiją m.in w *OpenVZ* (otwarta wersja *Virtuozzo)*, *Parallels Virtuozzo*, *LXC* (natywna technologia wbudowana w jądro Linux).

**Rys 6. Schemat budowy wirtualizacji systemu operacyjnego**



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

**2.2. Hipernadzorca typu 1 vs. hipernadzorca typu 2**

Z pojęciem wirtualizacji wiąże się wspomniane już pojęcie hipernadzorcy (hiperwisora).

Hiperwisor, czyli inaczej *Virtual Machine Monitor (VMM)* czuwa nad prawidłowym przebiegiem procesu wirtualizacji. Jest to program komputerowy, który pozwala na jednoczesne uruchomienie wielu systemów operacyjnych na tym samym komputerze. W momencie gdy system operacyjny maszyny wirtualnej (gość) odwołuje się do sprzętu komputerowego, jego żądanie jest przechwytywane, a następnie obsługiwane przez hipernadzorcę. Dzięki temu możliwy jest równoległy dostęp do sprzętu komputerowego przez wiele działających równocześnie systemów operacyjnych.[[28]](#footnote-28)

Wyróżniane są dwa typy hipernadzorców, w zależności od miejsca ich działania.

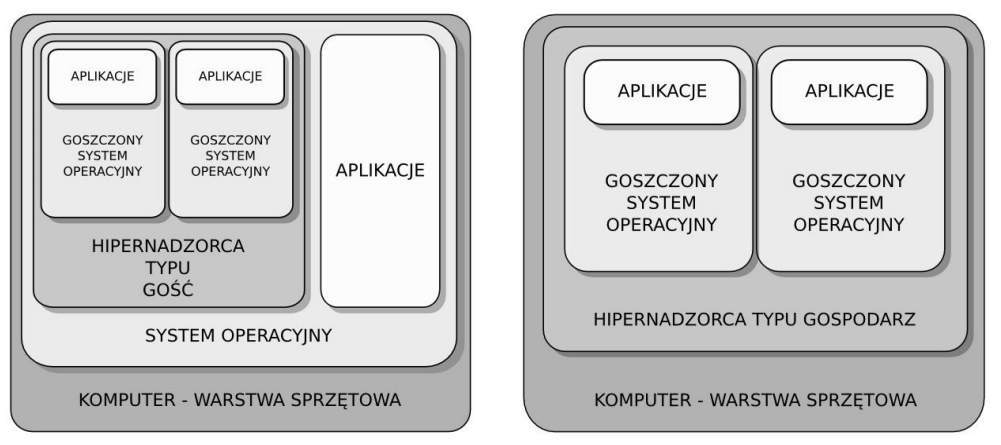
Są to:

• **Hipernadzorca typu 1/natywny** (ang. *Native Virtual Machine Monitor* , “*bare metal”*) - działa bezpośrednio na poziomie sprzętu, mając nad nim pełną kontrolę i monitorując uruchomione systemy operacyjne.

• **Hipernadzorca typu 2/ hostowany** (ang. *Hosted Virtual Machine Monitor*)- działa jako program uruchomiony na danym systemie operacyjnym (hoście).[[29]](#footnote-29)

Różnice między obydwoma typami przedstawia rysunek poniżej:

**Rys 7. Modele działania maszyny wirtualnej z hipernadzorcą typu gość (typ2) i gospodarz (typ1)**



Źródło: L.Kaczmarek, M.Wróbel; *Możliwości Stosowania Wirtualizacji w Systemach Komputerowych. Zastosowanie Komputerów w Nauce i Technice:* Zeszyty Naukowe Wydziału Elektrotechniki i Automatyki Politechniki Gdańskiej Nr 30; Gdańsk 2014; s.2

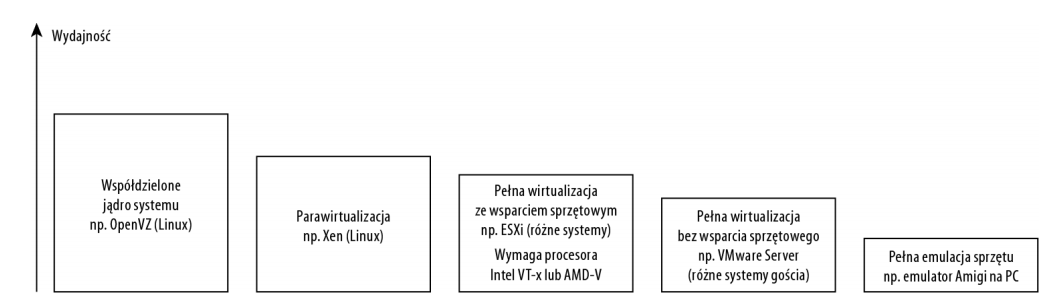
**2.3. Przegląd wybranego oprogramowania do wirtualizacji**

Poniżej przedstawiono spis najpopularniejszych obecnie rozwiązań wirtualizacyjnych, potocznie zwanych „maszynami wirtualnymi”. W zależności od różnych czynników (np.systemu operacyjnego gościa) poniższe oprogramowania mogą używać kilka różnych typów wirtualizacji jednocześnie:

* ***VMware ESXi, VMware Server*** — pełna wirtualizacja, wirtualizacja ze wsparciem sprzętowym, także elementy parawirtualizacji;
* ***Xen*** — parawirtualizacja, pełna wirtualizacja ze wsparciem sprzętowym, emulacja;
* ***Microsoft Hyper-V, Virtual PC*** — pełna wirtualizacja, wirtualizacja ze wsparciem sprzętowym, emulacja;
* ***VirtualBox*** — pełna wirtualizacja, wirtualizacja ze wsparciem sprzętowym, emulacja;
* ***OpenVZ***— współdzielone jądro — najwyższa wydajność, ale ograniczenie tylko do jednego typu systemu operacyjnego (Linux).

Porównanie wydajności tych technologii przedstawia poniższy rysunek:

**Rys 8. Porównanie wydajności technologii wirtualizacji**



Źródło: M.Serafin, *Wirtualizacja w Praktyce*, Helion, Gliwice 2012, s.16

**2.4. Architektura wirtualizacji w oparciu o QEMU/KVM**

Niniejsza praca zajmuje się szczegółowym omówieniem wirtualizacji, bazującej na technologiach QEMU/KVM. W poprzednim rozdziale wspomniano, iż KVM jest przykładem **wirtualizacji pełnej ze wsparciem sprzętowym** (HVM), natomiast QEMU należy do typu emulacji. Jak natomiast funkcjonują w obrębie jednej konfiguracji?

Aby lepiej zrozumieć tę problematykę, należy odwołać się do terminu, jakim jest emulator.

Wspomniano już, że emulacja to proces polegający na symulowaniu przez program komputerowy działąjcy w danym systemie komputerowym pełnego zachowania innego systemu komputerowego, najczęściej o odmiennej architekturze. Program wykonujący te symulacje nosi nazwę emulatora. W praktyce wyróżniamy dwa rodzaje emulatorów:

* **emulatory pełne**- czyli takie, które działają zgodnie ze specyfikacją określonego fizycznego komputera. W tej właśnie grupie znajduje się QEMU, jak również Bochs, UAE, Frodo
* **emulatory API**- które emulują jedynie działanie interfejsu użytkownika określonego systemu operacyjnego, nazywane są one często warstwami kompatybilności. Do tej grupy należy np. Wine, DosBox, Dosemu.

Wiadomo zatem, iż QEMU to zarówno maszyna wirtualna jak i emulator. Jest emulatorem, ponieważ może uruchamiać zarówno pojedyncze programy jak i całe systemy operacyjne stworzone dla innego procesora niż ten, na którym aktualnie pracuje. Dzieje się tak przez szybką translację rozkazów procesora docelowego na rozkazy procesora, na którym w danym momencie pracuje QEMU.

Z drugiej strony, w trybie wirtualizacji (z użyciem Xen lub KVM) QEMU działa na procesorze hosta.

Jeżeli chodzi o KVM (ang. Kernel-based Virtual Machine)- jest to rozwiązanie pozwalające na wirtualizację z wykorzystaniem hiperwizora działającego jako moduł jądra Linuksa. Komponent KVM pojawił się w jądrze Linuksa wraz z wersją 2.6.20 (wydaną w styczniu 2007r.). KVM jest platformą wirtualizacyjną dla systemu Linux na sprzęcie x86 zawierającym rozszerzenia wspierające wirtualizację (Intel VT lub AMD-V). W jej skład wchodzi moduł jądra (nazwany kvm.ko), który dostarcza podstawową infrastrukturę wirtualizacyjną, moduły specyficzne dla procesora (kvm-intel.ko lub kvm-amd.ko) i komponent przestrzeni użytkownika (zmodyfikowana wersja QEMU). Zgodnie z tym, maszyna wirtualna implementowana jest jako zwykły proces w Linuksie, przez co KVM korzysta ze wszystkich cech jądra, m.in. zapewnia standardowy Linuksowy model bezpieczeństwa i kontrolę zasobów. Za emulacje urządzeń odpowiedzialna jest zmodyfikowana wersja QEMU, która dostarcza emulowany BIOS, szynę PCI, szynę USB i standardowy zbiór urządzeń, takich jak kontrolery dysków IDE i SCSI, karty sieciowe itp. Dodatkowo QEMU w takiej architekturze wspiera dwa tryby pracy: emulatorora oraz hiperwisora z obsługą pełnej wirtualizacji (bez wsparcia dla rozszerzeń sprzętowych wirtualizacji).

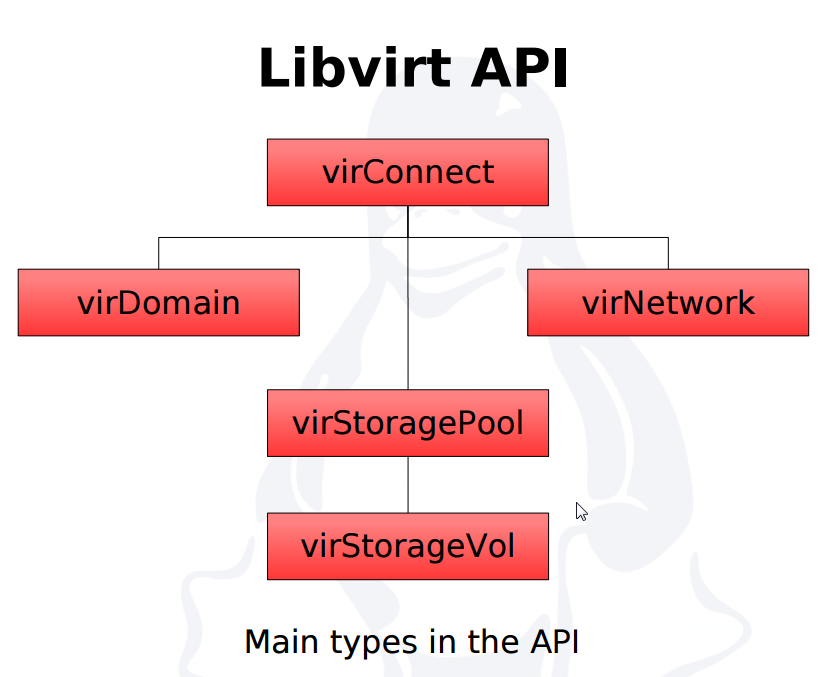
Odnośnie zarządzania pamięcią, KVM dziedziczy go z Linuxa. Pamięć maszyny wirtualnej jest przechowywana tak jak pamięć każdego procesu, przez co może podlegać wymianie, a współdzielenie stron pamięci jest dodatkowo wspierane przez KSM (ang. *Kernel Same-Page Merging*). KSM przegląda pamięć każdej maszyny wirtualnej i jeśli znajdzie identyczne strony pamięci, skleja je w jedną współdzieloną stronę. Jeśli gość spróbuje wprowadzić modyfikacje, to dostanie własną kopię.

Do zarządzania maszynami wirtualnymi w tej technologii wykorzystano bibliotekę Libvirt API, która stanowi uniwersalne i otwarte API do obsługi i zarządzania maszynami wirtualnymi w systemach GNU/Linux. Umożliwia także komunikowanie się z różnymi hiperwisorami –lokalnie i zdalnie. Obecnie Libvirt API wspiera nastepujące technologie, a jego najbardziej intensywny rozwój można zaobserwowac w ramach w ramach RHEL/CentOS i Fedora:

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

Niewątpliwym atutem Libvirt API jest zróżnicowanie pod względem zarządzanymi funkcjonalnościami maszyn wirtualnych. Najważniejsze z nich przedstawia poniższy rysunek:

**Rys 9. Zróżnicowanie funkcjonalności Libvirt API**



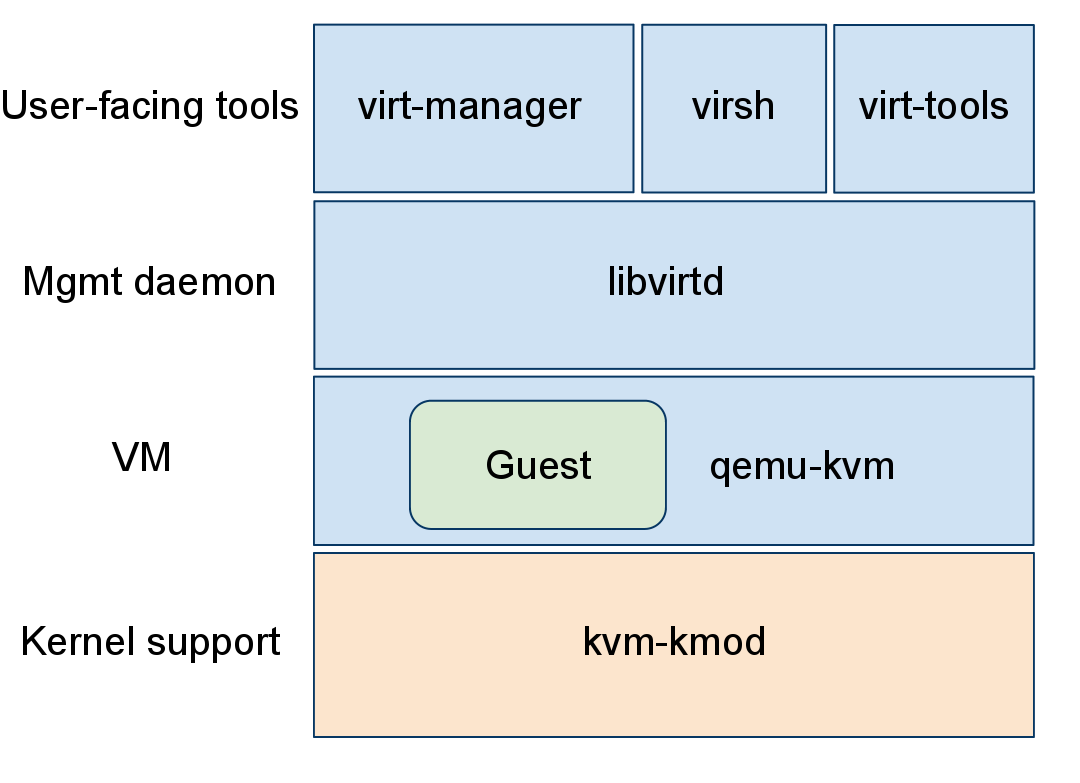
Źródło: <http://www.ibm.com/developerworks/library/l-libvirt/> (data odczytu 21 luty 2016)

Ww. API jest odpowiedzialne za konfigurację prawie wszystkich aspektów wirtualizacji. W ramach jego działania dostępna jest uniwersalna konsola *virsh*, pozwalająca m.in na tworzenie i konfigurację maszyn wirtualnych, konfigurację pamięci masowych oraz zarządzanie siecią. Możliwy jest także dostęp zdalny z użyciem protokołów: SSH, TLS, TCP.

Oczywiście, samo Libvirt API w połączeniu z KVM/QEMU nie jest wystarczające, aby móc efektywnie zarządzać środowiskami wirtualnymi. Potrzebny jest jeszcze zestaw narzędzi, które implementowałyby jego funkcje. W tej pracy użyto do tego celu aplikacji graficznej *Virtual Machine Manager*, która zarządza wirtualizacją poprzez bibliotekę libvirt.

Całoksztalt użytej architektury przedstawia poniższy schemat:

**Rys 9. Schemat budowy architektury QEMU/KVM**



Źródło: <https://www.ibm.com/developerworks/community/blogs/ibmvirtualization/entry/kvm_architecture_the_key_components_of_open_virtualization_with_kvm2?lang=en> (data odczytu 21 luty 2016)

**3. Cele i założenia wykonanego projektu**

**3.1. Opis i cele projektu**

Ogólnym celem wykonanego na potrzeby pracy projektu było stworzenie wirtualnych środowisk opartych o technologię QEMU/KVM oraz zbudowanie aplikacji, umożliwiającej zdalne zarządzanie wirtualnymi. zasobami.

Zaimplementowany system wykonano w oparciu o trzy maszyny fizyczne, nazwane kolejno komp1, komp2 i komp3, które znajdują się w obrębie tej samej sieci lokalnej LAN.

Jedna z maszyn służy jako serwer HTTP, z aplikacją webową do zdalnego zarządzania maszynami wirtualnymi i fizycznymi. Dostęp do niej, posiada jedynie administrator sieci, gdyż aplikacja działa tylko na jego maszynie fizycznej i tylko na tym komputerze jest skonfigurowana.

Na pozostałych dwóch komputerach fizycznych znajdują się maszyny wirtualne z zainstalowannymi systemami operacyjnymi: Slitaz40 oraz Damn Small Linux.

Stworzona w ten sposób infrastruktura ułatwia i przyspiesza dostępu do poszczególnych systemów operacyjnych na tym samym komputerze fizycznym.

Jest to niewątpliwa zaleta zarówno pod względem naukowym i dydaktycznym, jak i ekonomicznym. Dzięki temu można zminiejszyć koszty związane z zakupem licencji na oprogramowanie oraz koszty dostępu do poszczególnych aplikacji.

Duże znaczenie ma również efektywne wykorzystywanie zasobów i zmniejszenie podatności systemów na awarie i przeciążenia. Projekt umożliwia bezpieczne przeprowadzanie testów aplikacji w różnych systemach operacyjnych. W prosty sposób można zapisać także stan konfiguracji i powrócić do niej w innym, dogodniejszym terminie. Prowadzi to do konsolidacji systemów IT, co potwierdza wciąż rosnąca popularność rozwiązań zwirtualizowanych.

**3.2. Wymagania i założenia sprzętowe**

Podstawowym wymogiem sprzętowym opisanej w dalszych rozdziałach infrastruktury, jest poprawnie działająca sieć lokalna z dostępem do Internetu. Do jej konfiguracji w projekcie wykorzystano router TP-Link TL-WR1043ND.

Ponieważ podczas konfiguracji maszyn wirtualnych posłużono się mostem, który nie działa z interfejsami typu wireless, sieć lokalna musi być przewodowa (ang.*wired*).

Przed rozpoczęciem pracy z użyciem KVM należy sprawdzić, czy procesor maszyny fizycznej posiada wsparcie dla technologii VT (ang.*Virtualization Technology*). Aby zweryfikować tę informację, należy udać się do pliku **/proc/cpuinfo** i sprawdzić, czy nie ma tam flag **vmc** (dla procesorów Intela) lub **svm** (dla procesorów AMD). Procesory obsługujące wirtualizację posiadają odpowiednie wsparcie sprzętowe- dla procesorów AMD jest to technologia AMD-V, natomiast dla procesorów Intel - VT-x.

Lista dostępnych aktualnie procesorów, wspierających wirtualizacje znajduje się poniżej:

**Intel:**

• Pentium 4 662 and 672, Extreme Edition 955 i 965 (bez Pentium 4 Extreme Edition z HT).

• Pentium D 920 do 960 z pominięciem 945, 925 i 915;

• Core Duo T2300, T2400, T2500, T2600, T2700, L2000 i U2000;

• Core 2 Solo;

• Core 2 Duo z wyjątkiem E6540, E8190, E7xxx, E4xxx, T5200-T5550 i

T5750;

• Core 2 Quad z wyjątkiem Q8200;

• Core 2 Extreme Duo i Quad;

• seria Xeon 3000;

• seria Xeon 5000;

• seria Xeon 7000.

**AMD:**

• Athlon 64 i Athlon 64 X2 z serii "F" i "G" dla gniazd AM2 (z wyjątkiem

939);

• Turion 64 X2;

• Opteron 2 i 3r generacji;

• Phenom.

Maszyny fizyczne muszą posiadać także odpowiednią ilość pamięci, tak aby można było utworzyć na każdej z nich po trzy maszyny wirtualne (oraz ich kopie).

Dodatkowo, korzystając z KVM, zalecane jest aby komputer posiadał dual-core lub quad-core CPU.

Jeżeli chodzi o QEMU, technologia ta wspiera następujące architektury: x86, x86-64, PowerPC, SPARC; posiada jednak niepełne wsparcie dla systemu Windows.[[30]](#footnote-30)Dodatkowo, użycie QEMU możliwe jest dopiero od wersji 2.6.2.3 jądra Linuxa.

Do wykonania przedstawionego w pracy projektu użyto trzech maszyn fizycznych (dwóch typu laptop i jednej typu desktop) o następujących parametrach technicznych:

Tabela 1. Zestawienie wybranych parametrów technicznych sprzętu

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **nazwa komputera** | **typ komputera** | **model komputera** | **system operacyjny** | **parametry techniczne** |
| komp1 | laptop | Acer Aspire AS4830T-6678 | Xubuntu14.02 | – procesor: Intel® Core™ i3-2370M  -karta graficzna: Intel HD Graphics 3000  -płyta główna: Mobile Intel® HM65 Express chipset.  -karta sieciowa: 802.11b/g/n Wi-Fi CERTIFIED™, 10/100/1000 Gigabit Ethernet LAN  -karta dzwiękowa: Dolby® Advanced Home Theater® v4 Audio Enhancement,  -storage: SATA 3 - 6Gb/s  -RAM: 4GB  -dysk twardy: HDD, 320 GB, SATA |
| komp2 | desktop | Dell OptiPlex 790 | Xubuntu14.02 | - procesor: Intel Core i5 (2nd Gen) 2400 / 3.1 GHz  -karta graficzna: Integrated Intel® HD Graphics 2000  -płyta główna: Intel® H65 Express Chipset  -karta sieciowa: Intel® 82579LM Gigabit1 Ethernet LAN 10/100/1000  -karta dzwiękowa: Realtek ALC269Q High Definition Audio Codec  -storage: Serial ATA-600  -RAM: 1333 MHz  -dysk twardy: HDD, SATA 6Gb/s |
| komp3 | laptop | Dell Inspiron 17 5758 | Windows7 Professional | - procesor : Intel (R) Core (TM) i7-4800MQ CPU @ 2.70GHz (8 CPUs)  -karta graficzna: NVIDIA GeForce 920M  -płyta główna: 1000 GB HDD  -karta sieciowa: Intel® 82579LM Gigabit1 Ethernet LAN 10/100/1000  -karta dzwiękowa: Realtek ALC269Q High Definition Audio Codec  -storage: Serial ATA-600  -RAM: 8GB  -dysk twardy:1000 GB HDD |

Źródło: opracowanie własne

Jak można zauważyć, wszystkie powyższe procesory, z wyjątkiem Intel (R) Core (TM) i7-4800MQ CPU @ 2.70GHz (8 CPUs), wspierają wirtualizację.

**3.3. Wymagania i założenia programowe**

Głównym wymogiem programowym przy tworzeniu i konfiguracji maszyn wirtualnych z użyciem QEMU/KVM jest konieczność zainstalowania na maszynach gospodarzy dodatkowych rozszerzeń i narzędzi, służących do konfiguracji maszyn wirtualnych i działającej na nich sieci. Instalację można przeprowadzić z repozytorium danej dystrybucji.

Narzędzia pomocnicze, które yainstalowano to przede wszystkim:

• bridge-utils – narzędzie do tworzenia mostów sieciowych;

• QEMU/QEMU-KVM– QEMU z obsługą KVM;

• libvirt – daemon libvirt;

• virt-manager- aplikacja graficzna, która tworzy i zarządza maszynami wirtualnymi

przez libvirt.

Libvirt API jest domyślnym interfejsem do zarządzania maszynami KVM/QEMU w większości współczesnych dystrybucji GNU/Linux. Do obsługi maszyn używa wbudowanej konsoli *virsh* oraz przytoczonej już wcześniej aplikacji virt-manager.

Poza wymienionymi wyżej dodatkowymi narzędziami konfiguracyjnymi, do wykonania projektu użyto czterech różnych systemów operacyjnych. Na komputerach gospodarzach (komp1 i komp2) zainstalowano Xubuntu14.02. Z koleji na gościach; czyli maszynach wirtualnych virt1, virt2, virt3; użyto niewielkich pojemnościowo dystrybucji Slitaz40 oraz Damn Small Linux.

Ostatnia z maszyn fizycznych, komp3, działa na systemie operacyjnym *Windows7 Professional* i służy jako serwer HTTP z aplikacją webową do zarządzania zasobami, napisaną w języku JavaScript, przy użyciu platformy Bootstrap.

Konfiguracja systemu Xubuntu14.02. na gospodarzach jest identyczna, zarówno w przypadku maszyny komp1 jak i komp2.

Oprogramowanie zainstalowane i skonfigurowane na tym systemie to standardowe oprogramowanie z dystrybucji oraz wymienione juz wcześniej narzędzia dodatkowe.

Oprogramowanie standardowe obejmuje m.in.:

**Środowiska graficzne:**

• XFCE4;

• GIMP 2.8.10;

**Narzędzia programistyczne:**

• GCC 4.9.0;

• Eclipse 4.4;

**Dodatkowe oprogramowanie:**

• 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.

J eżeli chodzi o maszynę komp3 z systemem Windows7 Professional , posiada ona następujące oprogramowanie:

• 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 v2.

Z uwagi na to, że na maszynie komp3 działa aplikacja opierająca się o framework Bootstrap, na tym komputerze zainstalowano także następujące komponenty:

• Microsoft .NET Framework 1.1,

• Internet Explorer 6.0 SP1 lub wyższy,

• jQuery w wersji1;

• Bootstrap v3.3.5;

• AngularJS 1.4.1;

• Node Express 4.13.0.

**4. Charakterystyka poszczególnych elementów projektu**

**4.1. Charakterystyka sieci lokalnej i wirtualnej**

Wszystkie trzy maszyny fizyczne użyte w projekcie znajdują się w obrębie tej samej sieci lokalnej LAN i posiadają statyczne adresy IP, przypisane ręcznie. Do każdego IP przypisana jest określona nazwa komputera fizycznego- komp1, komp2, komp3.

Na laptopie z systemem operacyjnym Windows7 Professional (komp3) działa aplikacja webowa do zarządzania pulą maszyn wirtualnych na pozostalych dwóch komputerach.

Wszystkie trzy maszyny fizyczne mają otwarty port ssh, po którym zachodzi komunikacja między nimi.

Sieć opisaną w projekcie charakteryzują następujące parametry:

**• zakres IP dla hostów: 153.65.206.15– 153.65.206.17**

Tabela 2. Zestawienie adresów IP hostów

|  |  |
| --- | --- |
| **nazwa komputera** | **IP statyczne** |
| komp1 | 153.65.206.15 |
| komp2 | 153.65.206.16 |
| komp3 | 153.65.206.17 |

Źródło: opracowanie własne

**• zakres IP dla stworzonych maszyn wirtualnych: 153.65.206.18– 153.65.206.30**

Tabela 3. Zestawienie adresów IP maszyn wirtualnych

|  |  |  |
| --- | --- | --- |
| **nazwa komputera fizycznego** | **nazwa maszyny wirtualnej** | **IP maszyny wirtualnej** |
| **komp1** | **virt1** | **153.65.206.18– 153.65.206.23** |
| **virt2** |
| **virt3** |
| **komp2** | **virt1** |
| **virt2** |
| **virt3** |

Źródło: opracowanie własne

**• maska podsieci: 255.255.255.0**

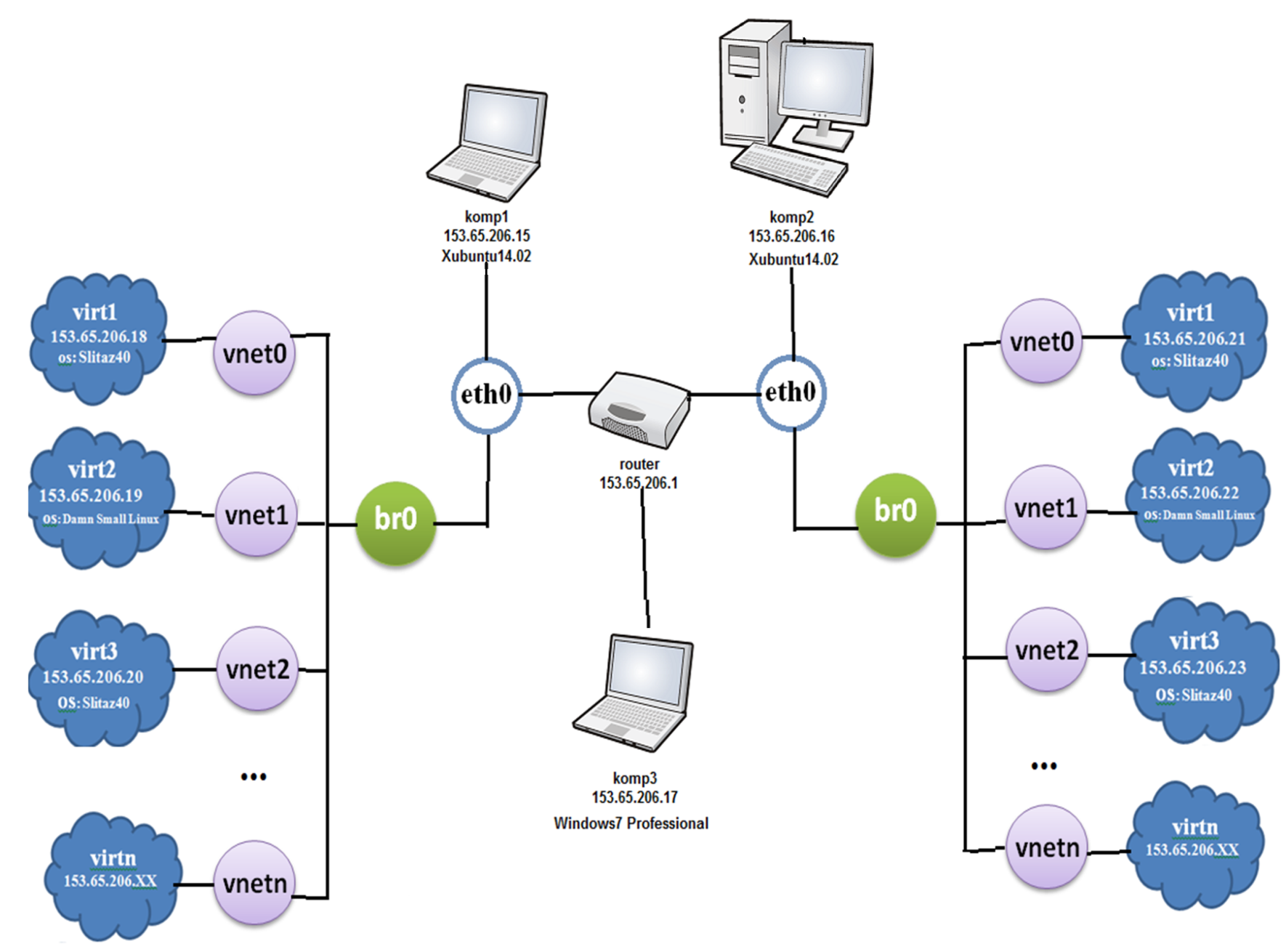
**• bama domyślna: 153.65.206.1**

**• broadcast: 153.65.206.255**

Jeżeli chodzi o maszyny wirtualne, utworzono je odpowiednio na komputerach komp1 i komp2. Każda z tych maszyn fizycznych posiada trzy maszyny wirtualne, nazwane kolejno virt1, virt2 i virt3.

Na fizycznym interfejsie gospodarza (eth0) postawiono most o nazwie br0. Most sieciowy przejmuje konfigurację sieci hosta. Dla każdej maszyny wirtualnej tworzony jest wirtualny interejs vnet0, vnet1 i vnet2 widoczny w systemie hosta. Most (br0) łączy ze sobą fizyczy interfejs gospodarza z wirtualnymi interfejsami gości,przez co maszyny wirtualne są w stanie wysyłać i odbierać pakiety przez interfejs hosta, nie są jednak w stanie komunikować się między sobą. Dodatkowo most został odpowiednio skonfigurowany, tak aby uruchamiał się przy starcie systemu.

Każda maszyna wirtualna posiada unikalny, wygenerowany automatycznie adres MAC wirtualnej karty sieciowej. Adresy IP także przydzielane są w sposób automatyczny, z dostępnej puli adresów przez serwer DHCP routera. Każda maszyna wirtualna ma przydzieloną rezerwację dla swojego adresu MAC, dzięki czemu otrzymuje z góy ustalony adres IP. Do adresów IP przypisane są określone nazwy maszyn wirtualnych- skonfigurowany jest DNS oraz revDNS.

Schemat budowy sieci zaimplementowanej w projekcie przedstawia poniższy rysunek:

**Rys10. Schemat budowy sieci projektu**

Źródło: opracowanie własne.

Na każdej maszynie wirtualnej; virt1, virt2 i virt3; zainstalowano niewielkie pojemnościowo systemy dystrybucji Slitaz40 oraz Damn Small Linux.

Poszczególnym maszynom wirtualnym przyporządkowano odpowiednio obrazy:

• virt1- Slitaz40,

• virt2- Damn Small Linux,

• virt3 - Slitaz40.

Do stworzenia maszyn wirtualnych na hostach użyto Virtual Managera (można także skorzystać z polecenia virt-install i qemu-img create).

Obrazy dostępnych systemów znajdują się w pliku **/var/lib/libvirt/images/**. Dyski wirtualnych maszyn zapisane są w formacie plików qcow2(ang.*copy-on-write*) ze względu na fakt, iż ten format przechowuje tylko zmiany w stosunku do bazowego obrazu, oszczedzając tym samym miejsce na dysku. Dodatkowo, ułatwia przywrócenie obrazu do określonego stanu, będąc tym samym alternatywą snapshota.

Konfiguracja zamontowanych maszyn wirtualnych przechowywana jest w pliku **XML** domeny, w katalogu /**etc/libvirt/qemu.** Właścicielem obydwu katalogów jest root, więc tylko on może dokonywać w nich modyfikacji.

Każda maszyna wirtualna została stworzona z takimi samymi parametrami, różnią się jedynie rodzajem zainstalowanego systemu operacyjnego. Przykładowy plik konfiguracyjny dla maszyny virt1 wygląda następująco:

<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>

Użytkownik może zalogować się do systemu używając konta administratora-użytkownika z odpowiednimi uprawnieniami (nazwa użytkownika:admin, haslo: admin) bądź też jako zwykły użytkownik (nazwa użytkownika: guest, hasło: guest).

**4.2. Architektura aplikacji do zarządzania zasobami wirtualnymi**

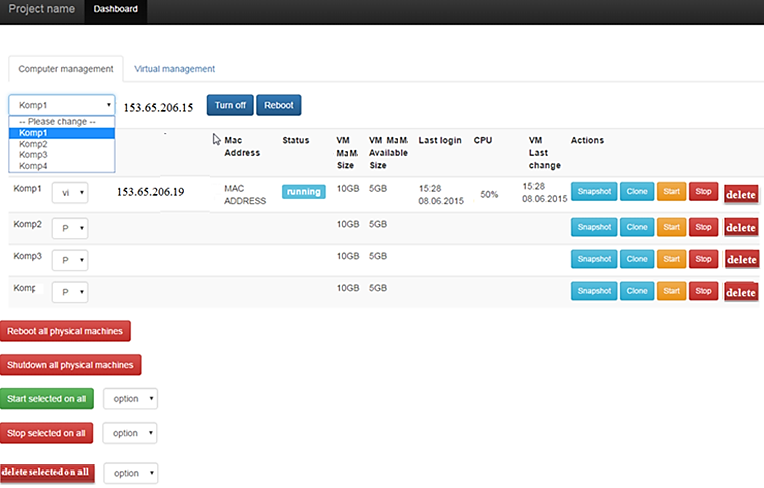
Aplikacja do zdalnego zarządzania wirtualnymi zasobami działa na komputerze komp3.

Podzielono ją na dwie części, każda z nich prezentowana jest w odrębnej zakładce strony.

Pierwsza część nosi nazwę ***Computer Management*** i odpowiada za dostarczanie informacji o maszynach wirtualnych działających na wybranych maszynach fizycznych.

Wygląd tej części przedstawiono poniżej:

**Rys11. Wygąd strony HTML zakładki *Computer Management***



Źródło: opracowanie własne

Użytkownik jest w stanie sprawdzić jakie maszyny wirtualne są dostępne na danym komputerze fizycznym i jakie są w danym momencie uruchomione. Lista rozwijana, znajdująca się obok nazwy komputera fizycznego, zawiera nazwy wszystkich dostępnych komputerów wirtualnych na danej maszynie. Po wybraniu nazwy wirtualnej maszyny, reszta dostępnych pól wypełni się automatycznie.

Wyświetlone zostaną informacje dotyczące:

• **IP** adresu maszyny wirtualnej;

• **MAC** adresu maszyny wirtualnej;

• **statusu** maszyny wirtualnej- dostępnych jest sześć rodzajów możliwych stanów maszyny wirtualnej, są to:

**→** **running**- kiedy maszyna wirtualna działa poprawnie i jest włączona;

**→** **blocked**- kiedy maszyna wirtualna znajduje się w stanie uśpienia lub czeka na I/O;

**→** **paused**- kiedy maszyna wirtualna jest wstrzymana przez administrator, nie zużywa wtedy zasobów CPU;

**→** **shutdown**- kiedy maszyna wirtualna jest wyłączona;

**→ dying**- kiedy maszyna wirtualna jest w trakcie zamykania system lub kiedy prawie przestała działać na skutek awarii;

**→** **crashed**- kiedy maszyna wirtualna nie jest dostępna, na skutek awarii systemu.

Warto zaznaczyć, iż aby stan *crashed* byl widoczny w odpowiedzi na komendę virsh list –all, dana maszyna wirtualna musi posiadać w pliku konfiguracyjnym XML ustawienia, mówiące o tym, iż w przypadku uszkodzenia maszyny nastąpi jej restart.

Fragment, kóry definiuje ten parametr, wygląda następująco:

…

</features>

<clock offset='utc'/>

<on\_poweroff>destroy</on\_poweroff>

<on\_reboot>restart</on\_reboot>

**<on\_crash>restart</on\_crash>**

<devices>

…

• aktualnego rozmiar pamięci wirtualnej maszyny;

• dostępny rozmiar pamięci maszyny wirtualnej;

• data ostatniego logowania do systemu maszyny wirtualnej;

• rozmiar CPU maszyny wirtualnej;

• data ostatniej modyfikacji konfiguracji maszyny wirtualnej.

Dostępne są także grupy przyciskiów *Actions*, odpowiadające za wykonywanie poszczególnych akcji na wirtualnej maszynie. Są to kolejno:

• snapshot,

• clone,

• start,

• stop,

• delete.

Do kontrolowania maszyn wirtualnych służą także dwa przyciski na dole strony:

• ***Start selected on all***- po wybraniu nazwy maszyny wirtualnej ze znajdujacej się obok drop-down listy i następnie wciśnięciu przycisku, dana maszyna wirtualna zostanie uruchomiona na wszystkich hostach, na których jest dostępna;

• ***Stop selected on all*-** po wybraniu nazwy maszyny wirtualnej ze znajdujacej się obok drop-down listy i następnie wciśnięciu przycisku, dana maszyna wirtualna zostanie wyłączona na wszystkich hostach, na których jest dostępna;

• ***Delete selected on all*-** po wybraniu nazwy maszyny wirtualnej ze znajdujacej się obok drop-down listy i następnie wciśnięciu przycisku, dana maszyna wirtualna zostanie usunięta z wszystkich hostów, na których jest dostępna.

Z tej części aplikacji możliwe jest także wykonanie określonych czynności na maszynach fizycznych. Służą do tego przyciski:

◼ ***Reboot all physical machines***- restart wszystkich maszyn fizycznych;

◼ ***Shutdown all physical machines***- wyłączenie wszystkich maszyn fizycznych.

oraz ***Turn off*** i ***Reboot***, dostępne przy wyborze nazwy maszyny fizycznej. Pierwszy z nich odpowiada za wyłączenie wybranej maszyny fizycznej, drugi zaś ją restartuje.

Druga część aplikacji; o nazwie ***Virtual Management***; umożliwia sprawdzenie jakie obrazy maszyn wirtualnych uruchomione są na danym komputerze fizycznym. Dla przejrzystości, umieszczono ją w odrębnej zakładce strony:

**Rys 12. Wygąd strony HTML zakładki *Virtual Management***



Źródło: opracowanie własne

W momencie wyboru nazwy obrazu maszyny wirtualnej z listy rozwijanej, reszta dostępnych pól wypełnia się automatycznie, dostarczając danych odnośnie:

◼ listy maszyn fizycznych, na których dostępny jest dany obraz;

◼ status danego obrazu na konkretnej maszynie fizycznej w danym momencie;

◼ data ostatniego logowania do systemu operacyjnego maszyny wirtualnej na danym

komputerze fizycznym;

◼ IP maszyny wirtualnej;

◼ rozmiar CPU maszyny wirtualnej.

Co do interfejsu graficznego aplikacji, stworzono go przy użyciu front-endowego frameworku Bootstrap v3 używającego języka JavaScript, a część back-endową aplikacji wykonano wykorzystując framework AngularJS v1, NodeJS oraz skrypt napisany w Perlu, odpowiedzialny za konsolidację danych oraz połączenie ich z odpowiednimi elementami widoków.

Ponadto, przy tworzeniu aplikacji, korzystano z następujących technologii:

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

Jeżeli chodzi o strukturę katalogów aplikacji, folder główny projektu *Virtual Dashboard* ma następującą budowę:

**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.gzip

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** /

glyphicons-halflings-regular.eot

glyphicons-halflings-regular.svg

glyphicons-halflings-regular.ttf

glyphicons-halflings-regular.woff

glyphicons-halflings-regular.woff2

└───**js** /

bootstrap.js

bootstrap.min.js

npm.js

**└───fonts /**

glyphicons-halflings-regular.eot

glyphicons-halflings-regular.svg

glyphicons-halflings-regular.ttf

glyphicons-halflings-regular.woff

glyphicons-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

Najważniejsze pliki, odpowiadające za poprawne działanie aplikacji możemy podzielić na:

**API:**

• **app/management.js** – zawiera metody do zarządzania maszynami wirtualnymi;

- shutdown - wyłączenie;

- getVirts - pobranie listy maszyn;

- findComputerbyIP - na podstawie listy komputerów pobierz ich IP;

• **app/pc-list.js** - wpisana na sztywno lista IP komputerów;

• **app/ssh.js** – uwierzytelnianie połączenia ssh.

**AngularJS:**

• **/public/app/app.js** - deklaracja modułu;

• **/public/app/app.routes.js** - deklaracja routingu dla stron;

• **/public/app/controllers/dashboardCtrl.js** - kontroler do obsługi ekranu głównego;

•/**public/app/services/computersService.js -** serwis do pobrania z api listy komputerów;

•/**public/app/views/index.html** – strona w HTML, zakładka *Computer Management*;

•/**public/app/views/index\_new.html** – strona w HTML, zakładka *Virtual Management*.

**5. Implementacja projektu**

**5.1.**  **Konfiguracja hostów**

W opisanym poniżej rozdziale przedstawiono w jaki sposób przebiegała konfiguracja maszyn fizycznych, będących hostami dla maszyn wirtualnych. Opisano instalację wymaganych komponentów oraz specyfikację konfiguracji sieci, w tym przypadku posłużono się mostkiem sieciowym.

**5.1.1. Przygotowanie środowiska i instalacja dodatkowych narzędzi**

Przed rozpoczęciem tworzenia maszyn wirtualnych , należy sprawdzić, czy maszyna fizyczna wspiera wirtualizację.Weryfikacji tej można dokonać używając poniższej komendy:

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

W przypadku, kiedy komputer posiada takie wsparcie, wynik powinien być zbliżony do poniższego:

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 ds\_cpl vmx est tm2 ssse3 cx16 xtpr pdcm lahf\_lm tpr\_shadow

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 ds\_cpl vmx est tm2 ssse3 cx16 xtpr pdcm lahf\_lm tpr\_shadow

Gdy mamy pewność, iż maszyna posiada takie wsparcie, jednak komenda po wykonaniu nie wypisała żadnych flag, należy sprawdzić, czy wirtualizacja nie jest wyłączona w BIOSie.

Następnym krokiem jest instalacja odpowiednich pakietów i narzędzi z dystrybucji.

W tym celu wykonujemy polecenie:

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

Po ich pomyślnym zainstalowaniu należy dodać użytkownika do grup libvirtd i kvm, w tym celu wykonujemy polecenie:

sudo adduser [user\_id] libvirtd

sudo adduser [user\_id] kvm

oraz uruchomić deamona libvirt i dodać go do autostartu:

sudo service libvirt-bin start

sudo update-rc.d libvirt-bin defaults

**5.1.2.**  **Tworzenie mostu sieciowego**

Most sieciowy tworzy się na hoście następującym poleceniem:

sudo brctl addbr br0

Następnie przystępujemy do jego konfiguracji, tak aby był on uruchamiany przy starcie systemu. Należy w tym celu wyłączyć Network Managera oraz odpowiednio zedytować plik o nazwie **/etc/network/interfaces**.

W przypadku konfiguracji sieci w projekcie, czyli z użyciem DHCP, plik ten po zedytowaniu powinien wyglądać następująco:

auto lo

iface lo inet loopback

iface eth0 inet manual

iface br0 inet dhcp

bridge\_ports eth0

Jeśli chodzi o konfigurację statyczną, plik miałby taką postać:

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

Po skonfigurowaniu mostu, należy wyłączyć NetworkManager i dokonać restartu sieci. Wydajemy kolejno polecenia:

sudo service network-manager stop

sudo update-rc.d netwok-manager remove

sudo service networking start

sudo update-rc.d networking defaults

Poprawność konfiguracji mostu można sprawdzić za pomocą:

sudo brctl show

Komenda powinna zwrócić następującą odpowiedź:

bridge name bridge id STP enabled interfaces

br0 8000.0200c0a80091 no eth0

**5.2.**  **Konfiguracja gości**

W poniższym rozdziale opisano w jaki sposób budowane są maszyny wirtualne działające na hostach oraz w jakich narzędzi można używać w celu zarządzania tymi maszynami.

Na początku projektu, na każdej z dwóch dostępnych maszyn fizycznych (komp1 i komp2) utworzono po trzy maszyny wirtualne, którym nadano kolejno nazwy virt1, virt2 i virt3. W miarę zagłębiania się w temat wirtualizacji i odkrywania złożoności tego zagadnienia,koniecznym stało się stworzenie większej ilości maszyn, aby móc zapisać na nich konfiguracje, użyte do testowania nowych właściwości. Pojawił sie zatem problem przechowywania obrazów nowych maszyn oraz problem szybkiej instalacji takich samych obrazów. W związku z tym, zdecydowano się poszerzyć pulę dostępnych maszyn, korzystając przy tym z właściwości QEMU, jaką jest pochodność obrazów.

Początkowe maszyny wirtualne; virt1, virt2 i virt3; stały się zatem obrazami bazowymi (ang. *base images*) z czystym systemem operacyjnym. Zapisano je w formacie qcow2.

Na podstawie każdego obrazu bazowego, stworzono obraz pochodny od niego, używając do tego celu polecenia qemu-img create. Przykład tego polecenia dla maszyny virt2:

**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

Stworzona maszyna virt2-s0.qcow2 ma takie same parametry jak maszyna bazowa. Właściwości maszyny bazowej spraw dzić możemy poleceniem:

**qemu-img info** virt2.qcow2

image: virt2.qcow2

file format: qcow2

virtual size: 5.0G (5368709120 bytes)

disk size: 5.0G

cluster\_size: 65536

Tak samo możemy sprawdzić stan obrazu pochodnego:

**qemu-img info** virt2-s0.qcow2

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

Następnie, aby wystartować maszynę pochodną, należy uprzednio skopiować plik konfiguracyjny XML maszyny bazowej, przeedytować go, zmieniając nazwę, uuid, ścieżkę dysku, MAC adres, a potem za pomocą komendy virsh zdefiniować i wystartować maszynę:

**virsh define** virt2-s0.xml

**virsh start** virt2-s0 --console

**virsh list**

Id Name State

----------------------------------------------------

9 virt2-s0 running

Po uruchomieniu możemy sprawdzić rozmiar dysku nowo utworzonej maszyny:

**ls -lash** virt2-s0.qcow2

14M -rw-r--r--. 1 root root 14M Oct 4 06:30 virt2-s0.qcow2

Rozmiar to tylko 14Mb, co świadczy o tym, iż tylko zmiany w stosunku do maszyny bazowej będą zapisywane na nowym dysku.

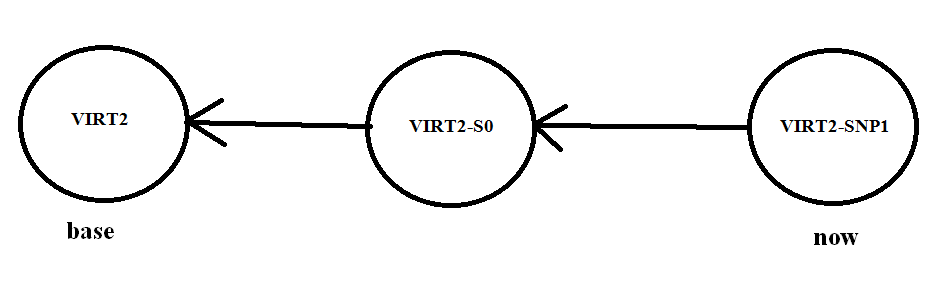
W efekcie, stworzony został tzw.obraz delta (ang.*delta image*) , przechowujący jedynie zmiany względem obrazu, na którym bazuje. Rozmiar ten będzie rósł w miarę zapisywania na nim nowych zmian.

Obraz virt2.qcow2 jest naszym obrazem bazowym. Od tej pory, przy testowaniu nowych rozwiązań, nigdy już nie startujemy tej maszyny, a zawsze uruchamiamy virt2-s0.qcow2 jako maszynę wyjściową.

Jeżeli chcielibyśmy wprowadzić jakieś zmiany do aktualnie uzywanego obrazu virt2-s0.qcow2, to zmiany te zapisujemy tworząc snapshot używanego obrazu, polecenie: **qemu-img snapshot -c <snapshot-name> <imagename>.**

Po jego stworzeniu, zależność naszych obrazów wyglądać będzie następująco:

**Rys13.Zależność obrazów pochodnych w QEMU**



Źródło: opracowanie własne

Obraz virt2 stanowi bazę dla obrazu virt2-s0, natomiast obraz virt2-s0 jest bazą dla virt2-snp1. Aktualnym obrazem jest virt2-snp1,a każda następna zmiana jaką będziemy chcieli dokonać na aktualnie obowiązującym obrazie, będzie wiązała się z koniecznością stworzenia nowego snapshota, bazującego na aktualnym obrazie. Nowy obraz będzie przechowywał różnice względem poprzedniego obrazu.

Do zarządzania snapshotami możemy użyć narzędzi qemu-img oraz virsh.

Najważniejsze operacje tych narzędzi przedstawia poniższa tabelka:

**Tabela4. Zarządzanie obrazami współdzielonymi za pomocą qemu-img i virsh.**

|  |  |  |  |
| --- | --- | --- | --- |
| **QEMU-IMG** | | **VIRSH** | |
| **snapshot** | tworzenie/zarządzanie tzw.*offline disk snapshots* | **snapshot-create-as** | tworzenie tzw.*internal/external snapshots* |
| **create** | tworzenie tzw.*offline external snapshots* | **blockcopy** | kopiowanie łańcucha obrazów |
| **commit** | aplikowanie zmian od najwcześniejszego obrazu pochodnego do bazowego | **blockcommit** | scalanie łańcucha od najwcześniejszych zmian do obrazu bazowego |
| **rebase** | kopiowanie zawartości obrazu bazy do obrazu pochodnego | **blockpull** | scalanie łańcucha od obrazu bazowego do obrazu pochodnego |

Źródło: opracowanie własne

**5.2.1.**  **Tworzenie maszyny wirtualnej**

Maszynę wirtualną w systemie operacyjnym hosta można tworzyć także za pomocą Virtual Managera oraz z linii komend z użyciem libvirt i qemu-img create.

Zarówno w jednym, jak i w drugim przypadku, konfigurację należy zacząć od pobrania obrazu systemu i skopiowanie go do odpowiedniego folderu na hoście, zazwyczaj jest to folder **/var/lib/libvirt/images/.**

Następnie, dodajemy dysk wirtualny na hoście:

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

Polecenie to tworzy pusty dysk obrazu typu qcow2, o pojemności 10G. Zaletą użycia formatu .qcow2 jest to, że dysk nie jest tworzony początkowo w pełnym rozmiarze, lecz rośnie w miarę pobierania danych.

Kiedy mamy juz dysk, instalujemy maszynę wirtualną, poniższym poleceniem :

**virt-install** -n virt1 --virt-type kvm --description "Slitaz40" --ram=1536 --vcpus=1 --cpu host --os-variant=ubuntuoneiric --accelerate –hvm --network bridge:br0,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** jest narzędziem libvirt API, służącym do tworzenia maszyny wirtualnej.

Narzędzie to przyjmuje następujące parametry:

**-r**- charakteryzuje wielkość RAM (w MB);

**--accelerate**-wskazuje na użycie *hardware acceleration*;

**-n-** definiuje nazwe maszyny wirtualnej;

**-f-** wskazuje na pełną ścieżkę to obrazu dysku maszyny;

**--cdrom**- jest ścieżką do ISO (nazwą obrazu ISO), który ściągnięto. Parametr ten jest potrzebny jedynie podczas instalacji, po jej zakończeniu może zostać usunięty;

**--vcpus=N**- określa gościa SMP z N wirtualnym CPUs;

**--description**- wskazuje na opis maszyny wirtualnej, który będzie umieszczony w jej pliku XML;

**-l**- używany, podczas instalacji maszyny z obrazu na URL;

**--disk**-określa format dysku maszyny, np.:qcow2 lub raw;

**-- soundw ac97**- tworzy gościa z wirtualna kartą dzwiękową AC’97; bez tej opcji żadna karta dzwiękowa nie zostanie przypisana

Innym narzędziem libvirt, które możemy wykorzystać przy tworzeniu maszyn wirtualnych jest **virt-image**. Polecenie to tworzy nową maszynę na podstawie jej konfiguracji zapisanej w pliku XML.

Jeżeli chodzi o tworzenie maszyn wirtualnych za pomocą Virtual Manager, sposób ten jest znacznie prostszy, gdyż program posiada graficzny interfejs. Tworzenie maszyny wirtualnej z użyciem Virtual Managera przedstawiają zrzuty z ekranu zamieszczone poniżej:

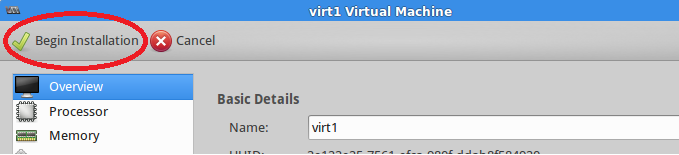
**Rys14. Tworzenie maszyny wirtualnej za pomocą Virtual Managera**

|  |  |
| --- | --- |
|  | 1. Ustalamy nazwę VM i lokalizację obrazu systemu (wybieramy *Local Install Media).* |
|  | 2. Wskazujemy na lokalizację systemu operacyjnego, który chcemy na maszynie zainstalować. |
|  | 3. Ustalamy wielkość pamięci RAM i ilość CPU. |
|  | 4. Ustalamy wielkość dysku maszyny wirtualnej. |
|  | 5. Przechodzimy do zaawansowanych konfiguracji maszyny. |
|  | 6. Ustalamy szczegółowego parametry sprzętu maszyny wirtualnej. |

Źródło: opracowanie własne

Po wybraniu wszystkich parametrów wirtualnej maszyny, końcowym etapem jest rozpoczęcie instalacji systemu. W tym celu klikamy przycisk ***Begin Installation***, który znajduje się w lewym górnym logu konfiguratora:

**Rys15.Instalowanie maszyny wirtualnej przy pomocy Virtual Managera**



Źródło: opracowanie własne

Instalacja przebiega w sposób automatyczny, bez potrzeby ingerencji użytkownika.

**5.2.2.**  **Zarządzanie maszyną wirtualną gościa**

Oprócz Virtal Managera, do zarządzania maszyną wirtualną gościa można użyć jeszcze wielu innych narzędzi. Obecnie najpopularniejszymi wśród nich są:

› libvirt API,

› QEMU,

› libguestfs,

› virt-v2v,

› sVirt,

› virt-\*,

› boxgrinder,

› spice.

Biblioteka **libvirt**, napisana w języku C, jest ciekawym i rozbudowanym projektem umożliwiającym zarządzanie wirtualnymi maszynami. Umożliwia sterowanie takimi procesami jak: uruchamianie maszyny wirtualnej, klonowanie, tworzenie interfejsów sieciowych, zwiększenie RAM, CPU.

Dodatkowym atutem biblioteki jest fakt, iż jej API ma charakter otwarty, co za tym idzie dostarcza jednolity interfejs dla programistów różnych języków, oferując tym samym możliwość przyczynienia się do rozwoju API. Wspieranymi obecnie językami są:

♦ 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/

Minusem stosowania libvirtu jest fakt, iż biblioteka nie wspiera wszystkich rodzajów monitorów. Aktualnie oferuje wsparcie dla poniższych rodzajów:

  • Hypervisor Xen na hostach Linux i Solaris;

• QEMU;

  • KVM;

  • LXC;

  • OpenVZ;

  • VirtualBox ;

  • VMware ESX i GSX;

  • VMware Workstation/Player;

  • User Mode Linux paravirtualized kernel ;

• Storage na dyskach IDE/SCSI/USB, FibreChannel, LVM, iSCSI i NFS.

Kolejną jej wadą jest brak automatyzacji; zarządzanie może odbywać się jedynie z poziomu jednego węzła a nie wielu jak w przypadku load balancingu.

Narzędzia libvirt to głównie polecenia wydawane z linii komend.

Jedną z komend libvirt jest polecenie **virsh**, które jest w stanie zarządzać konfiguracją następujących parametrów wirtualnej maszyny:

**→ zwiększanie/zmniejszanie pamięci maszyny wirtualnej (RAM)**

Do zarządzania pamięcią maszyny wirtualnej możemy użyć dwóch poleceń komendy virsh, które przedstawia poniższa tabelka:

**Tabela5.Polecenia virsh edytujące stan pamięci VM**

|  |  |  |
| --- | --- | --- |
| **polecenie edytujące stan pamięci VM** | **funkcja polecenia** | **polecenie zwracające stan pamięci danej VM** |
| **•** **virsh edit** < VM\_name, id\_VM> | pozwala na edytowanie pliku konfiguracyjnego XML danej maszyny | • **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> | definiuje rozmiar pamięci używanej |
| • **virsh setmaxmen** <VM\_name, id\_VM> <count\_in\_kilobytes> | ustawia maksymalny dostępny limit pamięci dla nadzorcy |  |

Źródło: opracowanie własne

Aby zwiekszyć pamięć konkretnej maszyny wirtualnej, należy wykonać następujące czynności:

• zamknąć wirtualną maszynę poleceniem **virsh shutdown** <VM\_name>:

virsh shutdown virt1

Domain virt1 is being shutdown

• edytować parametry maszyny wirtualnej za pomocą komendy **virsh edit** <VM\_name>:

virsh edit virt1

• w wyświetlonym pliku konfiguracyjnym XML należy odszukać zapis odnoszący się do rozmiaru pamięci maszyny wirtualnej i dowolnie go edytować:

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

• zapisujemy wprowadzone zmiany, znów używając **virsh edit** <VM\_name>:

virsh edit virt1

Domain virt1 XML configuration edited.

• konieczy jest jeszcze restart wirtualnej maszyny, poleceniem **virsh reboot** <VM\_name>:

virsh restart virt1

**→ zwiększanie/zmniejszanie wielkości procesora maszyny wirtualnej (CPU)**

Za zarządzanie procesorem maszyny wirtualnej odpowiadają następujące komendy:

**Tabela6. Polecenia virsh edytujące wielkość procesora VM**

|  |  |  |
| --- | --- | --- |
| **polecenie edytujące wielkość procesora VM** | **funkcja polecenia** | **polecenie zwracające stan procesora danej VM** |
| **•** **virsh edit** < VM\_name, id\_VM> | pozwala na edytowanie pliku konfiguracyjnego XML danej maszyny | • **virsh dominfo** < VM\_name, id\_VM> **| grep –i cpu**  CPU(s): 4  CPU time: 22.1s  • **virsh vcpuinfo** < VM\_name, id\_VM>  VCPU: 1  CPU: 4  State: running  CPU time: 22192.9s  CPU Affinity: yyyyyyyy |
| • **virsh vcpus** < VM\_name, id\_VM> <number\_vcpus> | definiuje ilość vcpus dla danej maszyny |

Źródłó:opracowanie własne

W przypadku edytowania CPU maszyny wirtualnej za pomocą zmiany konfiguracji pliku XML komendą **virsh edit**, postepujemy w analogiczny sposób jak w poprzednim przykładzie. Jedyną zmianą jest fakt, iż edytujemy fragment pliku odpowiadający za konfigurację vCPU:

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

**→ modyfikowanie przestrzeni dyskowej maszyny wirtualnej**

Aby dodać nowy dysk wirtualny do maszyny, najpierw należy stworzyć taki dysk, a następnie dodać go do przestrzeni dyskowej wirtualnej maszyny.

Nowy dysk zazwyczaj tworzy się w folderze **/var/lib/libvirt/images/,** używając do tego celu polecenia **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

Następnie, należy dodać go do istniejącej już maszyny, używając komendy **virsh attach-disk**:

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

Disk attached successfully

Powyższe polecenie korzysta z następujących parametrów:

**-virt1= <vm\_name>** - nazwa wirtualnej maszyny;

**-source-** po nim wpisana jestpełna ścieżka dostępu do dołączanego dysku;

**-target-** miejsce, gdzie chcemy zamontować ten dysk;

**-persistent-** opisuje w jaki sposób dysk ma być dodany do wirtualnej maszyny, może przyjąć także parametr *cache*.

W celu usunięcia dysku z wirtualnej maszyny, wydajemy komendę **virsh detach-disk** :

virsh detach-disk virt1 vdb

Disk detached successfully

Weryfikację poprawności konfiguracji można sprawdzić za pomocą:

fdisk -l | grep ‘vd’

**→ dodawanie wirtualnego CD-ROM/DVD do maszyny wirtualnej**

Używamy do tego celu polecenia **virsh attach-disk** z odpowiednimi parametrami:

**virsh attach-disk** <VM\_name> <source> <target> --driver file --type cdrom --mode readonly

**→ dodawanie urządzeń zdefiniowanych w pliku XML do maszyny wirtualnej**

Za pomocą komendy virsh możemy dodawać także do maszyny wirtualnej gościa urządzenia zdefiniowane przez plik konfiguracyjny XML.

Poniższy przykład przedstawia dodanie pliku ISO jako HDC do działającej maszyny gościa virt1. W tym celu najpierw zdefiniowano urządzenie za pomocą pliku konfiguracyjnego XML:

cat virt1iso.xml

<disk type="file" device="disk">

<driver name="file"/>

<source file="/var/lib/libvirt/virt1dysk. iso"/>

<target dev="hdc"/>

<readonly/>

</disk>

Następnie, za pomocą polecenia **virsh attach-device** dodajemy utworzony plik do gościa komp1:

**virsh attach-device** virt1 virt1iso.xml

**→ tworzenie snapshotów maszyny wirtualnej**

Jak już wcześniej wspomniano, snapshoty VM, odzwierciedlają obraz całego środowiska, w którym pracuje maszyna wirtualna- obejmują stan procesora (CPU), stan pamięci (RAM), emulowanych urządzeń i stany wszystkich zapisywalnych dysków.

Istnieje kilka typów snapshotów wykonywanych na maszynie wirtualnej:

**1. wewnętrzny (ang.*internal*)** - pojedyńczy obraz qcow2 przechowywuje zarówno obraz wzorcowy jak i zmiany jakie na nim wprowadzono. Jest to pojedyńczy obraz, gdzie znajdują się wszystkie informacje o wirtualnej maszynie w danym momencie. Tego typu snapshoty wykonują się dość długo (szczególnie kiedy maszyna wirtualna jest włączona) i obsługują tylko format plików qcow2, przez co nie są aktualizowane przez QEMU.

Wyróżniamy tu dwie podgrupy snapshotów wewnętrznych:

**►wewnętrzny zrzut dysku**- (ang**.***internal 'disk snapshot'*)-- {live/offline}

**Odzwierciedla stan wirtualnego dysku w danym momencie. Zarówno obraz wyjściowy jak i wprowadzone zmiany przechowywane są w tym samym pliku qcow2. Zrzut można wykonać przy włączonej lub wyłączonej maszynie gościa.**

***►* wewnętrzny zrzut typu ‘punkt kontrolny’** **– (ang. *internal 'system checkpoint' snapshot* ) -- {live}**

Jest wykonywanykiedy maszyna wirtualna jest włączona.Zapisywany jest stan maszyny wirtualnej i dysków,przy czym na stan maszyny wirtualnej składa się stan jej pamięci i emulowanych urządzeń (bez stanu jej dysku).

Przed stworzeniem zrzutu maszyny wirtualnej, należy przygotować plik xml dla tego zrzutu:

cat /var/tmp/virt1-snp1.xml

<domainsnapshot>

<name>virt1-snp1 </name>

<description>testowy snapshot virt1 </description>

</domainsnapshot>

Następnie można przystąpic do wykonania zrzutu kiedy maszyna jest włączona:

**virsh snapshot-create virt1 /var/tmp/virt1-snp1.xml**

Domain snapshot virt1-snp1pki created from '/var/tmp/snap1-komp1.xml'

W momencie wykonywania snapshotu, maszyna gościa zostanie zapauzowana i znów uruchomiona po zakończeniu procesu.

Aby sprawdzić, czy migawka została wykonana poprawnie, sprawdzamy, czy pojawiła się na liście dostępnych snapshotów dla danej maszyny:

**virsh snapshot-list virt1**

Name Creation Time State

------------------------------------------------------------

virt1-snp1pki 2011-10-04 19:04:00 +0530 running

Do tego celu możemy uzyć także polecenia **qemu-img info**, które także wyświetli informacje o powstałych zrzutach dla danego gościa:

**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

W celu przywrócenia wykonanego zrzutu stosuje się polecenie **virsh snapshot-revert <domain snapshotname>.**

**2.zewnętrzny (ang.*external*)** - kiedy plik wzorcowy występuje w formacie „tylko do odczytu” a nowy wygenerowany plik jest deltą zmian. Taki snapshot przechowywuje jedynie zmiany w stosunku do obrazu wzorcowego. Ten typ jest szczególnie przydatny w przypadku backupów. Nie wymaga wyłączenia maszyny gościa. Zrzuty zewnetrzne obsługuja wszystkie rodzaje formatów dysku- raw, qcow2, etc.

Podobnie jak w przypadku zrzutów wewnetrznych, także rozróżniamy tu dwa podtypy:

**►zewnętrzny zrzut dysku- (ang.*external 'disk snapshot'*)-- {live/offline}**

Snapshot dysku jest zapisywany w jednym pliku, a wprowadzane zmiany w nowym pliku qcow2.

**► zewnętrzny zrzut typu ‘punkt kontrolny’ – (ang. external 'system checkpoint' snapshot ) -- {live}**

Obraz stanu dysku maszyny wirtualnej jest zapisywany w jednym pliku, a jej RAM i stan emulowanych urządzeń w innym.

**3**. **stan wirtualnej maszyny**- występuje kiedy zapisujemy stan maszyny wirtualnej w danym momencie do pliku. Zapisywany jest jedynie stan VM, bez stanu dysków.

Konfigurację wirtualnej maszyny możemy zapisać kopiując jej plik konfiguracyjny XML. Służy do tego polecenie **virsh dumpxml**:

virsh dumpxml virt1 > VIRT1.xml

ls

virt1.xml

VIRT1.xml

Z powstałego pliku konfiguracyjnego (tutaj VIRT1) można stworzyć obraz istniejącej juz maszyny:

virsh create VIRT1.xml

Pełną listę dostępnych opcji komendy virsh stosowanych w przypadku snapshotów przedstawia poniższa tabelka:

**Tabela6.Polecenia virsh dotyczące snapshotów.**

| **polecenie** | **opis** |
| --- | --- |
| **snapshot-create <domain> <xml\_file>** | tworzy snapshot |
| **snapshot-current <domain>** | pokazuje plik XML obecnego snapshotu dla danej maszyny wirtualnej |
| **snapshot-delete** | usuwa snapshot |
| **snapshot-dumpxml <domain> <snapshot\_name>** | pokazuje plik XML snapshotu o danej nazwie dla wybranej domeny |
| **snapshot-list <domain>** | pokazuje dostępne snapshoty |
| **snapshot-revert <domain> <snapshot\_name>** | przywraca domenę do danego snapshotu |
| **snapshot-create-as** | tworzy snapshot z określonymi parametrami |
| **snapshot-parent <snapshot\_name>** | pokazuje nazwę maszyny wzorcowej snapshotu |

Źródło: opracowanie własne.

Inne przydatne komendy polecenia **virsh**:

**Tabela7. Polecenia komendy virsh dotyczące wirtualizacji.**

|  |  |
| --- | --- |
| **polecenie** | **rola** |
| **Domain Management** | |
| **virsh list ‐‐all** | lista wszystkich maszyn wirtualnych ze stanem |
| **virsh list ‐‐inactive** | lista nieaktywnych maszyn wirtualnych |
| **virsh start <nazwa\_VM, id\_VM>** | start wirtualnej maszyny |
| **virsh shutdown <nazwa\_VM, id\_VM>** | stop wirtualnej maszyny, zamknięcie |
| **virsh reboot <nazwa\_VM, id\_VM>** | restart wirtualnej maszyny |
| **virsh destroy <nazwa\_VM, id\_VM>** | wymuszone zatrzymanie wirtualnej maszyny, ­ (ang.force stop ) |
| **virsh suspend <nazwa\_VM, id\_VM>** | zatrzymanie (pauzowanie) wirtualnej maszyny |
| **virsh resume <nazwa\_VM, id\_VM>** | wznowienie pracy wirtualnej maszyny |
| **virsh dumpxml <nazwa\_VM, id\_VM>** | wyświetlenie pliku konfiguracyjnego wirtualnej maszyny |
| **virsh define <nazwa\_VM, id\_VM>** | tworzy plik konfiguracyjny xml wirtualnej maszyny |
| **virsh save <nazwa\_VM, id\_VM> <nazwa\_pliku>** | zapisywanie stanu wirtualnej maszyny do pliku |
| **virsh restore <nazwa\_pliku>** | przywracanie zapisanej wcześniej wirtualnej maszyny |
| **Domain Monitoring** | |
| **domblkinfo** | informacja o rozmiarze urządzeń blokowych domeny |
| **domblklist** | lista bloków domeny |
| **domblkstat** | wyświetla statystyki dla urządzeń blokowych domeny |
| **domif-getlink** | wyświetla stan interfejsu wirtualnego |
| **dominfo** | wyswietla informacje o domenie |
| **dommemstat** | wyświetla statystyki pamięci dla domeny |
| **domstate** | wyświetla stan domeny |
| **list** | wyświetla listę domen |
| **Host i Hyperwisor** | |
| **hostname** | wyświetla nazwę hipernadzorcy |
| **nodecpustats** | wyświetla statystyki cpu dla danego węzła |
| **nodememstats** | wyświetla statystyki pamięci dla danego węzła |
| **sysinfo** | wyświetla informacje o hipernadzorcy systemu |
| **nodeinfo** | wyświetla informacje o węźle |
| **Interface** | |
| **iface-start** | uruchamia fizyczny interfejs hosta (aktywuje go / "if-up") |
| **iface-destroy** | dezaktywuje fizyczny interfejs hosta (dezaktywuje go / "if-down") |
| **iface-undefine** | usuwa fizyczny interfejs hosta (usuwa go z konfiguracji) |
| **iface-list** | wyswietla liste interfejsów fizycznych |
| **iface-bridge** | tworzy mostek i dołącza go do istniejącego urządzenia sieciowego |
| **iface-unbridge** | usuwa mostek |
| **Networking** | |
| **net-create** | tworzenie interfejsu wirtualnego |
| **net-define** | twiorzenie danego interfejsu sieci wirtualnej |
| **net-destroy** | kasowanie interfejsu wirtualnego |
| **net-dumpxml** | wyświetlenie pliku konfiguracyjnego konkretnej wirtualnej sieci |
| **net-info** | wyświetlenie informacji o danej sieci wirtualnej |
| **net-list --all** | wyświetlenie informacji o wszystkich sieciach wirtualnych |
| **net-start** | aktywacja danego interfejsu sieci wirtualnej |
| **net-undefine** | dezaktywacja danego interfejsu sieci wirtualnej |

Oprócz komendy virsh, do zarządzania maszyną wirtualną możemy użyć także innych narzędzi libvirt. Poniżej przedstawiono najczęściej z nich używane:

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

Jest narzędziem służącym do klonowania obrazu maszyny wirtualnej. Virt-clone kopiuje obraz dysku wirtualnej maszyny gościa i na jego podstawie definiuje nową z identycznymi zasobami jak maszyna wzorcowa. Zmieniane są jedyne poniższe atrybuty definiujące wirtualną maszynę:

• nazwa komputera;

• ścieżka dostępu do dysku;

• IP maszyny;

• UUID;

•MAC address.

► **virt-top**

Narzędzie zbiera dane na temat wirtualnych maszyn na danym gościu, takie jak stan, zużycie CPU, RAM i prezentuje w formie malejącej, zaczynając od tych, które najbardziej obciążają system hosta.

► **virt-what;**

► **virt-df**

Pokazuje informacje na temat zużycia dysku maszyny wirtualnej:

virt-df -a virt1.img

Filesystem 1K-blocks Used Available Use%

virt1.img:/dev/sda1 99099 1551 92432 2%

Jeżeli chodzi o QEMU, poza wymiennionym już narzędziem qemu-img, dostępne są także następujące polecenia:

**Tabela8. Najważniejsze polecenia QEMU**

|  |  |  |
| --- | --- | --- |
| **polecenie** | | **rola** |
| **QEMU-IMG:** | |  |
|  | **-check** | sprawdza obraz i jeśli napotka w nim błędy to je wyświetla, jeśli nie nie ma błędów, to nie wyświetla nic |
|  | **-convert** | zamienia format obrazów |
|  | **-create** | tworzy obraz |
|  | **-snapshot** | tworzy tzw.*offline disk snapshot* |
|  | **-info** | wyświetla informacje na temat obrazu danej maszyny |
| **QEMU-INFO:** | | |
|  | **-block** | wyświetla informację o urządzeniach blokowych, takich jak cdrom, dysk twardy |
|  | **-blockstats** | wyświetla statystyki dotyczące urządzeń blokowych |
|  | **-cpus** | wyświetla informację o CPU |
|  | **-cpustats** | wyświetla statystyki zużycia CPU |
|  | **-mem** | wyświetla informację o pamięci maszyn wirtualnych |
|  | **-mmtree** | wyświetla drzewo pamięci |
|  | **-tlb** | wyświetla informację o pamięci wirtualnej i fizycznej na danycm hoście |
|  | **-usb** | wyświetla listę urządzeń usb na wirtualnym porcie USB |
|  | **-usbhost** | wyświetla listę urządzeń USB na konkretnej maszynie wirtualnej |
|  | **-history** | wyświetla historię poleceń konsoli |
|  | **-irq** | wyświetla statystykę dotyczącą przerwań |
|  | **-network** | wyświetla informacje o VLANs |
|  | **-pci** | wyświetla urządzenia PCI, które są emulowane |
|  | **-snapshots** | wyświetla dostępne snapshoty |
|  | **-qtree** | wyświetla drzewo urządzeń |
|  | **-roms** | wyświetla informację o dostępnych urządzeniach rom |
|  | **-migrate** | wyświetla informację o stanie migracji |
|  |  |  |
|  |  |  |
|  | **-snapshot** | snapshoty zapisują informację o całej wirtualnej maszynie-stanie CPU,RAM, stanie jej urządzeń, zawartości dysków. Aby móc używać snapshotów należy dysponować przynajmniej jednym nieusuwalnym i nadającym się do zapisu, urządzeniem blokowym, który używa formatu qcow2 jako obrazu dysku. |

Źródło: opracowanie własne

**5.3.**  **Konfiguracja aplikacji do zarządzania**

Poniższy rozdział szczegółowo opisuje w jaki sposób skonfigurowano aplikację webową służącą do zarządzania zasobami wirtualnymi na hostach. Podane sa przykłady implementacji kodu oraz wyjaśniona rola najważniejszych plików konfiguracyjnych.

**5.3.1. Uwierzytelnianie użytkowników**

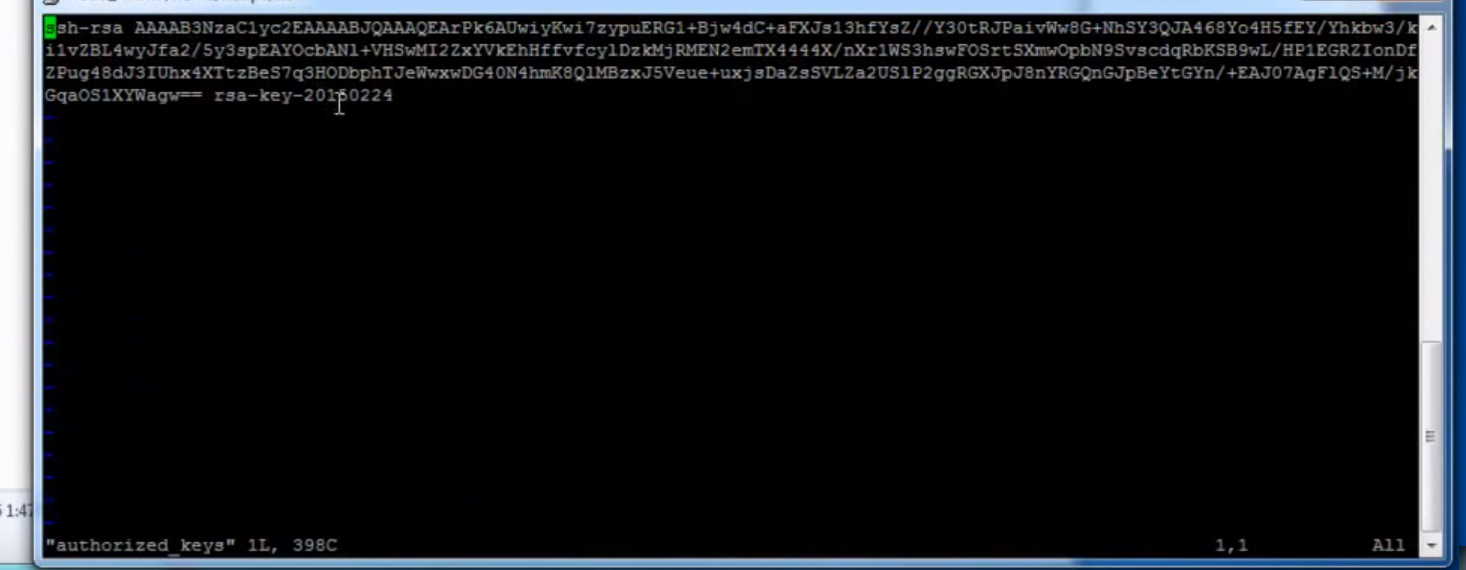
Stworzona aplikacja działa na komputerze fizycznym komp3 z systemem operacyjnym Windows7. Z założenia, dostęp do aplikacji ma tylko jeden użytkownik, który ma za zadanie zarządzać zbudowaną infrastrukturą. Z tego powodu, aplikacja działa tylko na jednym komputerze konkretnego użytkownika (na localhost:7070) i jako, że dostęp do niej będzie mieć tylko jedna osoba,nie zaimplementowano w niej modułu identyfikującego użytkowników ani bazy przetrzymującej ich dane. Gdyby jednak zaistniała potrzeba wystawienia aplikacji na zewnątrz,wtedy najrozsądniejszą metodą autoryzacji użytkowników będzie zaimplementowanie responsywnego formularza logowania Bootstrapa i połączenie go z bazą MySQL.

Jeżeli chodzi o autoryzację połączeń maszyny, na której działa aplikacja (komp3) z hostami (kom1, komp2), odbywa się ona przez SSH za pomocą klucza publicznego. Jest to bezhasłowa metoda logowania,która działa na zasadzie modelu *challenge-response*. Podczas procesu autoryzacji serwer duysponujący kluczem publicznym tworzy tzw. challenge, który tylko uzytkownik posiadający klucz prywatny tej samej pary jest w stanie odszyfrować i tym samym udzielić poprawnej odpowiedzi (ang.*response*).

Aby skonfigurować autoryzację za pomocą kluczy SSH, po stronie komputera, z którego łączymy się zdalnie, czyli w naszym przypadku komp3, generujemy parę klucz- publiczny i prywatny. Ponieważ komputer, na którym jest aplikacja działa z systemem operacyjnym Windows, który nie ma zainstalowanego serwera SSH, należało zainstalować dwa dodatkowe programy w celu wygenerowania kluczy- PuTTY i PuTTYgen.

Następnie, za pomocą PuTTYgen przystąpiono do tworzenia pary kluczy. Klucze zapisujemy w nastepujących formatach : prywatny jako id\_rsa, publiczny- id\_rsa.pub. Powstały klucz prywatny zaszyfrowano hasłem i zapisano na dysku komputera, natomiast publiczny skopiowano na maszyny, z którymi się łączy (kom1 i komp2). Na każdej maszynie z systemem Xubuntu otwieramy następujący plik **HOME/.ssh/authorized\_keys** i wklejamy w nim zawartość klucza publicznego skopiowanego z ramki PuTTYgen. Klucz ten będzie miał następującą postać:

**Rys16. Zawartość pliku HOME/.ssh/authorized\_keys**



Źródło:opracowanie własne

Należy ustawić odpowiedni dostęp do plików następującymi poleceniami:

chmod 755 ~/.ssh/authorized\_keys  
chmod 755 ~/.ssh

W kolejnym kroku trzeba zmodyfikować plik konfiguracyjny SSH- /**etc/ssh/sshd\_config**, tak aby akceptował autoryzacje klucza publicznego. Linijki, które za to odpowiadają znajdują się poniżej:

# enable public-key authentication

RSAAuthentication no

PubkeyAuthentication yes

W pliku należy także określić ścieżkę dostępu do klucza publicznego.

Stworzona aplikacja odwołuje się do SHH w pliku /app/ssh.js. Plik ten zawiera ścieżkę dostępu do klucza prywatnego zapisanego na tym komputerze oraz nazwę użytkownika.

**5.3.2. Projektowanie front-endu aplikacji**

Front-end, czyli wygląd aplikacji, powstał przy użyciu frameworku Bootsrap, zaopatrzonego w odpowiednie biblioteki dodatkowe. Szablon projektu strony internetowej znajduje się w pliku index.html (dla pierwszej zakładki) oraz newindex.html (dla następnej). Wersja podstawowa szablonu ma następującą postać:

<!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"></script>

<![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>

Napisany jest on w prostym języku HTML, natomiast fragment **<meta name="viewport"** odpowiada za skalowalność powstałej strony internetowej na urządzeniach mobilnych.

Przed ostatnim zamykającym tagiem tagiem <body>,do szablonu dołączono pliki CSS i JS z pobranej biblioteki oraz link do biblioteki jQuery.

Następnie do szablonu dołączono część odpowiadającą za menu i nawigację na stronie. Fragment ten wygląda w następujący sposów (w przypadku pliku 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>

Za dynamikę strony odpowiada z koleji Angular.js, który w swojej składni stał się adaptacją i rozszerzeniem języka HTML. AngularJS promuje deklaratywny styl pisania aplikacji (przeciwieństwo do imperatywnego HTML). Stanowi przykład modelu tworzenia aplikacji o nazwie MVC, czyli ang.*model-view-controller*. Wzorzec ten implikuje podział budowy aplikacji na trzy moduły:

* model- część odpowiedzialna za model danych- tutaj jest nim zbiór obiektów i prymitywów, do których można odwoływać się z obiektu Scope ($scope).
* widok- odpowiada za sposób prezentacji danych;
* kontroler- moduł odpowiedzialny za komunikowanie się użytkownika z systemem; przyjmuje żądania, przekazuje przetworzone dane do modelu i administruje widokami- tutaj jest nim klasa JavaScript podczepiana do zasięgu danego widoku.

W takim układzie widok aktualizuje model, który od razu jest zmieniony w kontrolerze,i na odwrót, gdy kontroler odświeży model, ten będzie od razu zaktualizowany w widoku. Jest to tzw. *two-way data-binding*, czyli wiązanie obustronne.

Dynamiką striny sterują dyrektywy, których można użyć na kilka sposobów:

* jako element,
* jako atrybut,
* jako klasa CSS,
* jako komentarz.

AngularJS posiada wbudowane dyrektywy, z których autor aplikacji może skorzystać. Do najważniejszych z nich należą :

* ng-app- deklaruje element jako główny element aplikacji;
* ng-init - wykonywania kodu na początku,
* ng-bind - zmienia automatycznie tekst elementu HTML na wartość danego wyrażenia,
* ng-model - służy do komunikowania się między użytkownikiem a modelem,
* ng-controller - definiuje klasę kontrolera JavaScript, która ma za zadanie przetwarzać dane zwracane do HTML,
* ng-show - gdy wartość będzie truly to kontener się wyświetli (dodaje klasę ng-hide - widoczność poprzez CSS),
* ng-if - usuwa albo dodaje dany fragment, obsługuje wyrażenia typu if, które pozwalają na pokazywanie danego elementu, jeżeli warunki są spełnione,
* ng-repeat - służy do iteracji po tablicy,
* ng-list - podczas prezentacja tablicy jako ciąg znaków, traktuje ten ciąg jako tablicę

Przykładów dyrektyw Angulara użytych w kodzie dostarcza poniższa tabelka:

**Tabela9. Przykłady dyrektyw AngularJS w kodzie aplikacji.**

|  |  |  |
| --- | --- | --- |
| **fragment kodu** | **nazwa dyrektywy** | **rola** |
| **<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 = $index">  <td>{{ computer.ip }}</td>  <td>  <select name="virtuals" class="form-control" ng-model="virtIndex" ng-change="dashboard.updateStatus(computerIndex,virtIndex)">  <option value="">Please select --</option>  <option **ng-repeat**="virt in computer.virts.concat().sort()" **ng-value**="{{$index}}">{{virt.name}}</option>  </select>  </td>  <td>  {{ computer.virts[computer.current\_machine\_index].ip }}  </td>  <td>  {{ computer.virts[computer.current\_machine\_index].mac }}  </td>  <td> | •**ng-repeat**  •**ng-init**  •**ng-value** | -odpowiada za iterowanie po kolekcji elementów oraz ich wyświetlanie i / lub filtrowanie  -odpowiada za rozpatrywanie danych w określonym zakresie; w tym przypadku występuje jako alias specjalnych wartości dyrektywy ng-repeat; przetrzymuje index pierwszego elementu indexu w zmiennej computerIndex  -stosuje sie ją, gdy w sposób dynamiczny generuje się listę, w tym przypadku maszyn wirtualnych  -$index- to usługa, która jest obiektem,to co zdefiniujemy w tym obiekcie, będzie od razu dostępne dla widoku (dyrektywy). |
| **<div class="row table-top">**  <select **ng-change**="dashboard.updateCurrentComputer(computerIndex)" class="pull-left form-control virt-selector pull-left" name="" **ng-model**="computerIndex">  <option value="">Please select --</option>  <option value="{{$index}}" ng-repeat="computer in dashboard.computers">{{computer.ip}}</option>  </select>  <a ng-click="dashboard.shutdown(dashboard.current\_computer.ip)" class="btn btn-primary">Turn off</a>  <a ng-click="dashboard.reboot(dashboard.current\_computer.ip)" class="btn btn-primary">Reboot</a>  <a ng-click="dashboard.refreshData()" class="pull-right btn btn-primary">Refresh data</a>  </div> | **•ng-change**  **•ng-model** | -dyrektywa wywoływana przy zmianie select,w tym przypadku zmianie ulegają wczytywane maszyny wirtualne  -powoduje przypisanie zawartości pola do zmiennej computerIndex,przypisuje wartość inputu do tego modelu |

Źródło: opracowanie własne

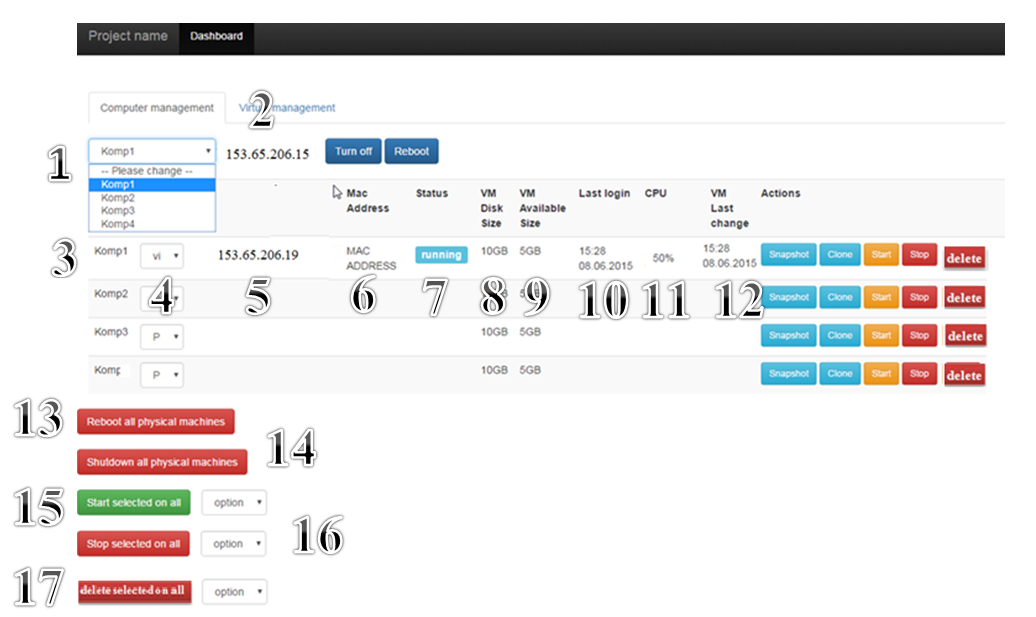
**5.3.3. Projektowanie back-endu aplikacji**

Warstwę backendowa stworzonej aplikacji stanowi NodeJS, skrypt napisany w Perlu uruchamiany po stronie zdalnego komputera oraz skrypty, które uruchamiane są na zdalnym komputerze.

Za komunikację ze zdalnym komputerem odpowiada AngularJS, który łączy sie do niego przez port SSH i tam uruchamia znajdujący się skrypt napisany w Perlu- comm.pl. Skrypt ten z koleji zbiera dane z danego komputera i zwraca je na standardowe wyjście (*standard output*).

Skrypt comm.pl w niektórych przypadkach uruchamia zdalnie pomocnicze skrypty napisane w shellu, które dostarczają mu danych o wirtualnych maszynach; innym razem sam wydaje polecenia, korzystając z komend takich jak virsh, qemu-img info, etc.

Przykładowe fragmenty kodu wraz z odniesieniami do widoku strony HTML aplikacji przedstawia poniższa tabelka:



|  |  |  |  |
| --- | --- | --- | --- |
| **nr** | funkcja przycisku | pobieranie informacji | pochodzenie informacji |
| **1** | Wybór maszyny fizycznej z listy rozwijanej. | Dane dotyczące zdalnych komputerów fizycznych znajdują sie w pliku \app\pc-list.js skąd są pobierane . | Dane wpisane ręcznie do pliku \app\pc-list.js. |
| **2** | IP maszyny fizycznej | Informacja ta wypełnia się automatycznie po wybraniu nazwy komputera. Dane dotyczące zdalnych komputerów fizycznych znajdują sie w pliku \app\pc-list.js skąd są pobierane. | Dane wpisane ręcznie do pliku \app\pc-list.js. |
| **3** | Wyświetla nazwę komputera fizycznego, z którego pobierane są dane, maszyna nie pojawi się na liście jeśli jest offline. | Nazwy komputerów znajdują się w pliku \**app\pc-list.js.** | Do nazw komputerów z pliku pc-list.js odwołujemy się w pliku \app\management.js, jeśli połączenie z danym komputerem nie jest możliwe, to nie wyświetli się na liście. |
| **4** | Nazwa maszyny wirtualnej dostępnej na danym komputerze fizycznym, wyłączone maszyny fizyczne także są wyświetlane. | Informacje tę pobiera skrypt w Perlu o nazwie **comm.pl** , który uruchamiany jest na danej maszynie fizycznej. Skrypt ten do wydobycia danych korzysta z polecenia virsh –all. | Fragment skryptu comm.pl, który odpowiada za uzyskanie danych:    getDataLines:    Skrypt comm.pl korzysta z polecenia :  virsh list --all  które wyświetla nazwy wszystkich maszyn wirtualnych na danym komputerze.    Przykładowy output polecenia:    Przy wyświetlaniu wyniku skrypt w Perlu pokazuje tylko drugą kolumnę, zaczynając od drugiego wersu; odpowiada za to fragment skryptu getDataLines.  Przykładowy output:    Ten sam efekt można osiągnąć korzystając z awk i sed:  virsh list --all| awk '{print $2}'| sed 1d |
| **5** | IP maszyny wirtualnej | Informację pobiera skrypt comm.pl, który na maszynie hoście uruchamia znajdujący się tam skrypt getIPaddress.sh, a następnie sczytuje jego wyniki. | Fragment skryptu comm.pl, który odpowiada za uzyskanie danych:    Skrypt comm.pl uruchamia skrypt getIPaddress.sh,który uzyskuje adresy IP wymienionych wcześniej maszyn wirtualnych zapisanych w zmiennej virts.  Skrypt getIPaddress.sh najpierw uzyskuje MAC adress maszyny wirtualnej, korzystając z libvirt,fragment kodu:  MAC\_ADDRESS=$($virsh dumpxml "$VM\_NAME" | $grep "mac address" | $sed "s/.\*'\(.\*\)'.\*/\1/g")  a następnie odwołuje się do arp, aby wyszukać IP odpowiadajace danemu adresowi MAC, fragment kodu:  $arp -an | $grep "$MAC\_ADDRESS" |
| **6** | MAC adres danej maszyny wirtualnej |  | Fragment kodu skryptu comm.pl:    Skrypt w Perlu korzysta z polecenia:  virsh dumpxml <VM-NAME-HERE> | grep -i '<mac'  które wykonywane jest dla każdej z maszyn wirtualnych.  Przykładowy output polecenia **virsh dumpxml virt1| grep –i** :  **<mac** address='52:54:00:4c:40:1c'**/>** |
| **7** | stan  maszyny wirtualnej:  -running,  -blocked,  -paused,  -shutdown,  -dying,  -crashed. | Informację pobiera skrypt comm.pl | Skrypt comm.pl pobiera tę informację korzystając z polecenia virsh –all uruchamianego na każdej maszynie hosta.  W przypadku, gdy maszyna wirtualna jest skonfigurowana tak, aby przy jej uszkodzeniu system uległ reinicjalizacji, w momencie wystąpienia usterki wpada ona w tzw.*endless loop*. Ten zapis dotyczący konfiguracji można znależć w pliku XML każdej maszyny, mówi o tym fragment: <on\_crash>restart</on\_crash>  W przypadku, kiedy maszyna wirtualna posiada tę opcję w pliku XML, polecenie **virsh –all** w polu ***State*** posiada sześć możliwych odpowiedzi.  Alternatywnie, w celu uzyskania informacji, czy obraz danej maszyny działa; można zastosować polecenie qemu-img check. |
| **8** | rozmiar używanej obecnie pamięci maszyny wirtualnej | Informację pobiera skrypt comm.pl | Skrypt comm.pl pobiera tę informację uruchamiając odpowiedni skrypt na każdej maszynie hosta.  Polecenie, do którego odnosi się skrypt w Perlu:  virsh dumpxml <VM-NAME-HERE> |grep -i memo  Przykładowy output polecenia dla virt1:  <**memory** unit='KiB'>16777216</**memory**>  <**currentMemory** unit='KiB'>1048576</**currentMemory**>  Alternatywnie, można także odnieść się do polecenia **virsh dominfo.** Przykładowy output tego polecenia dla maszyny virt1 wygląda następująco:  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 |
| **9** | maksymalny rozmiar pamięci maszyny wirtualnej | Informację pobiera skrypt comm.pl | Korzysta z tego samego polecenia jak poprzednio. |
| **10** | data ostatniego logowania do wirtualnej maszyny | Informację pobiera skrypt comm.pl | Informacja pochodzi z logu wirtualnej maszyny, który znajduje się w folderze /var/log/libvirt/qemu/virtual\_machine.log |
| **11** | data ostatniej zmiany w obrazie maszyny wirtualnej | Informację pobiera skrypt comm.pl | Wygenerowanie tej informacji powiązane jest z mechanizmem obrazów zagnieżdżonych (ang.*derived images*) jakie oferuje wirtualizacja. Zmianami w obrazie maszyny wirtualnej są w tym przypadku utworzone snapshoty, które stanowią tzw.*delta images*, czyli przechowują jedynie zmiany w stosunku do obrazu bazowego.  Datą ostatniej wprowadzonej zmiany na obrazie bazowym będzie więc data utworzenia ostatniego snapshotu tego obrazu.  Informacje o dostępnych snapshotach dla danej maszyny wirtualnej można uzyskać poleceniem virsh-snapshot –list <domain>  Przykładowy output: |

Aplikacja wykorzystuje do komunikacji z maszynami fizycznymi środowisko Node.js poprzez moduł node-ssh-exec.

Przykładowe żądanie do maszyny fizycznej (wyłączenie):

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);

});

}

Aplikacja posiada wystawione API do użycia przez klienta (strona internetowa). Każdy request do aplikacji jest przechwytywany i na podstawie jego typu, adresu oraz parametrów wykonywane jest odpowiednie żądanie.

**5.3.4. Uruchamianie aplikacji i odczytywanie wyników**

Aby aplikacja działała, najpierw należy zaktualizowac liste komputerów fizycznych, znajdującą się w pliku /app/pc-list.js a także odpowiednio zedytować plik /app/ssh.js, który zawiera konfigurację połączenia SSH. Kolejnym krokiem będzie umieszczenie klucza publicznego na każdym komputerze zdalnym, do którego chcemy się połączyć i sprawdzenie czy port SSH na maszynie jest prawidłowo skonfigurowany.

Po weryfikacji poprawności połączenia zdalnego, możliwe będzie uruchomienie aplikacji. W tym celu należy wejść do folderu głównego i wykonać z niej polecenie **npm install**, które zainstaluje aplikację wraz z jej rozszerzeniami.

Następnie, uruchamiamy serwer poleceniem: **node server.js.** W momencie, kiedy otrzymamy pozytywną odpowiedz o treści:‟**Server listening on port 7070**”, możliwe będzie korzystanie z aplikacji pod adresem **http://localhost:7070**.

1. A, Chrobot,G. Łukawski G., *Systemy Operacyjne — Wirtualizacja*, Katedra Informatyki, Politechnika Świętokrzyska w Kielcach, s.3 [↑](#footnote-ref-1)
2. Ibidem,s.4 [↑](#footnote-ref-2)
3. *Wirtualizacja*, https://pl.wikipedia.org/wiki/Wirtualizacja (data odczytu 07 grudnia 2015) [↑](#footnote-ref-3)
4. *Wirtualizacja*, https://pl.wikipedia.org/wiki/Wirtualizacja (data odczytu 07 grudnia 2015) [↑](#footnote-ref-4)
5. M.Stachowicz, *Wirtualizacja Aplikacji. Porównanie Technologii*, Politechnika Śląska: Wydział Inżynierii Materiałowej i Metalurgii, Katowice 2011,s.9 [↑](#footnote-ref-5)
6. Ibidem,s.10 [↑](#footnote-ref-6)
7. J.P.Buzen,U.O. Gagliardi, *The Evolution of Virtual Machine Architecture*, AHPS Press, Montvale, N.J., ss.291-300 [↑](#footnote-ref-7)
8. R.P.Goldberg,G.J. Popek; *Formal Requirements for Virtualizable Third Generation Architectures;* Honeywell Information Systems, Harvard University; Los Angeles; s.417 [↑](#footnote-ref-8)
9. Ibidem,s.418 [↑](#footnote-ref-9)
10. R.P. Goldberg, G.J.Popek; *Formal Requirements for Virtualizable Third Generation Architectures;* Honeywell Information Systems, Harvard University; Los Angeles; s.420 [↑](#footnote-ref-10)
11. M.Sawicz, *Wirtualizacja ale po co?*, <http://jesien.linux.org.pl/2008/materialy/prelekcje/wirtualizacja.pdf> ; s.7(data odczytu 07 grudnia 2015) [↑](#footnote-ref-11)
12. P.Przybylak; *Wirtualna Infrastruktura – Nowe Podejscie Do Systemow*; <http://zeszyty-naukowe.wwsi.edu.pl/zeszyty/zeszyt4/Wirtualna_Infrastruktura_-_Nowe_Podejscie_Do_Systemow.pdf>; s.13 (data odczytu 07 grudnia 2015) [↑](#footnote-ref-12)
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) [↑](#footnote-ref-13)
14. M. Tulloch; *Understanding Microsoft Virtualization Solutions, From the Desktop to the Datacenter*; Microsoft Press; Redmond Washington 2010; ss. 23–27 [↑](#footnote-ref-14)
15. A. S. Tanenbaum, *Systemy Operacyjne*, Gliwice, Helion 2010, ss. 671–677 [↑](#footnote-ref-15)
16. Editorial Board; *International Journal of Security and Networks,* www.inderscience.com/ijsn/; Wright,Ohio 2004,s.14 [↑](#footnote-ref-16)
17. M Bishopp, *Introduction to Computer Security*, Addison-Wesley, 2005, s.645 [↑](#footnote-ref-17)
18. *Wirtualizacja*, <http://ii.uni.wroc.pl/~msq/so09/slides/2_4_wirtualizacja.pdf> (data odczytu 20 grudnia 2015) [↑](#footnote-ref-18)
19. *Virtualization*, <http://cecs.wright.edu/~pmateti/Courses/7380/Lectures/Virtualization/Slides/vm-umass-2015.pdf> (data odczytu 17 grudnia 2015) [↑](#footnote-ref-19)
20. *Virtualization*, <http://cecs.wright.edu/~pmateti/Courses/7380/Lectures/Virtualization/Slides/vm-umass-2015.pdf> (data odczytu 17 grudnia 2015) [↑](#footnote-ref-20)
21. M.Bishopp, *Introduction to Computer Security*, Addison-Wesley, 2005, s.660 [↑](#footnote-ref-21)
22. G.Terlikowski, *Wstęp do Wirtualizacji i RIA*, Instytut Informatyki UPH w Siedlcach, Siedlce 2015, s.11 [↑](#footnote-ref-22)
23. <http://bip-slaskie.pl/zamowienia/18/1192108801/1299765671.pdf>, s.6 (data odczytu 20 grudnia 2015) [↑](#footnote-ref-23)
24. <http://winntbg.bg.agh.edu.pl/rozprawy2/10081/full10081.pdf> (data odczytu 21 grudnia 2015) [↑](#footnote-ref-24)
25. <http://bip-slaskie.pl/zamowienia/18/1192108801/1299765671.pdf>; s.6 (data odczytu 21 grudnia 2015) [↑](#footnote-ref-25)
26. A.Bula; *Wirtualizacja Systemow Operacyjnych- Analiza Porównawcza*, <http://www.math.uni.opole.pl/dokumenty/lnt/Wirtualizacja_systemow_operacyjnych.pdf>, s.9 (data odczytu 23 grudnia 2015) [↑](#footnote-ref-26)
27. [↑](#footnote-ref-27)
28. L.Kaczmarek, M.Wróbel; *Możliwości Stosowania Wirtualizacji w Systemach Komputerowych; Zastosowanie Komputerów w Nauce i Technice:* Zeszyty Naukowe Wydziału Elektrotechniki i Automatyki Politechniki Gdańskiej Nr 30; Gdańsk 2014; s.2 [↑](#footnote-ref-28)
29. Ibidem, s.3 [↑](#footnote-ref-29)
30. E. Łukasik, M. Skublewska-Paszkowska,J.Smołka J; *Android i iOS – tworzenie aplikacji mobilnych*; Politechnika Lubelska; Lublin 2014; s.70 [↑](#footnote-ref-30)