Akademia Górniczo-Hutnicza
im. Stanisława Staszica
w Krakowie

Wydział Fizyki i Informatyki Stosowanej

Praca inżynierska

Bartłomiej Mucha

kierunek studiów: **informatyka stosowana**

Migracja serwisów internetowych i integracja usług w środowisku kontenerów Docker

Opiekun: **dr inż. Piotr Gronek**

Kraków, styczeń 2020

Oświadczenie studenta

Uprzedzony(-a) o odpowiedzialności karnej na podstawie art. 115 ust. 1 i 2 ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (t.j. Dz. U. z 2018 r. poz. 1191 z późn. zm.): „Kto przywłaszcza sobie autorstwo albo wprowadza w błąd co do autorstwa całości lub części cudzego utworu albo artystycznego wykonania, podlega grzywnie, karze ograniczenia wolności albo pozbawienia wolności do lat 3. Tej samej karze podlega, kto rozpowszechnia bez podania nazwiska lub pseudonimu twórcy cudzy utwór w wersji oryginalnej albo w postaci opracowania, artystyczne wykonanie albo publicznie zniekształca taki utwór, artystyczne wykonanie, fonogram, wideogram lub nadanie.", a także uprzedzony(-a) o odpowiedzialności dyscyplinarnej na podstawie art. 307 ust. 1 ustawy z dnia 20 lipca 2018 r. Prawo o szkolnictwie wyższym i nauce (Dz. U. z 2018 r. poz. 1668 z późn. zm.) „Student podlega odpowiedzialności dyscyplinarnej za naruszenie przepisów obowiązujących w uczelni oraz za czyn uchybiający godności studenta.", oświadczam, że niniejszą pracę dyplomową wykonałem(-am) osobiście i samodzielnie i nie korzystałem(-am) ze źródeł innych niż wymienione w pracy.

Jednocześnie Uczelnia informuje, że zgodnie z art. 15a ww. ustawy o prawie autorskim i prawach pokrewnych Uczelni przysługuje pierwszeństwo w opublikowaniu pracy dyplomowej studenta. Jeżeli Uczelnia nie opublikowała pracy dyplomowej w terminie 6 miesięcy od dnia jej obrony, autor może ją opublikować, chyba że praca jest częścią utworu zbiorowego. Ponadto Uczelnia jako podmiot, o którym mowa w art. 7 ust. 1 pkt 1 ustawy z dnia 20 lipca 2018 r. — Prawo o szkolnictwie wyższym i nauce (Dz. U. z 2018 r. poz. 1668 z późn. zm.), może korzystać bez wynagrodzenia i bez konieczności uzyskania zgody autora z utworu stworzonego przez studenta w wyniku wykonywania obowiązków związanych z odbywaniem studiów, udostępniać utwór ministrowi właściwemu do spraw szkolnictwa wyższego i nauki oraz korzystać z utworów znajdujących się w prowadzonych przez niego bazach danych, w celu sprawdzania z wykorzystaniem systemu antyplagiatowego. Minister właściwy do spraw szkolnictwa wyższego i nauki może korzystać z prac dyplomowych znajdujących się w prowadzonych przez niego bazach danych w zakresie niezbędnym do zapewnienia prawidłowego utrzymania i rozwoju tych baz oraz współpracujących z nimi systemów informatycznych.

.....

(czytelny podpis)

Spis treści

Rozdział 1

Wstęp

Możliwości, jakie oferują maszyny wirtualne są nieocenione we współczesnym świecie. Systemy serwerowe, w skład których mogą wchodzić nawet tysiące powiązanych ze sobą lub indywidualnych serwisów, nie mogą występować albo są zasadniczo trudne do wdrożenia na jednej platformie o określonym systemie opracyjnym, konfiguracji i innych elementach wchodzących w skład szeroko rozumianego środowiska. Maszyny wirtulane są rozwiązaniem tego zagadnienia. Każda maszyna wirtualna może zostać stworzona niezależnie od architektury sprzętowej i oferuje dowolone środowisko, które jednocześnie będzie wyizolowane od środowisk gospodarza (*host*) i innych maszyn wirtualnych (*guest*). Korzyścią płynącą z izolacji środowiska jest chociażby zwiększone bezpieczeństwo systemu, ułatwienie zarządzania serwisami oraz wdrażania nowych wersji aplikacji. Nierzadko używane są stare serwisy, a jednak spełniające swoje zadanie, które wymagają rozwiązań nieaktualnych już wersji elementów środowiskowych, nie wspieranych przez nowsze wersje. Co za tym idzie wdrożenie np. dwóch aplikacji działających w ramach tej samej technologii, ale na różnych wersjach będzie tworzyć konflikt. Problemy tego typu zanikają w kontekście wirtualzacji, albowiem nic nie stoi na przeszkodzie osadzenia każdej pojedynczej aplikcaji na osobnej maszynie. Wirtualizacja posiada jednak szereg wad. Taką wadą jest na przykład to, że każda maszyna wirtualna musi rezerwować określoną i niezmienną w czasie działania ilość zasobów, takich jak pamięć operacyjna (np. *RAM*) i masowa (nieulotna) oraz liczbę wątków procesora i wiele innych. Stawia to przed adminstratorem problem, związany z dobraniem parametramów jakimi taka maszyna ma się cechować. Przydzialnie zbyt małej ilości zasobów sprawi, że serwis będzie niedomagał. Z kolei przydzielnie zbyt dużej ilości zasobów doprowadzi do marnowania się części zasobów, które mogłyby być lepiej wykorzystane. To, jak maszyna wirtualna zużywa zasoby, takie jak pamięć operacyjna, nie jest kontrolowane. Zarezerwowany zasób jest statyczny i nie jest zwalniany w przypadku braku użycia. Rozwiązaniem tego problemu jest zmiana

podejścia do koncepcji wirtualzacji, a mianowicie konteneryzacja serwisów. Platformy takie jakie *Docker*[6] czy *LXC*[11] pozwalają tworzyć niewymagające udziału hipernadzorcy wyizolowane środowiska o dynamicznie przydzielanych zasobach, tak zwane kontenery. Kontenery zarządzane przez platformę Docker współdzielą jądro systemowe z gospodarzem i innymi kontenerami, a w ramach osadzenia aplikacji wystarczy dostarczyć zbudowaną aplikację oraz wymagane do poprawnego jej działania składniki systemowe. W ten sposób można zaoszczędzić na użyciu pamięci twardej względem tej samej aplkacji wdrożnej na maszynie wirtualnej.

1.1 Cel pracy

Celem pracy jest przeniesienie aplikacji z maszyn wirtualnych na kontenery oraz opracowanie tej metodologii. W konsekwencji zostaną stworzone skalowalne środowiska usług internetowych, działające w oparciu o technologię kontenerów Docker na platformie Linux w systemie wirtualizacji *VMware ESX*[20] oraz porównanie i ocena korzyści wynikających z tych działań. Modelem przykładowym jest system aplikacji działających w technologii *Php*[13] i *Java*[10] z framework'iem *Spring Boot*[18] oraz baza danych *PostgreSQL*[14] na trzech osobnych kontenerach Docker. Kontenery będą osadzone na zainstalowanej w systemie wirtulazacji ESX maszynie wirtualnej z system operacyjnym *Debian 10*[4]. Finalnie wdrożone zostaną realne serwisy w wydziałowej sieci komputerowej.

Plan pracy jest następujący. W części teortycznej zostaną omówione mechanizmy stojące za działaniem maszyn wirtualnych i kontenerów na platformie Docker oraz ich porównanie. Natępnie omówiony zostanie schemat wdrożenia aplikacji. Część praktyczna będzie zawierać szczegółowy opis przygotowania środowiska oraz wdrożenia aplikacji zarówno przykładowych.

1.2 Założenia projektu

Celem projektowej części pracy jest opracowanie skalowalnego środowiska usług internetowych, działającego w oparciu o technologię kontenerów Docker na platformie *Linux* w systemie wirtualizacji *VMware ESX*[20]. Usługi realizowane przez system mają obejmować obsługę: - internetowych serwisów aplikacyjnych działających w technologiach odpowiednio: *Php*[13], *Apache Tomcat*[2], *Python*[15] *Django*[5], witryn *CMS WordPress*[25], serwerów relacyjnych baz danych *MySQL*[12], *PostgreSQL*[14]. Elementem pracy będzie także migracja istniejących serwisów internetowych do ich odpowiadających im instancji wyżej wymienionych kontenerów *Docker*[6] oraz ocena

uzyskanych korzyści pod kątem konsumowanych zasobów sprzętowych i wydajności działania.

Rozdział 2

Podstawa Teoretyczna

2.1 Wirtualizacja

Szeroko rozumiane urządzenie zwane komputerem składa się z dwóch kategorii komponentów: urządzeń fizycznych i oprogramowania. W skład urządzeń fizycznych wchodzą między innymi płyta główna, centralna jednostka przetwarzająca i urządzenia wejścia/wyjścia. Oprogramowanie składa się z warstw, najniżej, najbliżej sprzętu znajduje się oprogramownie do zarządzania i komunikacji z urządzaniami, w wyższych warstwach znajdziemy aplikacje użytkowe. Zadaniem systemu operacyjnego jest zarządzanie zarówno zasobami sprzętowymi jak i oprogramowaniem. Stanowi on interfejs pomiędzy maszyną a użytkownikem, umożliwiając tym samym stosowanie komputerów do celów osobistych znanych nam z życia codziennego i profesjonalnych na linii człowiek - maszyna, ale i maszyna - maszyna. Ideą systemu operacyjnego jest istnienie środowiska, które dynamicznie będzie tłumaczyć zlecenia użytkownika na język maszynowy, następnie zlecać sprzętowi ich wykonanie i w drugą stronę informować użytkownia o rezultatach pracy maszyny oraz jej zapotrzebowaniu na dane, zachowując jednocześnie optymalne działanie pracy całego systemu i zawartych w nim urządzeń.

Rysunek 2.1: Schmat warstw systemu operacyjnego

Na rysunku 2.1 przedstawiony jest schemat budowy systemu operacyjnego. W skład samego systemu operacyjnego wchodzą sterowniki, jądro i powłoki systemowe. System operacyjny jest również podzielony na przestrzenie jądra i użytkownika. Wirtualizacja zatem jest symulacją warstwy fizycznej komputera, na której można zainstalować i używać dowolny system operacyjny. Zasymulować można różnego rodzaju architektury i parametry sprzętowe, zalecane jest jednak by wirtualna platforma sprzętowa nie przekraczała możliwości rzeczywistego sprzętu. Wirtualizacji można dokonać na kilku poziomach.

2.1.1 Poziom architektury zestawu instrukcji (*ISA*)

Tak zwana pełna wirtualizacja. Wszytkie instrukcje procesorowe systemu gościa są interpretowane przez emulator, a następnie mapowane na rzeczywisty sprzęt fizyczny gospodarza. Dalej instrukcje są wykonywane i zwracane do emulatora, który przekazuje je do gościa. Z racji długiej drogi, jaką musi pokonać pojendyczna instrukcja to rozwiązanie należy do najwolniejszych. Tę metodę stosuje się, gdy architektury sprzętowe gościa i gospodarza są zbyt różne, aby dało się je zmapować lub jest to nieopłacalnie trudne. Rozwiąnie to stosuje się na przykład do emulacji środowisk smartfonów, konsol do gier i mikrokontrolerów.

2.1.2 Poziom warstwy abstrakcji sprzętowej (*HAL*)

Inaczej nazywany poziomem sprzętowego wspomagania. Ta metoda wirtulizacji polega na mapowaniu rzeczywistych zasobów sprzętowych na wirtualne zasoby. Zatem wszystkie instrukcje nieuprzywilejowane są wykonywane bezpośrednio na sprzęcie gospodarza. Instrukcje uprzywilejowane, czyli takie które może wykonać tylko gospodarz są przekazywane do hipernadzorcy i ten je wykonuje. Rozwiąznie to jest zasadniczo szybsze od wirtualizacji zestawu instrukcji. Z tej metody korzystają narzędzia do wirtualizacji takie jak *VMware Workstation*[22], *Oracle VirtualBox*[19] i wiele innych.

2.1.3 Poziom systemu operacyjnego

Wirtualizacja na poziomie systemu opercyjnego pozwala na działanie kilku odrębnych odizolowanych od siebie systemów operacyjnych, a dokładnie przestrzeni użytkownia korzystających z jednego jądra systemowego. Zatem nie dochodzi do duplikacji całych systemów operacyjnych, a co za tym idzie nie jest wymagana praca hipernadzorcy. Rozwiązanie takie jest nazywane konteneryzacją i jest stosowane na takich platformach jak Docker, *CoreOS*[3], *LXC*[11] i *Oracle Solaris Containers*[17]. Wadą tej metody jest przymus używania tylko takich systemów operacyjnych, które mogą współdzielić jądro z systemem gospodarza. Nic jednak nie stoi na przeszkodzie, aby umieścić kontenery na maszynie wirtualnej postawionej na poziomie wspierania sprzętowego, jednak spowolni to szybkość tych kontenerów do szybkości samej maszyny wirtualnej.

2.1.4 Poziom bibliotek lub języków programownia

Poziom ten jest stosowany w ramach technologii *JIT* (*just in time*). Niektóre środowiska programistyczne pozwalają na kompilację kodu źródłowego do postaci kodu pośredniego, który nie ma bezpośredniego przełożenia na kod maszynowy. Program jest wykonywany na maszynie wirtualnej, która kompiluje kod pośredni do kodu maszynowego, a może to zrobić na poczatku wywołania programu lub pierwszego wywołania konkretnej części kodu. Z tej technologii korzysta *Java*[10], *.Net*[8], *Python*[15] i wiele innych języków programowania.

2.1.5 Poziom aplikacji

Wirtualizowane są tylko aplikacje, co oznacza, że nie wykorzystują one bezpośrednio dostarczanego przez system operacyjny środowiska wykonawczego, a używją

dostarczonego przez inną aplikację takie środowisko na jakie zostały zbudowane. Przykładem platformy umożliwijącej wirtualizacje aplikacji jest *Wine*[24], która umożliwia działanie aplikacji przystowanych do pracy w systemie *Microsoft Windows*[23] na systemie operacyjnym typu Linux.

2.2 Hipernadzorca

Komponentem systemu komputerowego umożliwiającym dokonanie wirtualizacji na poziomie wsparcia sprzętowego jest hipernadzorca (*hypervisor*). Rozróżnanie są dwa typy hipernadzorców:

2.2.1 Hipernadzorca typu pierwszego

Jest to tak zwany hipernadzorca natywny (*bare metal*) osadzony bezpośrednio na sprzęcie, nie wymaga on zainstalowanego systemu operacyjnego gospodarza. Pozwala on osadzać maszyny wirtualne nad sobą, co znacząco upraszcza schemat abstrakcji architektury całego systemu. Przykładami tego typu hipernadzorców są *Microsoft Hyper-V*[9] i *VMware ESXi*[20].

2.2.2 Hipernadzorca typu drugiego

Hipernadzorca hostowany jest osadzony w systemie operacyjnym gospodarza jako aplikacja. Takimi aplikacjami są przykładowo *VMware Workstation*[22], *VMware Player*[21], *Oracle VirtualBox*[19] i *QEMU*[16].

2.3 Konteneryzacja

Konteneryzacja jako wirtualizacja na poziomie systemu opercyjnego, z racji współdzielenia przez kontenery jądra systemu gospodarza, powinna cechować się większą wydajnością od wirtulizacji na poziomie *ISA* lub *HAL*. O ile nie zastąpi ona klasycznych maszyn wirtualnych pod względem możliwości uruchamiania różnorodnych systemów operacyjnych, o tyle najmocniejszą stroną konteneryzacji jest dużo niższe zapotrzebowanie na pamięć nieulotną, co za tym idzie przenaszalność oraz szybkość zarówno działania jak i startowania. Współdzielenie jądra systemowego oraz dostępu do fizycznej warstwy komputera niesie ze sobą pewne ryzyko jakim jest drastyczny spadek wydajności. Wszystkie kontenery konkurują ze sobą o zasoby sprzętowe, ale również o zasoby jądra, które nie jest odizolowne między nimi. Zespół Sysdig napotkał na problem degradacji szybkości działania środowiska wewnątrz kontenerów.

Z artykułu autorstwa Gianluca Borello[26] wynika, że jeden kontener spowalniał inny, poprzez wielkrotne wyszukiwania plików o zadanych ścieżkach dostępu. Proces rozwikłania scieżek dostępu jet procesem nietrywilanym i czasochłonnym, jądro *Linuxa* zapisuje w związku z tym dane jakie uzyskał w tablicy haszowanej, również próby nieudane, by w pszyszłości nie powtarzać całego procesu. Ma to na celu szybsze rozwikłania ścieżek. Jeżeli jednak dojdzie do nadmierngo rozrostu tablicy to efekt staje się odwrotny do zamierzonego i jądro działa wolniej, tym samym spowalniając uruchomione w każdej przestrzeni użytkownika apliakcje. Jądro systemu jest bardzo skomplikowanym oprogramowaniem i jak każde oprogramowanie może w pewnych warunkach nie działać optymalnie, jak w wymienionym przypadku i innych tworząc wąskie gardło pomiędzy warstwami użytkownika i warstwą fizyczną.

Rozdział 3

Założenia projektowe

3.1 Architektura wejściowa serwisów

3.1.1 Platforma systemu serwerowego

Na wydziałowym serwerze zainstalowny jest hipernadzorca typu pierwszego, dokładnie *VMware ESX 6.5*[20], osadzony bezpośrednio nad warstwą fizyczną klastra komputerowego. Na serwerze wdrożonych jest szereg aplikacji działających na odrębnych maszynach wirtulanych.

3.1.2 Wizualizaja platformy wejściowej

Na rysunku x.x znajduje się schemat wejściowej architektury systemu serwerowego, który w ramach tej pracy ma zostać przekształcony.

3.2 Architektura docelowa serwisów

Koncepcja architektury docelowej polega na zlikwidowaniu pewnej ilości maszyn wirtulanych i przeniesieniu serwisów w nich zawartych do kontenerów znajdujących się w jednej maszynie wirtualnej. Oczywistą korzyścią z tej akcji jest zaoszczędzenie pewnej, być może nawet dużej ilości pamięci twardej. Nie jest jednak oczywiste, do jakiej zmiany dojdzie w kwestii wydajności. O ile pierwotny układ składa się z jednego poziomu wirtualizacji, tak ten układ będzie się składał z dwóch: wirtualizacji wspomaganej sprzętowo oraz na poziomie systemu operacyjnego.

Rozdział 4

Opis przykładowych aplikacji

4.1 Opis

Przykładowe aplikacje to najprostsze aplikcje o identycznych wymaganiach jak aplikacje pochodzące z wydziałowego serwera. Aplikacja Java Spring Boot jest naprostszym serwisem http wyświetlający prostą zawartość w przeglądarce internetowej. Serwis Apache-Php również jest serwisem http, który udostępnia przez przeglądarkę interfejs *CRUD* do trzeciego serwisu, który dostarcza bazę danych PostgreSQL.

4.2 Środowisko projektowe i wymagania aplikacji

Środowisko projektowe:

- Narzędzie do wirtualizcji *VMware Workstation Player 15.5*[21]
- System operacyjny *Debian 10.2*[4]
- platforma *Docker 19.03.5*[6]
- narzędzie do zarządzania kontenerami *Docker-compose 1.25.0*[7]

Wymagania Aplikacji:

- Aplikacja *Java Spring Boot*
  - ⋆ *Java 8*[10] lub nowsza
- Aplikacja *Php*
  - ⋆ *Php 5.4*[13] lub nowsze z dodatkiem umożliwiającym połączenie z bazą danych PostgreSQL
  - ⋆ *Apache 2*[1] lub nowszy
  - ⋆ Dostęp do istniejącej bazy danych *PostgreSQL*[14]

4.3 Działanie i powiązania zbioru kontenerów

Docker

Platforma Docker do utworzenia kontenera wymaga dwóch komponentów: zbudowanej aplikacji i jej zasobów w postaci na przykład plików html, skryptów itd. oraz pliku konfiguracyjnego *dockerfile*, na podstawie którego zostanie utworzona przestrzeń użytkownika w kontenerze. Wykorzystanymi komendami *dockerfile* w projekcie są:

- *RUN* wykonuje polecnie powłoki systemu operacyjnego w konenerze podczas jego budowy
- *CMD* nadpisuje domyślną instrukcję *command*, która domyślnie uruchamia aplikacje wewnątrz kontenera. Dotyczy to przypadku, w którym kontener pracuje w trybie odłączonym od powłoki systemu gospodarza. Instrukcja *CMD* może zostać wykorzystana raz, każda kolejna nadpisze poprzednią, w konsekwencji wykonana zostanie tylko ostatnia.
- *COPY* kopiuje pod podane miejsce w systemie plików gościa wskazane pliki i katalogi z systemu plików gospodarza
- *ADD* jest to rozszerzona instrukcja *COPY*, pozwala również na dodanie plików z pod podanego adresu *url* oraz automatycznie rozpakować foldery skompresowane w trakcie kopiowania
- *ENTRYPOINT* wykonuje tę samą czynność co *CMD* z tą różnicą, że może być użyta wielokrotnie i każde użycie zostanie wykonane. Instrukcje *ENTRYPOINT* i *CMD* nie kolidują ze sobą
- *VOLUME* tworzy obszar w pamięci twardej gospodarza, w którym będzie przechowywana kopia danych wygenerownych przez kontener, np. logi, bazy danych itd.
- *FROM* instrukcja, której obecność jest zawsze wymagana do zbudowania obrazu. Musi znajdować się w pierwszej linii pliku docker file. Jej argumentem jest tag obrazu który ma zostać zbudowny. W trakcie budowy obraz jest pobierany z repozytorium.
- *ENV* tworzy zmienną środowiskową w goszczonym systemie opercyjnym
- *EXPOSE* pokazuje port widoczny dla maszyn zewnętrznych

Z racji tego, iż wybranym system opercyjnym gospodarza dla kontenerów jest *Debian 10*[4], *dockerfile* musi zawierać komendy, które zainstalują wymagane do działania aplikacji paczki i biblioteki oraz koniecznie musi korzystać z obrazów systemów z rodziny *Linux*.

Docker-compose

W przypadku kiedy w systemie pracuje wiele kontenerów pojawia się problem zarządzania nimi. Uruchamianie każdego kontenera z wiersza poleceń jest żmudnym i podatnym na pomyłki procesem. Zastosowanie skryptu np. w powłoce bash cechowałoby się nie małą nieprzerzystością. Programem służącym do tego celu jest *Docker-compose*. Pozwala on budowanie i uruchamianie wielu kontenerów docker jednocześnie. Tak jak *Docker*[6], *Docker-compose*[7] wymaga pliku konfiguracyjnego, tym razem w notacji *YAML* i o rozszerzeniu *docker-compose.yml*. W tym pliku są zawarte lokacje plików dockerfile i parametry służące do budowy i uruchomienia obrazu oraz konfiguracja sterowników np. sieciowych.

Aplikacja Java Spring Boot

Realizuje prosty zadanie, tworzy serwis internetowy i wyświetla przykłądowy napis oknie przeglądarki.

Aplikacja Php

Zadaniem tej aplikacji jest zainicjalizowanie bazy danych i udostępnienie za pośrednictwem strony *www* interfejsu *CRUD* do tej bazy.

Baza danych PostgreSQL

Baza danych musi istnieć i być dostępna dla aplikacji *Php*

4.4 Zrealizowane pliki dockerfile

Rozdział 5

Wdrożenie przykładowych oraz rzeczywistych serwisów wydziałowego systemu informatycznego

5.1 Wdrożenie przykładowych aplikacji

Wdrożenie utowrzonego systemu w ramach tej pracy polega na imporcie maszyny wirtualnej zawierającej gotowe środowisko i aplikacje do środowiska VMware ESX na wydziałowym systemie serwerowym. Po rozwiązaniu kliku pomniejszych problemów z konfiguracją dostarczonej maszyny wirtualnej udało się ostatecznie ją zainstalować i uruchomić. Zbudowne ówcześnie kontenery zostały wzbudzone komendą: `docker-compose up`. Operacja zakończyła się pełnym sukcesem, serwisy działały poprawnie i były widoczne dla wszystkich urządzeń sieci wydziałowej.

5.2 Wdrożenie rzeczywistych serwisów wydziałowego systemu informatycznego

Rozdział 6

Wnioski

Rozdział 7

Kod źródłowy

7.1 Repozytorium Git

7.2 Dołączona płyta CD

Bibliografia

[1] Httpd Apache. http://web.archive.org/web/20080207010024/http://www.808multimed
. Accessed: 2020-01-14.

[2] Apache Tomcat. http://web.archive.org/web/20080207010024/http://www.808multime
. Accessed: 2020-01-14.

[3] Coreos. http://web.archive.org/web/20080207010024/http://www.808multimedia.com
Accessed: 2020-01-14.

[4] Debian 10. http://web.archive.org/web/20080207010024/http://www.808multimedia.
Accessed: 2020-01-14.

[5] Django. http://web.archive.org/web/20080207010024/http://www.808multimedia.com
Accessed: 2020-01-14.

[6] Docker. http://web.archive.org/web/20080207010024/http://www.808multimedia.com
. Accessed: 2020-01-14.

[7] Docker-compose. http://web.archive.org/web/20080207010024/http://www.808multim
. Accessed: 2020-01-14.

[8] .Net. http://web.archive.org/web/20080207010024/http://www.808multimedia.com/w
Accessed: 2020-01-14.

[9] Microsoft hyper-v. http://web.archive.org/web/20080207010024/http://www.808multi
Accessed: 2020-01-14.

[10] Java. http://web.archive.org/web/20080207010024/http://www.808multimedia.com/w
Accessed: 2020-01-14.

[11] Lxc. http://web.archive.org/web/20080207010024/http://www.808multimedia.com/wi
Accessed: 2020-01-14.

[12] Mysql. http://web.archive.org/web/20080207010024/http://www.808multimedia.com/
Accessed: 2020-01-14.

[13] Php. http://web.archive.org/web/20080207010024/http://www.808multimedia.com/w
Accessed: 2020-01-14.

[14] Postgresql. http://web.archive.org/web/20080207010024/http://www.808multimedia.
Accessed: 2020-01-14.

[15] Python. http://web.archive.org/web/20080207010024/http://www.808multimedia.co
Accessed: 2020-01-14.

[16] QEMU. http://web.archive.org/web/20080207010024/http://www.808multimedia.com
Accessed: 2020-01-14.

[17] Oracle Solaris Containers. http://web.archive.org/web/20080207010024/http://www.80
Accessed: 2020-01-14.

[18] Spring Boot framework. http://web.archive.org/web/20080207010024/http://www.808m
Accessed: 2020-01-14.

[19] Oracle VirtualBox. http://web.archive.org/web/20080207010024/http://www.808multi
Accessed: 2020-01-14.

[20] VMware ESX. http://web.archive.org/web/20080207010024/http://www.808multimedi
. Accessed: 2020-01-14.

[21] VMware Player. http://web.archive.org/web/20080207010024/http://www.808multime
. Accessed: 2020-01-14.

[22] VMware Workstation. http://web.archive.org/web/20080207010024/http://www.808mu
. Accessed: 2020-01-14.

[23] Microsoft Windows. http://web.archive.org/web/20080207010024/http://www.808mult
. Accessed: 2020-01-14.

[24] Wine. http://web.archive.org/web/20080207010024/http://www.808multimedia.com/
. Accessed: 2020-01-14.

[25] Cms wordpress. http://web.archive.org/web/20080207010024/http://www.808multime
Accessed: 2020-01-14.

[26] Gianluca Borello. Container isolation gone wrong. 2017. URL https://sysdig.com/blog/container-isolation-gone-wrong/. Accessed: 2020-01-14.