Wyższa
Szkoła
Informatyki
i Umiejętności

WYŻSZA SZKOŁA INFORMATYKI I UMIEJĘTNOŚCI
Wydział Informatyki i Zarządzania
Kierunek: Informatyka

Sebastian Florek
nr albumu: 29571

Praca Magisterska
Monitorowanie otoczenia za pomocą mikrokontrolerów Raspberry Pi w oparciu o system zarządzania klastrem Kubernetes

Praca napisana pod kierunkiem
dr inż. Grzegorz Zwoliński

Rok akademicki 2016/2017

Spis treści

Rozdział 1

Wstęp

1.1 Problematyka i zakres pracy

Wraz z rozwojem Internetu Rzeczy na świecie powstają coraz to nowe urządzenia mające na celu automatyzację działań i ułatwienie życia człowieka. Dzięki ich zdolności do wzajemnej komunikacji, wymiany informacji oraz zdalnego zarządzania zasobami stają się one coraz bardziej popularne, a wręcz wymagane w życiu codziennym coraz większej grupy osób. Sterowanie oświetleniem, radiem czy innymi sprzętami elektronicznymi za pomocą naszego smartfona już nikogo nie dziwi. W wielu przypadkach urządzenia te muszą zbierać bardzo duże ilości danych oraz przesyłać je do centralnego punktu. Powoduje to ogromny wzrost ilości danych, które nie są w stanie zostać obsłużone przez jeden serwer. Powstaje więc potrzeba stworzenia niezawodnych, wydajnych i bezpiecznych systemów o wysokiej dostępności.

Technologie wirtualizacji [1] powstały w celu realizacji tych wymagań. Wirtualizacja serwerów dawno już wyparła tradycyjne serwery, które zostały zastąpione przez rozwiązania chmurowe. Niezależność sprzętowa, lepsza utylizacja zasobów, większe bezpieczeństwo, łatwa migracja danych i redukcja kosztów to tylko niektóre z wielu zalet wirtualizacji. Właśnie ta niezależność sprzętowa pozwala na coraz lepsze wykorzystanie mikrokontrolerów, których głównymi zaletami są mały koszt i niewielki pobór mocy, a dzięki coraz lepszej optymalizacji systemów i postępującej miniaturyzacji, również rosnąca wydajność.

Obecnie nie trzeba już wydawać ogromnych kwot w celu uruchomienia własnego klastra do zarządzania danymi, czy wyposażenia naszego mieszkania, a nawet biura w inteligentne czujniki i kontrolery. Wystarczą nam w tym celu mikrokontro-

[1] [?]

lery RaspberryPi, które dzięki systemom typu open source takim jak Kubernetes pozwolą nam na stworzenie w pełni funkcjonalnego klastra.

Proponowanym rozwiązaniem powyższych problemów będzie projekt o nazwie KubePi. Projekt ten skupia się na wirtualizacji opartej o kontenery Dockera [2] zarządzane przez system zarządzania klastrem Kubernetes i ma na celu stworzenie rozproszonego systemu służącego do monitorowania otoczenia. Przykładowy klaster opierać się będzie na dwóch mikrokontrolerach RaspberryPi. Do pierwszego urządzenia będącego zarazem głównym węzłem klastra podłączone zostaną trzy czujniki: temperatury, wilgotności oraz alkoholu. Jego zadaniem będzie udostępnianie zbieranych informacji. Drugie urządzenie zostanie natomiast wyposażone w wyświetlacz LED [3] , co pozwoli na odczyt i wyświetlanie temperatury raportowanej przez pierwsze urządzenie. Dodatkowo w klastrze zostanie zainstalowana aplikacja webowa [4] pozwalająca na zdalny monitoring. W celu lepszego zobrazowania komunikacji między urządzeniami zostanie ona uruchomiona na drugim urządzeniu.

1.2 Uzasadnienie wyboru tematu

W tej części opisane zostaną powody wyboru tematu związanego z systemem monitorowania otoczenia przy użyciu nowoczesnych technologii w oparciu o mikrokontrolery. Głównym powodem jest wykazanie użyteczności systemów chmurowych wysokiej dostępności w oparciu o mikrokontrolery. Wymaga to gwarancji pracy systemu nawet w razie uszkodzenia jednego z węzłów klastra. Drugim powodem jest zbadanie możliwości mikrokontrolerów jako alternatywy dla standardowych systemów chmurowych oraz porównanie wydajności względem ceny przy budowaniu klastra. Ze względu na małe rozmiary i koszt mikrokontrolery takie jak RaspberryPi mogą służyć jako tania alternatywa do budowy inteligentnego systemu dla naszego domu i nie tylko.

[2] [**?**]
[3] [**?**]
[4] [**?**]

1.3 Metoda badawcza

1.4 Przegląd literatury w dziedzinie

1.5 Układ pracy

Celem pracy jest zaproponowanie architektury i sprawdzenie w działaniu rozproszonego systemu wysokiej dostępności służącego do monitorowania otoczenia.

Rozdział pierwszy zawiera szczegółowy opis problemu. Zostają w nim przedstawione różne problemy związane z wydajnością oraz bezpieczeństwem tradycyjnych rozwiązań, wraz z opisem metod badawczych użytych do analizy tematu. Podsumowane zostają również główne założenia i cele pracy. Na koniec przeprowadzony zostaje przegląd literatury związanej z tematem, z naciskiem na kluczowe zagadnienia dotyczące wirtualizacji, rozwiązań chmurowych oraz mikrokontrolerów opartych na architekturze ARM, wraz z krótkim opisem użytych źródeł.

W rozdziale drugim przybliżona zostaje tematyka systemów zarządzania klastrami pod kątem ich wymagań, bezpieczeństwa oraz komunikacji sieciowej. Kolejnym krokiem jest dokładniejsze zapoznanie się z wirtualizacją, a konkretniej wirtualizacją opartą o kontenery Dockera, co pozwoli lepiej zrozumieć ideę pracy. Następnie po krótce przedstawione zostają tematyki związane z mikrokontrolerami oraz Internetem Rzeczy.

Rozdział 3 skupia się na analizie istniejących systemów zarządzania klastrami oraz ich pochodnych. Dodatkowo przedstawiona zostaje analiza kilku systemów typu Smart Home [5] .

Kolejny rozdział opisuję fazę projektowania i implementacji projektu KubePi. Spisane zostają wymagania funkcjonalne aplikacji, a także ograniczenia projektowe. Wymienione i opisane zostają użyte technologie. Opisany zostaje proces konfiguracji urządzeń, sieci oraz systemu. Następnie wskazane zostają kluczowe punkty aplikacji wraz z kodem źródłowym i opisem. W kolejny kroku przechodzimy do fazy testów stworzonych aplikacji jak i całego systemu.

W podsumowaniu pracy opisane zostają słabe i mocne strony przedstawionego rozwiązania. Na podstawie uzyskanych wyników następuje ocena możliwości i przydatności zaproponowanego rozwiązania. Na końcu omówione zostają możliwe perspektywy rozwoju projektu.

[5] [**?**]

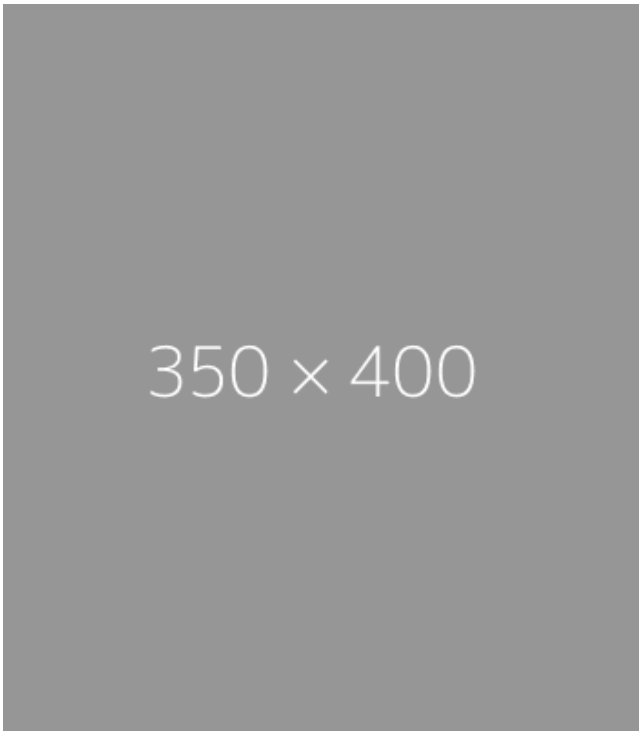
Rozdział 2

Podstawy teoretyczne

Użyte koncepcje i terminy używane w dalszej części pracy muszą zostać wyjaśnione w celu lepszego zrozumienia opisywanej problematyki. W kolejnych sekcjach zostają objaśnione podstawowe pojęcia związane z Internetem Rzeczy, przetwarzaniem danych w klastrze, tematyką wirtualizacji, rozproszonych systemów chmurowych oraz mikrokontrolerów. Następuje przedstawienie narzędzi do tworzenia, wdrażania i uruchamiania aplikacji rozproszonych w oparciu o kontenery aplikacji, takich jak Docker i Rkt. Następnie opisane zostaną systemy zarządzania kontenerami w klastrze takie jak Kubernetes czy trochę mniej zaawansowany Docker Swarm. Na koniec opisany zostanie system oparty o kontenery Dockera, który przenosi zalety kontenerów do Internetu Rzeczy, czyli Resin.io.

2.1 Internet Rzeczy

Internet Rzeczy zyskuje na popularności, głównie za sprawą swojego podejścia do komunikacji. Głównym założeniem jest umożliwienie urządzeniom oraz zwykłym przedmiotom codziennego użytku wzajemnej interakcji ze sobą jak i z użytkownikiem. Celem natomiast jest stworzenie inteligentnych urządzeń w celu zminimalizowania i jak największego uproszczenia interakcji użytkownika z takimi urządzeniami. Zaczynając od prostych czujników i wyświetlaczy a kończąc na autonomicznych pojazdach, budynkach czy całych miastach zarządzanych przez mikrokontrolery.



Rysunek 2.1: Przedstawienie Internetu Rzeczy w ujęciu graficznym

2.1.1 Obszary zastosowań

Należy zaznaczyć, że systemy realizacji Internetu Rzeczy są w zasadzie ograniczone tylko przez naszą wyobraźnię, a obejmują one między innymi:

- Inteligentne domy i budynki

  Interfejsy nowej generacji pozwolą nam nie martwić się o to czy zgasiliśmy światło, albo czy zamknęliśmy drzwi. System sam wykryje czy jesteśmy w domu lub pokoju i zrobi to za nas. Możliwość zarządzania urządzeniami codziennego użytku znacznie podwyższy nasz komfort i bezpieczeństwo.

- Inteligentne miasta

  Zaawansowane systemy pozwolą zoptymalizować infrastrukturę miejską. System sterowania ruchem dzięki zebranym danym sam dostosuje się do panujących warunków i pozwoli na rozładowanie ruchu w mieście. Sieć czujników może zostać użyta do wykrywania przestępstw czy sterowania oświetleniem w mieście.

- Inteligentne przedsiębiorstwa i przemysł

  Obecnie możemy doświadczyć dużych zmian w przedsiębiorstwach, które wykorzystują Internet Rzeczy w celu zarządzania całym łańcuchem produkcji i dostaw. Inteligentne maszyny same wiedzą w jakim są stanie i mogą reagować odpowiednio wcześnie w celu wymiany podzespołów czy przeprowadzenia konserwacji. Dla klienta natomiast dużym udogodnieniem mogą być zamówienia dostarczane przez drony, które są już testowane przez niektóre duże firmy takie jak Amazon czy UPS.

- Monitorowanie otoczenia i zagrożeń

  Rozległe sieci czujników już teraz pozwalają na całodobowe monitorowanie temperatury, opadów, wiatru, poziomu rzek, itp. Zebrane dane są przetwarzane i służą do wykrywania anomalii i przewidywania zdarzeń, które mogą zagrażać ludziom. W efekcie zwiększają one nasze ogólne bezpieczeństwo.

2.2 Chmury Obliczeniowe

Chmury obliczeniowe stają się coraz ważniejszym elementem Internetu Rzeczy. Dostarczają one swoje zasoby i usługi przy użyciu internetu. Słowo chmura opisuje sposób przetwarzania danych, który oderwany jest od naszego systemu, a przeprowadzany jest na zdalnych serwerach.

W szerszym ujęciu natomiast chmura jest centrum danych, w którym poszczególne węzły są wirtualizowane przy użyciu wirtualnych maszyn. Głównym powodem używania chmur obliczeniowych jest rozwiązywanie złożonych problemów oraz analiza dużych ilości danych. Wszystkie dane mogą być łatwo udostępniane innym osobom, a dostęp gwarantowany z każdego zakątka świata.

Bardziej formalna definicja chmur obliczeniowych przedstawia się następująco.

Chmura obliczeniowa 1 *Dostarczanie usług obliczeniowych, serwerów, magazynu, baz danych, sieci, oprogramowania analiz, itd. za pośrednictwem internetu. Firmy oferujące te usługi obliczeniowe są nazywane dostawcami chmury i zazwyczaj pobierają opłaty za usługi chmury obliczeniowej w zależności od użycia, podobnie jak dostawcy energii elektrycznej lub wody.*

2.2.1 Architektura

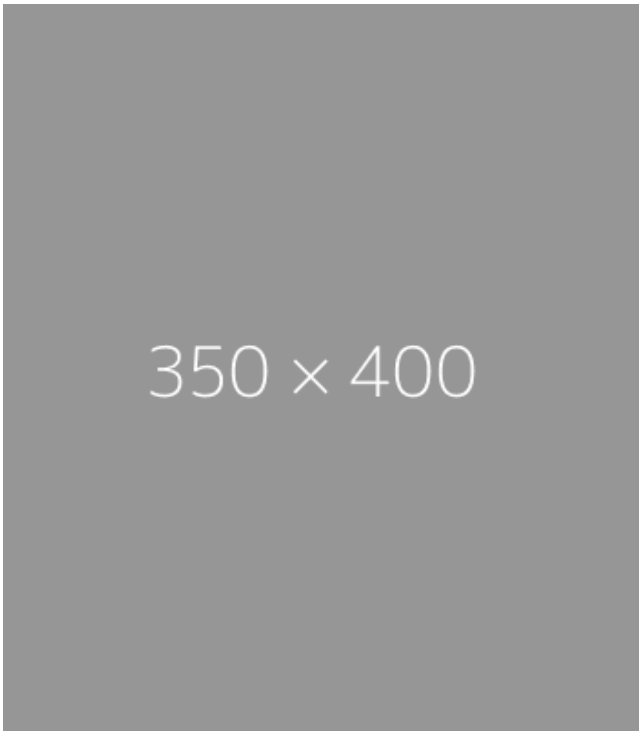
W związku z szybkim rozwojem chmur obliczeniowych, zostały one podzielone na trzy główne modele usług w celu utylizacji mocy obliczeniowej dla konkretnych potrzeb użytkowników. Modele te zostały zilustrowane na poniższym obrazku obrazującym architekturę chmur obliczeniowych. Architektura ta uwzględnia następujące podziały:

- Infrastruktura jako serwis (z ang. Infrastracture-as-a-Service) — znana również pod nazwą hardware-as-a-service[1]. Jest najprostszą odmianą usługi chmury obliczeniowej i pozwala na zdalny dostęp do zasobów sprzętowych. Dostarcza moc obliczeniową, przestrzeń dyskową i zasoby sieciowe. Jednym z najbardziej znanych dostawców takich usług jest Amazon Elastic Compute Cloud.

- Platforma jako serwis (z ang. Platform-as-a-Service) — usługa ta oferuje przede wszystkim środowisko deweloperskie oraz zbiór aplikacji, które mogą być używane do tworzenia i uruchamiania własnych aplikacji. Dostarcza bazy danych, serwery webowe, system operacyjny, środowisko uruchomieniowe, przestrzeń dyskową i wiele innych zasobów. Głównymi dostawcami tych usług są obecnie Microsoft Azure[2] oraz Google App Engine[3]

- Oprogramowanie jako serwis (z ang. Software-as-a-Service) — ostatni z modeli usług chmury obliczeniowej, który oferuje przechowywanie i uruchomienie aplikacji klienta na własnych serwerach oraz udostępnienie jej użytkownikom przez internet. Pozwala to na wyeliminowanie potrzeby instalacji oraz uruchamiania aplikacji na komputerze klienta. W efekcie to dostawca usług ma obowiązek zapewnienia ciągłości działania naszej aplikacji. Najbardziej znane usługi SaaS to np. Google Docs czy Google Apps.

[1] [?]
[2] [?]
[3] [?]



Rysunek 2.2: Architektura chmur obliczeniowych

2.3 Wirtualizacja

Ze względu na gwałtowny rozwój internetu, zapotrzebowanie na zasoby sprzętowe również wzrasta. Utylizacja i optymalne wykorzystanie zasobów zaczęły odgrywać kluczową rolę. Uruchamianie i zapewnienie ciągłości działania dużym aplikacjom sieciowym stworzyło potrzebę jak najlepszego zarządzania zasobami w celu maksymalnego wykorzystania dostępnych zasobów. Jednym z rozwiązań tych problemów jest właśnie wirtualizacja, czyli stworzenie wirtualnego środowiska zbudowanego w oparciu o infrastrukturę sieciową i dostarczenie go użytkownikom końcowym.

Wirtualizacja to tak na prawdę oddzielenie dostępnych zasobów od fizycznego sprzętu. Pozwala to uruchomić wiele systemów na tej samej platformie sprzętowej i systemowej w celu uzyskania jak najlepszej wydajności oraz utylizacji zasobów. Przy użyciu jednej platformy sprzętowej można dostarczyć usługi wielu użytkownikom końcowym. Dwoma najbardziej znanymi podejściami do wirtualizacji są: wirtualizacja oparta o nadzorcę oraz o kontenery. Zostaną one opisane w następnych sekcjach.

2.3.1 Wirtualizacja oparta o nadzorcę

Podejście to w ciągu ostatniej dekady było szeroko używane do tworzenia środowisk wirtualnych dla dużych systemów obliczeniowych. Nadzorca znany również jako monitor wirtualnej maszyny jest oprogramowaniem, które ma za zadanie tworzenie, monitorowanie wirtualnych środowisk oraz przydzielanie im zasobów. Przykładem może być uruchomienie systemu operacyjnego Windows jako maszyny wirtualnej na systemie operacyjnym Linux.

Dodatkowo wirtualizacja dostarcza nam niezależne środowisko, które może służyć do uruchamiania aplikacji w izolowanym środowisku bez ingerencji w inne aplikacje. Nadzorców możemy podzielić na dwa typy, które zostały zilustrowane na poniższym rysunku:

- Native/Bare metal hypervisor — ten nadzorca działa bezpośrednio na poziomie sprzętu. Dzięki pełnej kontroli nad sprzętem, może kontrolować i monitorować uruchomione systemy operacyjne, które działają poziom wyżej niż nadzorca. Popularnymi przykładami takich nadzorców są: Oracle VM, Microsoft Hyper-V czy VMWare ESX.
- Host based hypervisor — ten nadzorca działa jako program uruchomiony na danym systemie operacyjnym (hoście). Pełni on rolę emulatora i izoluje wirtualne systemy operacyjne od sprzętu jak i od hosta, czyli działa dwa poziomy ponad sprzętem. Popularnymi przykłądami takich nadzorców są: VirtualBox, VMWare Workstation, KVM.

Rysunek 2.3: Architektura chmur obliczeniowych

Właściwości nadzorcy

Według książki [odnosnik] nadzorcy wyróżniają się następującymi właściwościami:

1. Transparentność — pozwala uruchamiać oprogramowanie na maszynie wirtualnej bez żadnych modyfikacji i niezależnie od sprzętu oraz umożliwia dzielenie zasobów sprzętowych przez wiele maszyn wirtualnych.

2. Izolacja — umożliwia nadzorcy tworzenie i uruchamianie niezależnych, odizolowanych od siebie środowisk wirtualnych w obrębie jednego fizycznego systemu. Główną zaletą jest zapobieganie oddziaływania błędów aplikacji na inne aplikacje uruchomione w innych środowiskach wirtualnych.

3. Enkapsulacja — zapewnia elastyczność i bezpieczeństwo aplikacji uruchomionych w środowisku wirtualnym poprzez zamknięcie systemu w obrębie wirtualnego dysku twardego. Dzięki temu instalacja i tworzenie kopii zapasowych maszyn wirtualnych sprowadza się do kopiowania plików.

4. Łatwość zarządzania — wbudowane opcje nadzorcy do zamykania, restartowania, uruchamiania, usypiania, dodawania oraz usuwania wirtualnych maszyn umożliwiają łatwe zarządzanie wieloma maszynami wirtualnymi.

2.3.2 Wirtualizacja oparta o kontenery

Wirtualizacja oparta o kontenery jest mniej wymagającym w kontekście zasobów podejściem do wirtualizacji. Operuje ona na poziomie systemu operacyjnego i tworzy środowiska wirtualne jako procesy systemowe, co pozwala na dzielenie zasobów sprzętowych z systemem operacyjnym. Wprowadza ona pewien poziom abstrakcji, w którym jądro systemu jest dzielone pomiędzy kontenery i umożliwia uruchamianie więcej niż jednego procesu w obrębie każdego kontenera. Dzięki takiemu rozwiązaniu nie ma potrzeby uruchamiani pełnego systemu operacyjnego odizolowanego od systemu operacyjnego hosta. Przekłada się to bezpośrednio na oszczędność zużycia pamięci, czasu procesora i przestrzeni dyskowej.

Na poniższym rysunku możemy zobaczyć cztery kontenery uruchomione w systemie operacyjnym. Zasoby sprzętowe dzielone są pomiędzy nadrzędny system operacyjny oraz systemy operacyjne gości (kontenery). Zaletą tego typu wirtualizacji jest możliwość uruchomienia dużej ilości kontenerów w obrębie jednego systemu operacyjnego. Ma to również swoje wady, a mianowicie nie możemy przykładowo uruchomić systemu operacyjnego Windows jako kontener aplikacji na hoście z systemem operacyjnym Linux. Dodatkowo w porównaniu ze standardowymi nadzorcami kontenery nie gwarantują odpowiedniej izolacji zasobów, co

może mieć wpływ na bezpieczeństwo. Funkcje wirtualizacji przy użyciu kontenerów wykorzystują dwie podstawowe właściwości jądra systemu: grupy kontrolne(cgroups) oraz przestrzenie nazw(namespaces).

Rysunek 2.4: Architektura wirtualizacji opartej o kontenery

Grupy kontrolne

Grupy kontrolne to jedna z głównych funkcji jądra systemu, która pozwala użytkownikom alokować oraz ograniczać zasoby pomiędzy grupami procesów. Zasoby te to między innymi czas procesora, pamięć systemowa, przestrzeń dyskowa czy wykorzystanie sieci. Grupy kontrolne umożliwiają również sprawne zarządzanie zasobami przeznaczonymi dla kontenerów i działających w nich aplikacji. Przykładowo jeśli aplikacja tworzy dwa procesy z odrębnymi zasobami, może być rozdzielona na dwie grupy kontrolne z różnymi limitami użycia zasobów. Dodatkowo dzięki użyciu grup kontrolnych można w łatwy sposób monitorować i odmawiać lub przyznawać zasoby odpowiednim procesom. Możliwa jest również dynamiczna konfiguracja w trakcie działania systemu. Dzięki temu administrator może w łatwy sposób kontrolować zarządzanie zasobami w systemie.

Przestrzenie nazw

Przestrzenie nazw to kolejna ważna funkcjonalność dostępna w jądrze systemu, która pozwala na izolacje procesów wewnątrz kontenerów. Gwarantuje ona każdemu procesowi dedykowaną przestrzeń, w której może on działać, bez ingerencji w procesy uruchomione w innych przestrzeniach nazw. Dodatkowo zapewnia

izolację środowiska uruchomieniowego procesów. Obecnie w systemach Linux możemy znaleźć sześć domyślnych przestrzeni nazw:

- zamontowanych systemów plików (mnt) — zawiera drzewo katalogów z zamontowanymi systemami plików. Podstawowy system plików jest montowany w korzeniu drzewa w czasie uruchamiania systemu. Utworzenie osobnej przestrzeni nazw systemu plików dla grupy procesów umożliwia zamontowania sytemu plików, który będzie widoczny tylko dla tej grupy procesów.
- stacji roboczej i domeny (uts) — przechowuje aktualną nazwę stacji roboczej oraz nazwę domeny, do której należy stacja robocza. Utworzenie osobnej przestrzeni nazw UTS umożliwia zmianę tych parametrów dla poszczególnej grupy procesów.
- identyfikatorów procesów (pid) — umożliwia istnienie wielu procesów w różnych przestrzeniach nazw z tym samym identyfikatorem procesu. Użyteczna przy przenoszeniu grupy procesów między maszynami bez zmiany ich identyfikatorów.
- obiektów IPC (ipc) — pozwala na tworzenie systemowych obiektów do komunikacji między procesami, widocznych tylko dla grupy procesów, które współdzielą dana przestrzeń nazw IPC.
- użytkowników (user) — pozwala na odizolowanie procesów bazując na identyfikatorze użytkownika i grupy, która uruchomiła dany proces. Dzięki temu dany proces może być uruchomiony z przywilejami administratora w konkretnej przestrzeni nazw, natomiast w innych działać jako proces z prawami zwykłego użytkownika.
- zasobów sieciowych (net) — pozwala na odizolowanie podsystemu sieciowego dla grupy procesów. Dzięki temu grupa procesów może konfigurować urządzenia sieciowe oraz nawiązywać połączenia niezależnie od reszty systemu. Taka grupa posiada własne urządzenia sieciowe, adresy IP, trasy, itp.

Właściwości kontenerów

Według [dodac ksiazke], możemy wyróżnić cztery właściwości kontenerów Linuxa, które sprawiają, że są one atrakcyjne dla użytkowników:

1. Przenośność — kontenery mogą działać na wielu różnych środowiskach i nie wymagają dodatkowych kroków w celu dostosowania systemu operacyjnego. Dodatkowo można uruchamiać wiele aplikacji w obrębie jednego kontenera, a następnie uruchamiać je na różnych środowiskach.

2. Szybkość tworzenia — kontenery są łatwe i szybkie w budowie. W porównaniu z tworzeniem standardowej maszyny wirtualnej ich budowa i start zajmują sekundy, a nie minuty. Dzięki temu administratorzy mogą w łatwy sposób uruchamiać kontenery w środowisku produkcyjnym. Dodatkowo skracają czas potrzebny na stworzenie aplikacji uruchamianej w kontenerze. W łatwy sposób pozwalają na dzielenie się aplikacjami z zespołem i testowanie w różnych środowiskach.

3. Skalowalność — tworzenie i instalacja kontenerów na dowolnym systemie czy w chmurze obliczeniowej są bardzo proste. Dużym plusem jest również skalowalność kontenerów. W prosty sposób możemy zautomatyzować tworzenie czy usuwanie kontenerów bazując na aktualnym zużyciu zasobów sprzętowych. Dzięki temu idealnie nadają się one do instalacji w chmurze.

4. Elastyczność — przy użyciu kontenerów możemy w prosty sposób uruchomić dużą liczbę aplikacji na pojedynczym systemie operacyjnym. Biorąc pod uwagę, że nie uruchamiają one pełnego systemu operacyjnego, możemy w łatwy i wydajny sposób zarządzać wieloma aplikacjami.

Po zapoznaniu się z technikami wirtualizacji opartymi o nadzorcę i kontenery możemy w łatwy i czytelny sposób przedstawić ich porównanie w formie poniższej tabeli.

Rysunek 2.5: Architektura wirtualizacji opartej o kontenery

2.3.3 Docker

Docker jest jednym z najpopularniejszych narzędzi bazujących na wirtualizacji opartej o kontenery. Został on po raz pierwszy przedstawiony przez Solomona Hykesa, 15 marca 2013 roku. W tamtym czasie niewiele, bo około 40 osób uzyskało szansę poznania tego narzędzia.

Docker jest narzędziem które w łatwy sposób pozwala na tworzenie i dystrybucję aplikacji na różne środowiska. Dodatkowo umożliwia proste skalowanie i konfigurację naszych aplikacji. Pozwala w znaczący sposób skrócić proces tworzenia i testowania oprogramowania oraz natywnie korzysta z dwóch opisanych wcześniej funkcji jądra systemu, a mianowicie grup kontrolnych i przestrzeni nazw. Ważne jest również to, że Docker pozwala uruchomić niemal każdą aplikację bezpiecznie w kontenerze, dostarczając izolacjęi bezpieczeństwo na poziomie systemu. Pozwala to uruchamiać wiele kontenerów jednocześnie bez obaw, że będą one oddziaływały na siebie w jakikolwiek szkodliwy sposób. Wystarczy jedynie minimalny system operacji ze zgodną wersją jądra systemu oraz plik wykonalny Dockera.

W związku z tym, że aplikacje zostają uruchomione w izolowanym środowisku, Docker udostępnia narzędzia dla ułatwienia budowy, uruchamiania i zarządzania kontenerami. Oprócz możliwości uruchomienia go na lokalnym komputerze, istnieje również szereg dostawców usług, którzy wspierają kontenery Dockera, np. Google Cloud Platform, Microsoft Azure, Amazon EC2. Istnieje również platforma o nazwie Resin.io, która oferuje wsparcie systemów opartych na Dockerze dla urządzeń z zakresu Internetu Rzeczy, jednak dokładniej zapoznamy się z nią w rozdziale [link 2.8 Resin.io].

Cele Dockera

Głównymi celami Dockera sa:

- Dostarczenie łatwego sposobu modelowania złożonych systemów wymagających wielu współpracujących aplikacji. Dzięki użytym mechanizmom jest on bardzo szybki i łatwy do dostosowania do własnych potrzeb. Dodatkowo uruchamianie kontenerów aplikacji zajmuje w większości przypadków niespełna sekundę oraz nie wymaga uruchamiania nadzorców i zarządza naszymi zasobami systemowymi w bardziej optymalny sposób.

- Usprawnienie cyklu produkcji poprzez jego przyspieszenie i bardziej efektywne zarządzanie zasobami. Głównym celem jest tu redukcja czasu potrzebnego na tworzenie i testowanie oprogramowania, a zapewnione jest to głownie dzięki zapewnieniu możliwości uruchamiania tej samej aplikacji w wielu środowiskach bez dodatkowych zmian.

- Spójność aplikacji uruchamianych na różnych środowiskach, dzięki użyciu wirtualizacji opartej o kontenery. Zapewnia to, że nasza aplikacja będzie zachowywała się tak samo niezależnie od tego na jakim systemie uruchamiany jest kontener z nią.

Architektura Dockera

Docker bazuje na modelu klient-serwer. Klient Dockera komunikuje się z demonem[4], który odpowiada za tworzenie, uruchamianie oraz dystrybucję kontenerów. Zazwyczaj oba te komponenty uruchamiane są na tej samej maszynie, jednak możliwe jest uruchomienie demona na zdalnej maszynie i podłączenie się do niej za pomocą aplikacji klienckiej. Architektura została pokazana na poniższym rysunku. Każdy z komponentów ma określone zadania, które zostaną opisane poniżej.

Rysunek 2.6: Architektura wirtualizacji opartej o kontenery

Demon Dockera

Głównym zadanie tego komponentu jest zarządzanie działającymi kontenerami. Jak widać na powyższym rysunku demon uruchomiony jest na systemie, na którym działają kontenery. Klient natomiast używany jest w celu komunikacji z demonem, ponieważ użytkownik nie może bezpośrednio się z nim komunikować.

[4] [**?**]

Klient Dockera

Aplikacja kliencka Dockera jest, interfejsem uruchamianym z poziomu konsoli. Służy ona do komunikacji z procesem demona. Użytkownik poprzez odpowiednie komendy klienta może pośrednio komunikować się z demonem.

Obrazy Dockera

Obrazy są podstawowym źródłem służącym do tworzenia kontenerów. W pewnym sensie są one kodem źródłowym kontenerów. Obrazy można budować samemu praktycznie od zera lub użyć już istniejących obrazów dostępnych na platformach takich jak DockerHub[5]. Są one łatwe w konfiguracji i tworzeniu, a dzięki możliwości łatwego dzielenia się nimi, również łatwo dostępne.

Obrazy złożone są z wielu warstw instrukcji. Przykładowo, użytkownik może zdefiniować, aby przy tworzeniu obrazu uruchomiona została odpowiednia komenda systemowa, skopiowany folder czy ściągnięte dodatkowe zależności potrzebne do uruchomienia aplikacji. Następnie wszystkie warstwy są łączone w celu stworzenia obrazu. Dzięki takiemu podejściu do tworzenia obrazów, jeśli aktualizujemy tylko naszą aplikację, nie wymaga to przebudowy innych warstw, a jedynie aktualizacji warstwy zawierającej naszą aplikację.

Rejestr Dockera

Rejestry używane są do przechowywania stworzonych obrazów. Mogą być one prywatne lub publiczne. Publicznym rejestrem jest przykładowo wspomniany wcześniej DockerHub. Jest to jedna z najpopularniejszych platform służąca do przechowywania obrazów Dockera. Możemy z łatwością znaleźć obraz serwera dla naszej aplikacji webowej, bazę danych czy nawet cały system operacyjny. Jeśli nie chcemy natomiast dzielić się naszymi obrazami z innymi, możemy z łatwością skorzystać z prywatnych rejestrów, które również mogą być uruchamiane jako kontenery Dockera i udostępniać je tylko w obrębie naszej organizacji.

Kontenery

Jak już wcześniej wspomniano, kontenery tworzone są przy użyciu obrazów, które mogą zawierać również aplikacje i różne serwisy. Uruchamiają aplikacje w odizolowanym środowisku zawierającym wszelki zależności wymagane do ich działania.

[5] [**?**]

2.4 Systemy zarządzania kontenerami

2.4.1 Kubernetes

Architektura

2.4.2 Docker Swarm

2.4.3 Resin.io

2.5 Mikrokontroler RaspberryPi

2.5.1 Protokoły komunikacji

Rozdział 3

Projekt KubePi

3.1 Analiza wymagań

3.2 Użyte technologie

3.3 Projekt

3.4 Opis użytkowania

3.5 Przykład użycia

3.6 Możliwości rozszerzania aplikacji

Rozdział 4

Podsumowanie

Bibliografia

[1] Some books

Spis rysunków

Spis tablic