Politechnika
Śląska

PROJEKT INŻYNIERSKI

System do obsługi pracowni informatycznych z wykorzystaniem maszyn wirtualnych

Wojciech JANOTA
Nr albumu: ⟨**290357**⟩

Kierunek: ⟨Informatyka⟩
Specjalność: ⟨Bazy Danych i Inżynieria Systemów⟩

PROWADZĄCY PRACĘ
⟨**dr inż. Błażej Adamczyk**⟩
KATEDRA ⟨**Katedra Sieci i Systemów Komputerowych**⟩
Wydział Automatyki, Elektroniki i Informatyki

Gliwice 2023

Tytuł pracy

System do obsługi pracowni informatycznych z wykorzystaniem maszyn wirtualnych

Streszczenie

Tematem pracy jest stworzenie systemu do zarządzania pracowniami informatycznymi używając możliwości oferowanych przez maszyny wirtualne. Proponowana solucja umożliwia zdalne, zautomatyzowane przygotowanie pracowni informatycznej do zajęć lekcyjnych, minimalizując wymaganą liczbę manualnych kroków.

Słowa kluczowe

automatyzacja,PXE,Python,Ansible,Linux,QEMU

Thesis title

A system for supporting IT labs with the use of virtual machines

Abstract

Topic of this thesis is a automated system for managing IT labs using virtual machines. Solution allows for remote, automated preparation of the lab for classes, minimizing number of manual steps required.

Key words

automation,PXE,Python,Ansible,Linux,QEMU

Spis treści

Rozdział 1

Wstęp

Wprowadzenie Jednym z wielu problemów, z jakimi musi zmierzyć się administrator systemów (na przykład w placówce edukacyjnej), jest konfiguracja i zarządzanie flotą wielu maszyn, użytkowanych często przez osoby nietechniczne. Powstało wiele narzędzi służących do ułatwienia tego zadania, wśród których zwrócono uwagę na kilka przykładów:

- Microsoft Active Domain jako system do zarządzania już skonfigurowanymi maszynami opartymi o system Microsoft Windows
- Ansible jako system do zarządzania maszynami opartymi o systemy z rodziny UNIX
- Ubuntu Landscape jako system służący do zarządzania flotą maszyn opartych o system Ubuntu Linux
- Microsoft Windows Unattended Install, czyli narzędzie służące do automatycznego konfigurowania instalacji systemu Microsoft Windows

Każde z tych narzędzi pozwala na zautomatyzowanie jednego z podstawowych kroków w procesie zarządzania i utrzymania pracowni informatycznej: instalacji systemu/systemów operacyjnych, konfiguracji systemu, aktualizacji i utrzymania systemu. Pewne przypadki użycia wymagają jednak pewnych cech, które bardzo trudno osiągnąć używając powyższych narzędzi. Przykładem takiej sytuacji jest resetowanie urządzeń po każdych zajęciach lekcyjnych, zapewniając przy tym, że każde środowisko na którym pracują uczniowie/studenci jest identyczne. W takiej sytuacji najczęściej wykorzystuje się maszyny wirtualne, które po zakończeniu zajęć są przywracane do migawki, lub ich obraz dysku jest podmieniany na oryginalny. Wymaga to jednak kilku manualnych kroków, które należy wykonać pomiędzy zajęciami.

Proponowane rozwiązanie automatyzuje proces dystrybucji obrazów maszyn wirtualnych, instalacji oraz konfiguracji systemu operacyjnego, pod kontrolą którego będą pracować maszyny wirtualne. Celem tej pracy było:

- napisanie programu serwera obrazów maszyn wirtualnych, którego zadaniem jest ich rejestrowanie, przypisywanie oraz dystrybuowanie
- stworzenie klienta synchronizującego stan maszyny klienckiej ze stanem obecnym na serwerze, pobierającego obrazy, wyświetlającego ekran wyboru systemy do uruchomienia oraz obsługującego ich uruchmianie poprzez mechanizm QEMU wraz z KVM
- przygotowanie konfiguracji dla serwera opartego o system operacyjny Linux:
  - do automatycznego instalowania systemu nadzorcy dla maszyn klienckich
  - do obsługi sieciowej podłączonych maszyn (przydzielanie adresów IP, wskazywanie na serwer konfiguracji)
  - do zarządzania maszynami klienckimi
- wdrożenie rozwiązania w symulowanym środowisku testowym

Prezentowana praca podzielona została na kilka rozdziałów:

- analiza tematu: przybliżenie wykorzystanych technologii, ich charakterystyka, historia i zastosowania
- wymagania i narzędzia: jakie są wymagania dla aplikacji, jakie narzędzia zostały wykorzystane do rozwiązania problemu, ich opis oraz uzasadnienie wyboru
- specyfikacja zewnętrzna: specyfikacja opisująca całe rozwiązanie z perspektywy użytkowników końcowych, z wyszczególnionymi elementami składowymi rozwiązania
- specyfikacja wewnętrzna: specyfikacja opisująca techniczne aspekty rozwiązania, z perspektywy osoby technicznej zaznamiającej się z kodem źródłowym oprogramowania
- weryfikacja i walidacja: wyniki testów przeprowadzonych na testowym środowisku wdrożeniowym, ich analiza
- podsumowanie i wnioski: prezentacja wniosków wynikających z analizy wyników testów, krytyka proponowanego rozwiązania i propozycje poprawek

Wkład pracy autora Przedmiotem pracy było napisanie aplikacji kontrolera floty urządzeń, aplikacji klienta zarządzającej systemem nadzorcy, konfiguracji fizycznego serwera oraz samo jego skonfigurowanie. Wszystkie te zadania zostały zrealizowane, wraz z zakupem maszyny w roli serwera oraz skonfigurowaniem domowej sieci lokalnej do obsługi rozwiązania.

Rozdział 2

Analiza tematu

Praca została podzielona na dwie główne części: obsługę maszyn wirtualnych i ich obrazów, oraz zarządzanie, instalację i konfigurację systemu zarządcy maszyn wirtualnych.

Istnieje kilka popularnych sposobów synchronizacji i przesyłania plików pomiędzy urządzeniami opartymi o systemy z rodziny Linux. Wśród nich warto zwrócić uwagę na poniższe rozwiązania:

- RSYNC [11]
- SFTP [10]
- serwer HTTP serwujący pliki statyczne

RSYNC RSYNC to narzędzie służące do synchronizacji, kopiowania i przenoszenia plików na maszynach pracujących pod kontrolą systemów z rodziny UNIX. Podstawą działania aplikacji RSYNC jest architektura klient-serwer, gdzie klient najpierw nawiązuje połączenie z serwerem poprzez potok, a w przypadku połączenia sieciowego najpierw uruchamia powłokę na maszynie zdalnej, następnie uruchamia proces serwera i tworzy potok do komunikacji między nimi. Po nawiązaniu połączenia, proces wysyłający dane tworzy listę wszystkich plików do wysłania, wraz z informacjami o własności, wielkości plików, trybie, uprawnieniach i czasie ostatniej zmiany. Następnie, każda ze stron transferu sortuje te listy leksykograficznie, względem ścieżki do bazowego katalogu transferu. W kolejnym kroku tzw. generator iteruje się po liście plików i sprawdza, czy mogą zostać pominięte (czy data ostatniej zmiany i wielkość nie uległy zmianie, lub, jeżeli została wybrana odpowiednia flaga, czy sumy kontrolne poszczególnych plików nie uległy zmianie) oraz tworzy brakujące katalogi. Jeżeli plik nie został oznaczony jako możliwy do pominięcia, jego obecna wersja staje się "plikiem bazowym", a jej obliczone sumy kontrolne bloków plików są wysyłane do procesu odbierającego dane. Proces odczytuje tablicę sum kontrolnych bloków, tworzy indeks funkcji skrótu, aby przyspieszyć operacje wyszukiwania bloków. Następnie plik lokalny jest odczytywany i liczona jest suma kontrolna dla

pierwszego bajtu pliku. Jeżeli wykryta zostanie różnica w sumach kontrolnych, do pierwszego niepasującego bajtu dołaczony zostanie odpowiadający bajt z pliku wysyłanego, a następnie obliczona zostanie suma kontrolna bloku zaczynającego się w kolejnym bajcie. Jeżeli znalezione zostanie dopasowanie w tablicy indeksów funkcji skrótów, dany blok jest traktowany jako pasujący i jest pomijany. W ten sposób proces jest w stanie zrekontruować plik źródłowy. Wszystkie dane z tej analizy wysyłane są do procesu odbierającego, który na ich podstawie odtwarza oryginalny plik. Mechanizm „rolling checksum" opisany powyżej jest integralną częścią algorytmu RSYNC, który sprawia, że nie ma potrzeby transferu całych plików, jedynie zmian, które w nich nastąpiły.[18][24][25]

SFTP SFTP (Secure File Transfer Protocol) to bezpieczny protokół przesyłu plików wykorzystujący protokół SSH do uwierzytelnienia oraz szyfrowania przesyłanych danych. Pierwszym krokiem do przesyłu danych jest nawiązanie połączenia SSH pomiędzy urządzeniami, następnie serwer SFTP sprawdza dostęp do maszyny klienta poprzez SSH (Secure SHell). Jeżeli test przebiegnie pomyślnie, nawiązywane jest połączenie SFTP, przez które rozpoczyna się transfer plików. Protokół obsługuje przesył wielowątkowy, gdzie każda operacja transferu ma przypisany unikalny numer identyfikacyjny, nadawany przez klienta, dzięki czemu serwer może odpowiadać na zapytania klienta asynchronicznie i bez zachowania kolejności. Aby poprawić wydajność, klient wysyła często wiele zapytań zanim zatrzymuje się i czeka na odpowiedzi.[20]

SSH SSH (Secure SHell) to standard protokołów, który miał w założeniu stanowić bezpieczną alternatywę dla protokołu telnet. Do autoryzacji wykorzystuje kryptografię klucz publiczy - klucz prywatny, gdzie w najprostszej postaci pary kluczy sa generowane automatycznie i służą do zabezpieczenia połączenia sieciowego, natomiast sama autoryzacja odbywa się za pomocą hasła użytkownika systemu zdalnego.[15] Obecnie najpopularniejszą implementacją protokołu SSH jest OpenSSH, stworzona przez część członków projektu OpenBSD. Jest wykorzystywana przez większość najpopularniejszych systemów operacyjnych, takich jak różne dystrybucje Linux, macOS, Microsoft Windows, czy OpenBSD.[19]

Serwer HTTP Protokół HTTP (HyperText Transfer Protocol) powstał jako część WWW (World Wide Web) i miał służyć do transferu dokumentów hipertekstowych. Prace nad nim były prowadzone przez Tima Berners-Lee zatrudnionego w CERN (Conseil Européen pour la Recherche Nucléaire) od 1989 roku (pierwsza propozycja systemu hipertekstowego opartego o Internet), gdzie implementacja nastąpiła w 1990 roku, by już rok później pojawiły się pierwsze serwery poza organizacją macierzystą twórcy[6]. W 1997 roku powstała pierwsza ustandaryzowana wersja protokołu, HTTP/1.1, która została wykorzystana w tej pracy.[9] HTTP to obecnie bardzo wszechstronne narzędzie, pozwalające obecnie na przesyłanie dowolnych danych poprzez sieć Internet. Jako protokół

jest podstawą współczesnej sieci Web, służy do obustronnego transferu danych pomiędzy serwerami i klientami. Komunikacja jest rozpoczynana zawsze przez klienta. Tok takiej tranzakcji jest następujący (opis dotyczy wersji HTTP/1.1):

- otwarcie połączenia TCP (Transmission Control Protocol) z serwerem
- wysłanie zapytania HTTP
- odebranie odpowiedzi HTTP od serwera
- zamknięcie połączenia, lub ponowne jego użycie (do wysłania kolejnego zapytania)

Wiadomości HTTP (dla wersji HTTP/1.1 i starszych) mają ustandaryzowany format, czytelny dla człowieka i zapisywalny w postaci tekstu. Format różni się dla zapytań i odpowiedzi, gdzie zapytanie posiada następujące pola[5]:

- metoda HTTP: definiuje operację, którą chce przeprowadzić klient. Może to być np. operacja pobrania danych (metoda GET), czy wysłania danych (metoda POST). Standard HTTP/1.1 zawiera poniższe metody[7]:
  - CONNECT
  - DELETE
  - GET
  - HEAD
  - OPTIONS
  - POST
  - PUT
  - TRACE
- ścieżka dostępu do danych
- wersja protokołu HTTP
- opcjonalne nagłówki
- ciało zapytania, dla metod obsługujących przesył danych z klienta do serwera

Natomiast struktura odpowiedzi jest następująca[5]:

- wersja protokołu HTTP
- kod statusu: jeden z kodów HTTP oznaczający czy zapytanie zostało obsłużone oraz jaki był efekt jego obsługi

- wiadomość statusu: krótka wiadomość opisująca znaczenie kodu statusu
- nagłówki HTTP, podobnie do zapytania
- ciało odpowiedzi, jeżeli wymagane

Z perspektywy prezentowanego projektu ważną cechą protokołu HTTP jest możliwość przesyłu dowolnych danych w ten sam sposób. Zostało to wykorzystane do stworzenia serwera, który potrafi przesyłać dane do klientów w formacie JSON (JavaScript Object Notation), czyli tekstowym formacie opisującym (w tym konkretnym przypadku) konfigurację oraz w formacie binarnym, co jest wykorzystywane do przesyłu obrazów maszyn wirtualnych. Powyższe rozwiązanie zostało wybrane właśnie z powodu łatwości połączenia części przesyłającej konfigurację z serwera do klientów oraz części przesyłającej obrazy. W dalszej części pracy opisano dokładną architekturę tego rozwiązania, jej zalety oraz wady.

Maszyny wirtualne Maszyna wirtualna to w najprostszym ujęciu programowa implementacja fizycznego urządzenia, która wykonuje zadany program w sposób identyczny do tego urządzenia. Aplikacja, która to umożliwia jest nazywana nadzorcą (ang. hypervisor). Jej zadaniem jest tłumaczenie zapytań programów do natywnego sprzętu na zapytania do sprzętu, na którym działa uruchomiona maszyna wirtualna. Istnieje kilka popularnych rodzajów wirtualizacji[3]:

- pełna wirtualizacja z translacją binarną, rys. 2.2 przedstawia schemat działania takiego systemu
- wirtualizacja ze wsparciem sprzętowym, rys. 2.3 przedstawia schemat działania takiego systemu
- wirtualizacja wspierana przez system operacyjny i parawirtualizacja, rys. 2.1 przedstawia schemat działania takiego systemu

W prezentowanym projekcie wykorzystano model ze sprzętowym wsparciem dla wirtualizacji poprzez KVM (Kernel Virtual Machine) wraz z QEMU (Quick EMUlator) i technologie Intel VT-d/AMD-V, pod kontrolą systemu operacyjnego Ubuntu Linux w wersji 22.04 LTS.

KVM KVM to technologia obsługi akceleratorów sprzętowych wirtualizacji wbudowana w systemy z rodziny Linux oparte o jądro w wersji co najmniej 2.6.20, a jej część przeznaczona dla użytkownika została wbudowana w narzędzie QEMU w wersji 1.3.[14]

Intel VT-d/AMD-V Technologie Intel VT-d/AMD-V to zbiór rozwiązań technicznych zaimplementowanych podczas tworzenia procesora oraz chipsetu płyty głównej, które wspierają i ułatwiają używanie maszyn wirtualnych. Sposób działania został opisany na podstawie technologii Intel VT-d.[12] Wsparcie dla wirtualizacji zostało tam osiągnięte poprzez zapewnienie:

- zestawu rozszerzeń dla procesora (nazwanych Intel VMX, Virtual Machine eXtensions)
- wirtualizacji urządzeń We/Wy (Wejścia/Wyjścia)
- wirtualizacji dostępu do pamięci

Wirtualizacja dostępu do pamięci jest zapewniana poprzez mechanizm MMU (Memory Management Unit). Jest to układ tłumaczący adresy logiczne pamięci operacyjnej na jej fizyczne adresy i zapewniający ochronę dostępu procesów do pamięci.

Wirtualizacja urządzeń We/Wy jest realizowana poprzez mechanizm IOMMU (Input/Output Memory Management Unit). Działa on na analogicznej zasadzie co MMU, zapewniając kontrolę oraz tłumaczenie adresów logicznych urządzeń do ich adresów fizycznych.[1] Pozwala to na przekierowanie urządzenia bezpośrednio pod kontrolę systemu operacyjnego uruchomionego w maszynie wirtualnej, pomijając sterownik systemu zarządcy. Technologia ta może zostać wykorzystana do przekazania w taki sposób urządzeń wymagających bardzo niskich opóźnień i dużej przepustowości, na przykład kart graficznych, lub kart sieciowych. Wtedy pełną kontrolę nad sprzętem otrzymuje system operacyjny gościa w maszynie wirtualnej, a narzut wirtualizacji na wydajność jest zminimalizowany.

DHCP Protokół DHCP (Dynamic Host Configuration Protocol) odpowiada za automatyczne przypisywanie adresów IP oraz wysyłanie konfiguracji sieciowej do urządzeń. Jego założeniem była także iteroperatywność z już wtedy istniejącym protokołem BOOTP (BOOTstrap Protocol), przez co pojedyncze paczki danych (nazwane wiadomościami) są te same dla obu protokołów (z kilkoma odstępstwami).[8] Format wiadomości DHCP jest następujący:[8]

- OP (1 bajt): typ wiadomości
- HTYPE (1 bajt): typ adresu karty sieciowej
- HLEN (1 bajt): długość adresu karty sieciowej
- HOPS (1 bajt): ustawiane przez klienta na 0 (zero); używane przez agentów pośredniczących
- XID (4 bajty): numer identyfikacyjny (ID) tranzakcji; używany do rozpoznania wiadomości przeznaczonych dla konkretnego klienta

- SECS (2 bajty): liczba sekund od rozpoczęcia procesu odnawiania lub nabywania adresu; ustawiane przez klienta
- FLAGS (2 bajty): flagi
- CIADDR (4 bajty): adres IP klienta; uzupełniany tylko, jeżeli klient może odpowiedzieć na zapytanie ARP (Address Resolution Protocol) i jest w stanie BOUND, RENEW, lub REBINDING
- YIADDR (4 bajty): „twój" (klienta) adres IP
- SIADDR (4 bajty): adres następnego serwera do użycia podczas pobierania
- GIADDR (4 bajty): adres agenta pośredniczącego
- CHADDR (16 bajtów): adres fizyczny klienta
- SNAME (64 bajty): (opcjonalnie) nazwa serwera, łańcuch znaków zakończony zerem (0)
- FILE (28 bajtów): nazwa pliku uruchomieniowego, łańcuch znaków zakończony zerem (0)
- OPTIONS (zmienna długość): opcjonalne parametry

Zdefiniowano następujące wiadomości w protokole DHCP:[8]

- DHCPDISCOVER - Klient wysyła pakiet rozgłoszeniowy, aby zlokalizować dostępne serwery
- DHCPOFFER - Serwer odpowiada klientowi na zapytanie DHCPDISCOVER z proponowaną konfiguracją
- DHCPREQUEST - Klient odpowiada serwerowi (a) żądając oferowanych parametrów od jednego serwera i ignorując pozostałe, (b) potwierdzając prawidłowość otrzymanej wcześniej konfiguracji, lub (c) przedłużając dzierżawę adresu IP
- DHCPACK - Serwer odpowiada klientowi parametrami konfiguracji, wraz z przyznanym adresem IP
- DHCPNAK - Serwer odpowiada klientowi zaznaczając, że konfiguracja żądana przez klienta jest nieprawidłowa, lub dzierżawa adresu wygasła
- DHCPDECLINE - Klient odpowiada serwerowi, że dany adres sieciowy jest już w użytku

- DHCPRELEASE - Klient odpowiada serwerowi, zwalniając dany adres sieciowy i anulując bieżącą dzierżawę
- DHCPINFORM - Klient wysyła zapytanie serwerowi, prosząc o konfigurację lokalną; adres sieciowy skonfigurowany zewnętrznie

Proces konfiguracji w takiej sieci jest następujący[8]:

- klient wysyła zapytanie `DHCPDISCOVER`
- serwer wysyła klientowi proponowaną konfigurację poprzez wiadomość `DHCPOFFER`
- klient żąda zadanej konfiguracji poprzez wiadomość `DHCPREQUEST`
- serwer odpowiada klientowi wiadomością `DHCPACK`, jeżeli żądana konfiguracja jest poprawna, w przeciwnym przypadku odpowiada `DCHPNAK`
- przy odłączaniu od sieci, klient wysyła wiadomość `DHCPRELEASE`, zwalniając zajmowany adres sieciowy

TFTP TFTP (Trivial File Transfer Protocol) to nieskomplikowany protokół przesyłu plików. Jego jedynym zadaniem jest odczyt pliku po stronie serwera i przesłanie go do klienta. Nie obsługuje listowania plików, ani uwierzytelniania.[22]

Protokół obsługuje następujące tryby przesyłu danych[22]:

- `NETASCII`, czyli zmodyfikowana forma formatu `ASCII`, rozszerzona do długości 8 bitów i zawierająca dodatkowe znaki kontrolne
- `OCTET`, czyli format przesyłu surowych bajtów danych
- `MAIL`, uznany za przestarzały w specyfikacji RFC1350[22]

Proces transferu plików jest następujący[22]:

- klient wysyła zapytanie RRQ (read request/żądanie odczytu), lub WRQ (write request/żądanie zapisu), zawierające nazwę pliku i tryb transferu do serwera na port 69
- serwer odpowiada pakietem OPTIONS ACK (jeżeli były użyte) i pakietem ACK w przypadku zapytania WRQ; w przypadku zapytania RRQ odpowiedź to od razu pierwszy pakiet żądanych danych
- maszyna będąca źródłem pliku wysyła go w ponumerowanych segmentach (domyślnie o wielkości 512 bajtów); maszyna odbierająca na każdy segment odpowiada pakietem ACK

- ostatni segment jest rozpoznawany po wielkości, która jest mniejsza od poprzednich (domyslnie musi mieścić się w przedziale od 0 bajtów do 511 bajtów)

- jeżeli odpowiedź ACK dla danego segmentu nigdy nie została otrzymana przez maszynę wysyłającą, po upływie określonego czasu (timeout) segment jest retransmitowany

PXE PXE (Preboot eXecution Environment) to specyfikacja ustandaryzowanego środowiska pozwalającego na uruchamianie oprogramowania poprzez sieć. Rozwiązanie takie pracuje w architekturze klient-serwer, gdzie od klienta wymagane jest jedynie posiadanie odpowiedniej karty sieciowej (wspierającej to rozwiązanie) oraz wsparcie w oprogramowaniu inicjalizującym maszyny, odpowiedzialnym za uruchamianie oprogramowania (na przykład BIOS/UEFI dla maszyn o architekturze X86). Część serwera jest zbiorem już wcześniej istniejących rozwiązań, składającym się z serwera DHCP oraz serwera TFTP. [23]

Proces uruchamiania składa się z następujących kroków[23]:

- wysłanie zapytania `DHCPDISCOVER` w trybie BROADCAST przez klienta

- serwer DHCP odpowiada poprzez wiadomość `DHCPOFFER`; jeżeli serwer DHCP nie implementuje obsługi PXE, wysyła poprawną wiadomość nie zawierającą informacji o lokalizacji plików potrzebnych do uruchomienia minimalnego systemu, a maszyna klienta nie uruchamia się poprzez PXE. W przeciwnym przypadku, serwer DHCP poza zwykłymi informacjami przesyłanymi we wiadomości `DHCPOFFER` dodaje także informacje o lokalizacji serwera TFTP, gdzie znajdują się pliki potrzebne do uruchomienia minimalnego środowiska uruchomienowego, oraz o nazwach tych plików

- klient PXE konfiguruje się danymi przesłanymi przez serwer DHCP oraz pobiera pliki uruchomieniowe do pamięci RAM (Random Access Memory); uruchamia otrzymane pliki

- minimalne środowisko uruchomieniowe powinno pobrać pełen obraz instalacyjny lub obraz systemu do uruchomienia i obsłużyć jego urchomienie/instalację; ten krok najczęściej jest wykonywany poprzez bardziej skomplikowane i niezawodne protokoły, takie jak HTTP, lub NFS

Jako minimalne środowisko uruchomieniowe wykorzystuje się często zestaw jądra Linux oraz prostego obrazu initrd, które mają za zadanie pobrać i uruchomić instalator pełnego systemu operacyjnego.

WakeOnLAN (WOL) WakeOnLAN (w skrócie WOL) to technologia pozwalająca na zdalne uruchomienie urządzenia przez przesłanie specjalnego pakietu w sieci LAN (Local Area Network) poprzez adres rozgłoszeniowy. Pakiet ten składa się z 6 bajtów wypełnionych jedynkami (w zapisie szesnastkowym: FF FF FF FF FF FF) oraz powtórzonego 16 razy adresu MAC urządzenia, które powinno zostać wzbudzone. Aby maszyna potrafiła zinterpretować taki pakiet, wymagane jest wsparcie dla tej technologii w oprogramowaniu karty sieciowej oraz płyty głównej urządzenia.[2]

Rozwiązania zarządzające flotą urządzeń klienckich Istnieje wiele rozwiązań mających na celu automatyzację instalacji i zarządzanie flotą komputerów w pracowniach informatycznych/biurach/itp. Zapewniają one narzędzia do skonfigurowania automatycznej instalacji systemu i zdalnego nim zarządzania (między innymi dodawania, usuwania użytkowników, czy instalacji oprogramowania).

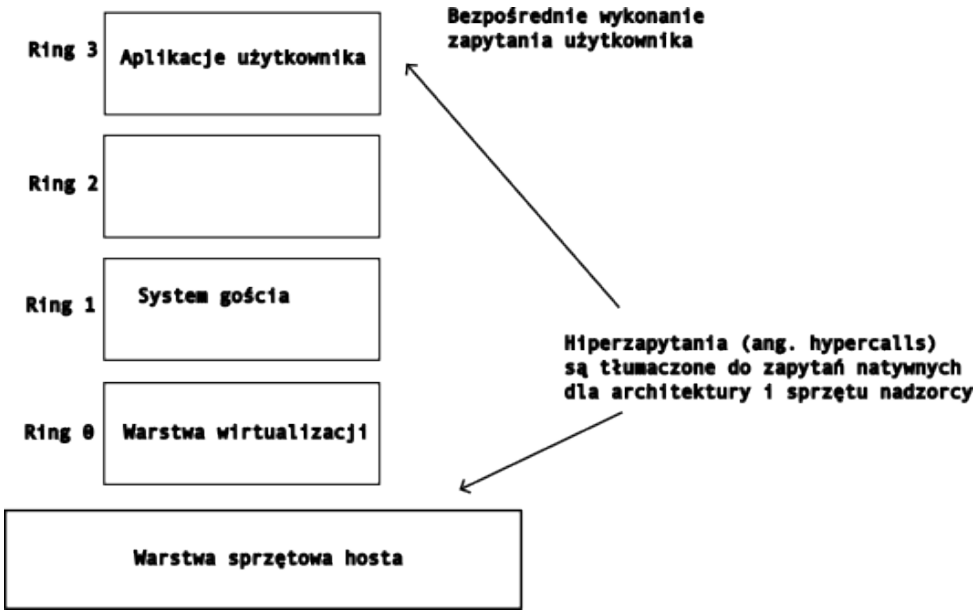
Microsoft Windows Dla systemów z rodziny Microsoft Windows zbiorem narzędzi do zarządzania instalacją systemu jest Windows Assessment and Deployment Kit (ADK). Zawiera on pełny zestaw aplikacji służący do prekonfiguracji instalacji, czy stworzenia minimalnego środowiska uruchomieniowego dla PXE. Zawarte w nim narzędzie WISM (Windows System Image Manager) pozwala utworzyć pliki odpowiedzi dla instalatora systemu, automatyzujące proces instalacji. Z kolei narzędzie DISM (Deployment Image Servicing and Management) daje możliwość modyfikacji elementów obrazu systemu, na przykład pozwalając na wbudowanie zewnetrznych aplikacji bezpośrednio w obraz instalatora.[17]

Po zainstalowaniu, Microsoft zaleca dla takiego przypadku użycia przyłączenie maszyny do usługi Active Directory (AD). Jest to zestaw narzędzi wbudowanych w system Microsoft Windows Server pozwalający na zdalne, scentralizowane zarządzanie wieloma klientami opartymi o system Microsoft Windows.[16]

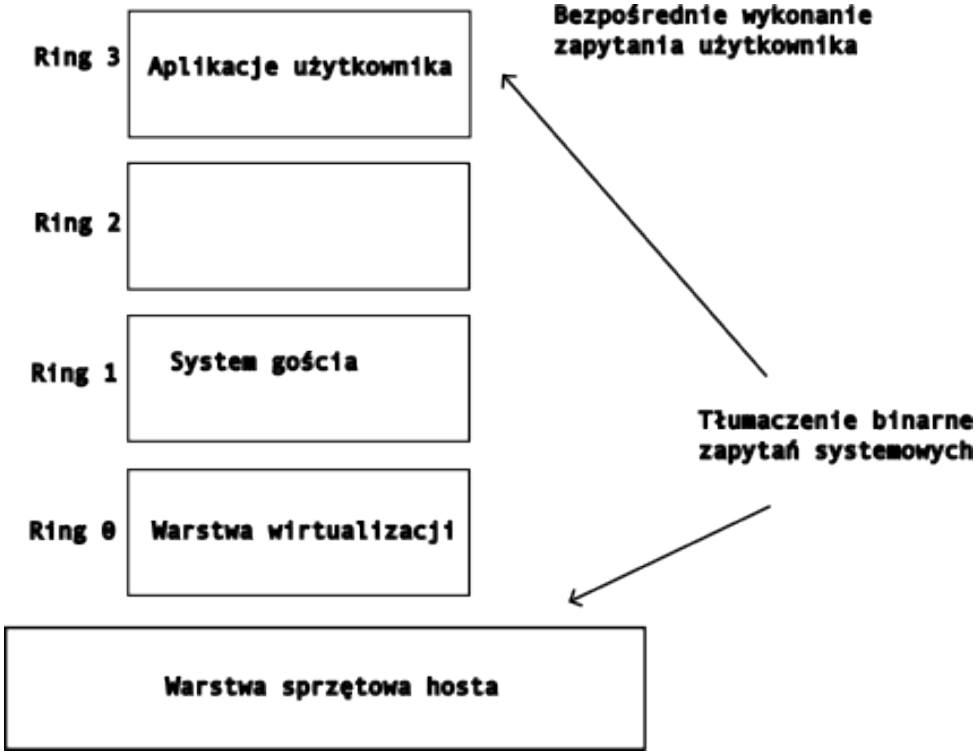
Linux Z kolei dla systemów z rodziny Linux podobny efekt można uzyskać poprzez kombinację kilku narzędzi. Do automatyzacji procesu instalacji powstało narzędzie cloud-init, zaprezentowane przez firmę Canonical. Pozwala ono na stworzenie „przepisu" na podstawie którego instalator systemu między innymi konfiguruje parametry, instaluje pakiety, tworzy użytkowników, czy importuje klucze SSH. W połączeniu z serwerem PXE możliwe jest w ten sposób instalowanie tak samo skonfigurowanego systemu na wielu maszynach, w sposób w pełni zautomatyzowany.[4]

Z kolei do zarządzania flotą urządzeń można wykorzystać narzędzia takie jak Ansible, Chef, czy Puppet. Umożliwiają wykonywanie skryptów na wielu maszynach w sposób zautomatyzowany. Pozwalają one na zarządzanie użytkownikami, aplikacjami, a nawet potrafią synchronizować pliki. Chef oraz Puppet używają procesu agenta, który odpytuje

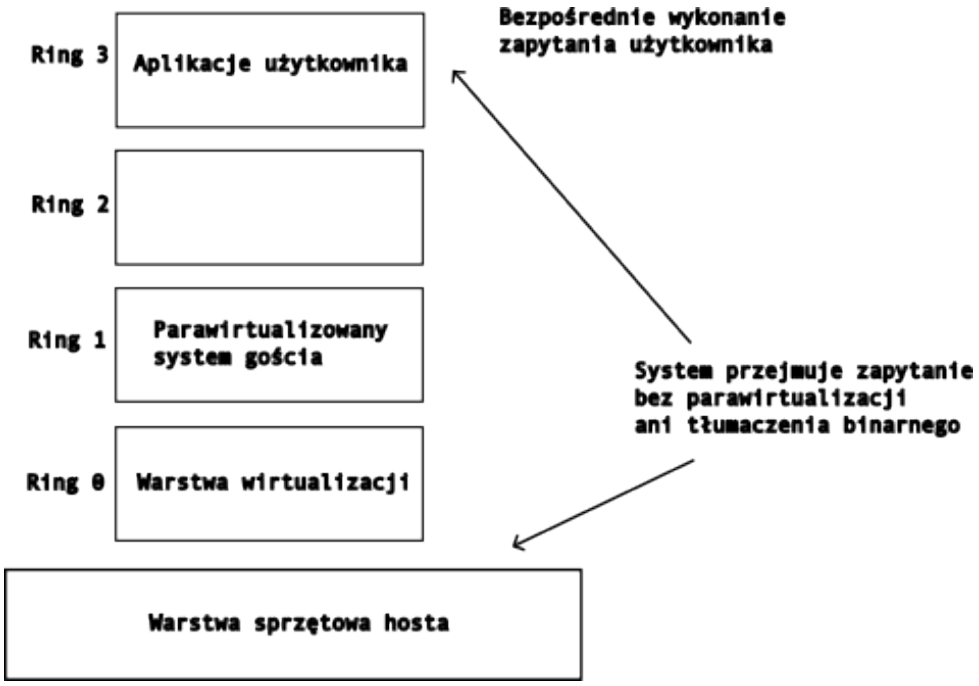
serwer konfiguracji i aplikuje wykryte zmiany. Ansible natomiast wykorzystuje podejście, w którym to maszyna nadzorcy wysyła pierwsza konfigurację na maszyny zarządzane. Nie wymaga także instalacji dodatkowego oprogramowania na maszynie zarządzanej, wystarczy jedynie uruchomiony serwer SSH oraz zainstalowany interpreter języka Python.



Rysunek 2.1: Diagram przedstawiający schemat działania systemu z parawirtualizacją (za [3], rys. 1.6)



Rysunek 2.2: Diagram przedstawiający schemat działania systemu z pełną wirtualizacją (za [3], rys 1.5)



Rysunek 2.3: Diagram przedstawiający schemat działania systemu ze sprzętowym wsparciem wirtualizacji (za [3], rys. 1.7)

Rozdział 3

Wymagania i narzędzia

Wymagania funkcjonalne Rozwiązanie powinno spełniać następujące wymagania funkcjonalne:

- pełna automatyzacja procesu instalowania i konfiguracji systemu operacyjnego na urządzeniach
- użycie maszyn wirtualnych
- pozorna „bezstanowość": po zakończeniu pracy w maszynie wirtualnej i jej wyłączeniu, kolejne jej uruchomienie przywróci pierwotny obraz dysku maszyny, bez zmian wprowadzonych przez poprzedniego użytkownika
- proste tworzenie obrazów maszyn wirtualnych o zmodyfikowanej konfiguracji
- łatwość użycia: użytkownicy końcowi powinni być w stanie w łatwy sposób obsługiwać maszyny klienckie

Wymagania niefunkcjonalne Rozwiązanie powinno spełniać następujące wymagania niefunkcjonalne:

- modularność: możliwość łatwego dodania nowych funkcjonalności
- użycie rozwiązań, które są Wolnym Oprogramowaniem
- możliwość wbudowania rozwiązania w już istniejącą architekturę sieciową

Aby osiągnąć pełną automatyzację procesu instalacji i konfiguracji systemu operacyjnego na urządzeniach wykorzystano serwer PXE, stworzony z serwera TFTP (tftp-hpa), serwera DHCP (dhcp-isc-server) oraz serwera HTTP (Apache2), a także narzędzie cloud-init do automatycznego przeprowadzenia procesu instalacji systemu operacyjnego nadzorcy oraz narzędzie Ansible, do pełnej konfiguracji systemu operacyjnego zarządcy.

Całość rozwiązania instalacyjno-konfiguracyjnego zaimplementowano pod kontrolą systemu operacyjnego Ubuntu Linux w wersji 22.04 LTS na urządzeniu Dell Wyse 5060. Jest to tak zwany „cienki klient", czyli zminiaturyzowana wersja komputera osobistego o ograniczonej mocy obliczeniowej, której celem było uruchamianie klienta zdalnego pulpitu i umożliwienie w ten sposób pracy na dużo wydajniejszej maszynie. Maszyny te są obecnie często wykorzystywane przez pasjonatów jako energooszczędne alternatywy dla komputerów jednopłytkowych (takich jak Raspberry Pi) i służą jako prywatne serwery domowe. Wykorzystany terminal posiada czterordzeniowy procesor AMD GX-424CC, 8 GB pamięci RAM, dysk SATA SSD 256GB oraz gigabitową kartę sieciową opartą o chip Realtek RTL8111/8168/8411.

Do obsługi wirtualizacji wykorzystano narzędzie QEMU. Jest to zintegrowany system do emulacji procesorów poprzez tłumaczenie binarne, potrafi także korzystać z technologii KVM do uruchamiania aplikacji/maszyn wirtualnych z minimalnym narzutem wydajnościowym.[21] W proponowanym rozwiązaniu użyto trybu, gdzie QEMU korzysta z KVM, aby zapewnić jak najlepszą wydajność.

Rozdział 4

Specyfikacja zewnętrzna

Wymagania sprzętowe - strona kliencka Maszyny klienckie, na których uruchomiona ma być aplikacja kliencka i maszyny wirtualne, powinny spełniać następujące wymagania:

- procesor 64-bit, wspierający Intel VT-d, lub AMD-V
- program uruchomieniowy UEFI
- co najmniej 4GB pamięci RAM, preferowane 8GB
- dysk o pojemności co najmniej 250GB
- płyta główna wspierająca Wake On LAN oraz uruchamianie poprzez PXE
- karta graficzna o wydajności zbliżonej do Intel HD Graphics 620, na przykład APU z serii AMD Ryzen, Nvidia GT 1030, lub nawet Nvidia GT 730
- karta sieciowa wspierająca Wake On LAN i prędkości 1 gigabit

Wymagania sprzętowe - strona sieciowa Sieć, w której ma operować projekt, powinna spełniać poniższe warunki:

- warstwa łącza zbudowana w oparciu o sieć Ethernet w klasie co najmniej 5E (osiągającą prędkości 1 gigabita)
- urządzenia sieciowe pozwalające na transmisję „magic packet”
- preferencyjnie dedykowana podsieć, wyłącznie dla zarządzanych maszyn klienckich

Wymagania sprzętowe - strona serwera Serwer, na którym uruchomione będą aplikacje serwerów, powinien spełniać następujące wymagania:

- co najmniej gigabitowa karta sieciowa
- procesor czterordzeniowy
- 8GB pamięci RAM, 8GB pamięci SWAP
- dysk o pojemności co najmniej 250GB

Sposób instalacji Procedura instalacji serwera zarządcy obrazów i maszyn jest następująca:

- system operacyjny, na którym uruchamiany jest serwer powinien posiadać zainstalowane następujące pakiety:
  - serwer HTTP Apache2 (apache2 dla Ubuntu)
  - implementację ISC serwera DHCP (dhcp-isc-server dla Ubuntu)
  - serwer TFTP (tftp-hpa dla Ubuntu)
  - interpreter Python w wersji co najmniej 3.9
- po instalacji wymaganych pakietów należy utworzyć wymagane katalogi:
  - `/ks` - katalog zawierający pliki konfiguracyjne dla narzędzia cloud-init
  - `/images` - katalog zawierający obraz systemu operacyjnego zarządcy dla maszyn klienckich (Ubuntu Server 22.04 LTS)
  - `/srv/tftp` - katalog zawierający pliki do uruchomienia instalatora poprzez PXE
- skopiować pliki konfiguracyjne z repozytorium poniżej do odpowiednich katalogów

  `https://git.nixenos.ovh/engineering-degree/configuration-files.git`
- przygotować listę adresów MAC zarządzanych urządzeń
- skonfigurować serwer DHCP tak, aby przypisywał stałe adresy IP urządzeniom klienckim, spisać te adresy
- do katalogu należy pobrać kod źródłowy aplikacji serwera:

  `git clone https://git.nixenos.ovh/engineering-degree/orchestrator-app.git`
- następnie należy przejść do katalogu zawirającego aplikację `cd orchestrator-app`
- utworzyć środowisko wirtualne Python i uruchomić je:

  - `python3 -m venv .env`
  - `source .env/bin/activate`
- skonfigurować opcje serwera w pliku konfiguracyjnym `config.yml`
- zresetować serwer dhcp, tftp oraz http
- uruchomić serwer obrazów komendą `python fleetcontrol run`
- pobrać zestaw skryptów Ansible

  `git clone https://git.nixenos.ovh/engineering-degree/ansible-scripts.git`
- uzupełnić plik inventory/nodes.yml adresami IP przypisanymi do urządzeń klienckich w konfiguracji serwera DHCP

Proces instalacji i konfiguracji klientów jest następujący:

- uruchomić wspracie dla PXE oraz Wake On LAN w oprogramowaniu BIOS
- ustawić PXE jako pierwsze urządzenie uruchomieniowe, jako drugie ustawić lokalny dysk twardy
- wysłać do każdego urządzenia „magic packet”:

  `wakeonlan -i adres-broadcast-sieci adres-mac-urządzenia`
- uruchomić instalator klikając enter w oknie wyboru przedstawionym na rys. 4.1
- po restarcie wszystkich urządzeń, podmienić adres IP serwera w pliku config.yml i uruchomić skrypty ansible z katalogu `ansible-scripts` na serwerze:

  `ansible-playbook provision-new-node.yml -i inventory/nodes.yml -K`

Obsługa - strona serwera Po wykonaniu wstępnej konfiguracji, należy zbudować obraz maszyny wirtualnej i zarejestrować go w systemie ich obsługi. Aby zbudować obraz maszyny wirtualnej, wystarczy stworzyć obraz dysku w formacie qcow2 (np. `qemu-img create -f qcow2 ubuntu.qcow2 20G`), a następnie uruchomić maszynę wirtualną z tym obrazem dysku i medium instalacyjnym, zainstalować oraz skonfigurować system operacyjny, a następnie skopiować utworzony obraz dysku na serwer.

Będąc zalogowanym do powłoki na serwerze, należy umieścić obraz dysku w miejscu, gdzie użytkownik uruchamiający serwer ma dostęp do odczytu i zapisu plików. Następnie, należy zarejestrować obraz w bazie danych serwera komendą `python fleetcontrol`

`add_image`, przykład użycia przedstawiono na rys. 4.2. Po dodaniu obrazu można zweryfikować, że obraz został prawidłowo zarejestrowany poprzez komendę `python fleetcontrol print_images`. Dla skonfigurowanego serwera dała ona rezultat przedstawiony na rys. 4.3.

Po pierwszym uruchomieniu wszystkie maszyny klienckie powinny same się zarejestrować na serwerze, co można sprawdzić komendą `python fleetcontrol print_clients`. Przykładowy rezultat takiej komendy predstawiono na rys. 4.4.

Po pomyślnej weryfikacji, można przypisać obrazy do klientów. Odbywa sie to poprzez komendę `python fleetcontrol assign_image` zaprezentowaną na rys. 4.5.

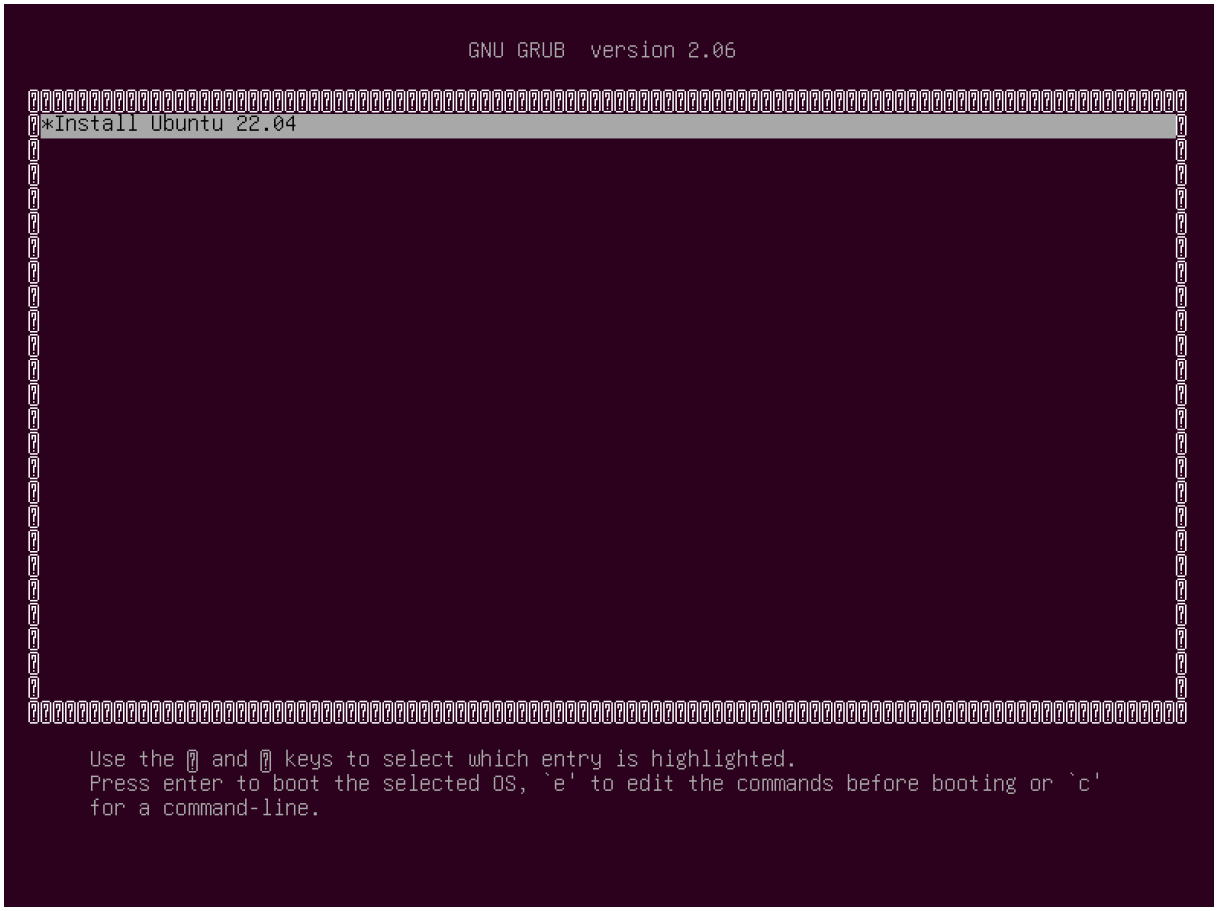
Konfiguracja serwera odbywa się poprzez edycję pliku `config.yml`, lub poprzez ustawienie następujących zmiennych środowiskowych przed uruchomieniem serwera:

- `VALHALLA_SERVER_NAME` - nazwa serwera
- `VALHALLA_SERVER_PORT` - port na którym ma nasłuchiwać serwer
- `VALHALLA_SERVER_HOST` - adres IP hosta, na którym działa serwer (adres karty sieciowej wpiętej do sieci, w której są maszyny klienckie)
- `VALHALLA_SERVER_PASSWORD` - hasło dostępu do serwera dla klientów
- `VALHALLA_SERVER_ACCESS_USERNAME` - nazwa użytkownika dostępowego dla klientów
- `VALHALLA_JWT_SECRET` - sekretna fraza wykorzystywana do obliczenia tokenu JWT
- `VALHALLA_DATABASE_FILE` - inny plik bazy danych SQLite3
- `VALHALLA_LOGLEVEL` - poziom logowania dla aplikacji serwera

Każde z tych ustawień można ustawić także poprzez plik config.yml, jednak to zmienne środowiskowe mają pierwszeństwo i są brane pod uwagę preferencyjnie. Po zmianie ustawień serwera należy pamiętać o aktualizacji ustawień klienta. Odbywa się to poprzez edycję pliku `config.yml` w katalogu `ansible_scripts`.

Obsługa - strona klienta Maszyna kliencka po uruchomieniu przedstawia ekran wyboru obrazu maszyny wirtualnej do uruchomienia, zaprezentowany na rys. 4.6. Użytkownik wybiera interesujący go obraz korzystając ze strzałek, następnie klawiszem tabulatora wybiera opcję OK i naciska klawisz Enter. Wtedy cały ekran przejmowany jest przez maszynę wirtualną, która uruchamia się. Po skończonej pracy w maszynie wirtualnej i jej wyłączeniu, znów pojawia się ekran wyboru obrazu maszyny.

Bezpieczeństwo - zarządzanie obrazami Każde zapytanie do serwera zarządcy obrazów musi być zautoryzowane. Odbywa sie to poprzez token JWT (JSON Web Token), który klient wysyła z każdym zapytaniem do serwera. Proces instalacji systemu bazowego i konfiguracji nie jest jednak chroniony, jeżeli w sieci znajdują się maszyny, które uruchomią się poprzez PXE, mogą zainstalować podstawową konfigurację systemu. Nie posiada ona jednak aplikacji klienckiej, ani żadnej konfiguracji, jedynie klucz publiczny SSH, wymagany do wykonania skryptów Ansible. Jednak o ile złośliwy aktor nie podszyje się pod adres IP dowolnej maszyny do skonfigurowania, nie zostana na takiej maszynie wykonane skrypty konfigurujące aplikację kliencką oraz nie zostaną przekazane pliki konfiguracyjne.



Rysunek 4.1: Ekran wyboru systemu do zainstalowania

```
python fleetcontrol add_image --image-name ubuntu22 --image-filepath ubuntu.qcow2 --image-version v1.0.0
```

Rysunek 4.2: Przykład rejestrowania obrazu maszyny wirtualnej

```
nixen …/orchestrator-app  master !?  ♥ 19:26 python fleetcontrol print_images
+----+----------+---------+---------------+----------------------------------+
| Id |   Name   | Version | File location |               Hash               |
+----+----------+---------+---------------+----------------------------------+
| 1  | ubuntu22 |  v1.0.0 |  ubuntu.qcow2 | dbdc4d7f10511387ef89ffc503080b92 |
+----+----------+---------+---------------+----------------------------------+
```

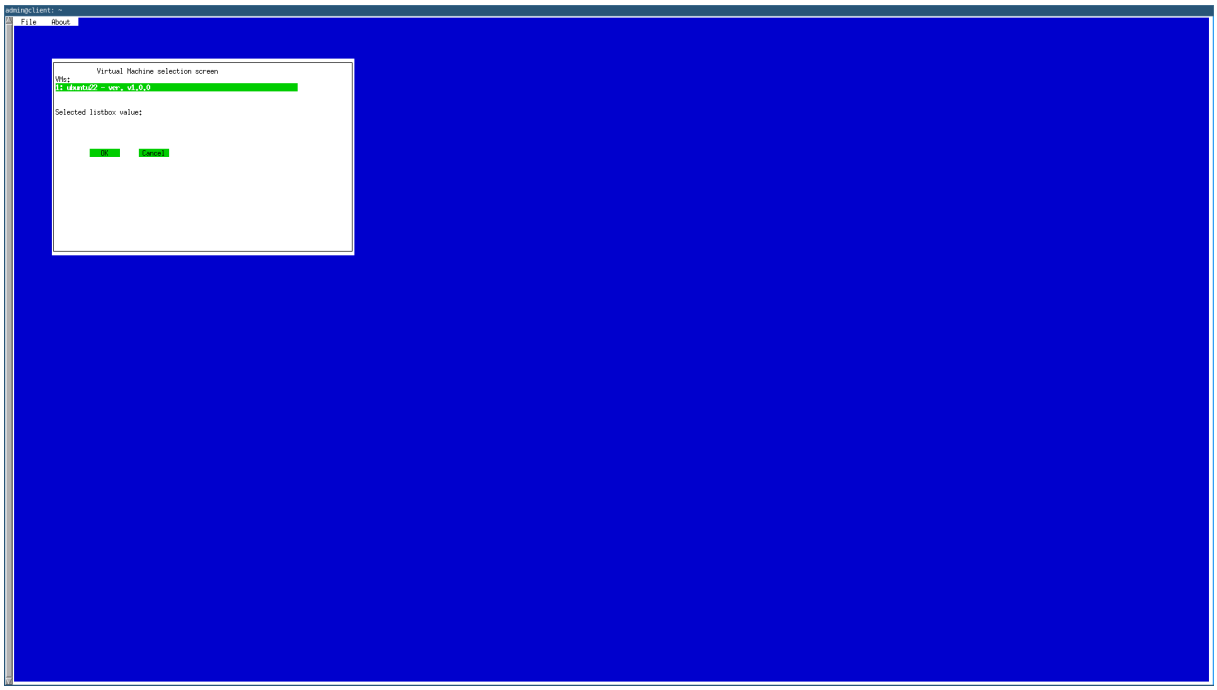
Rysunek 4.3: Przykładowy wynik komendy wyświetlającej wszystkie zarejestrowane obrazy

```
nixen …/orchestrator-app  master !?  ♥ 19:27 python fleetcontrol print_clients
+-------------------+---------------+----------+-------------+
|    MAC address    |   IP address  | Hostname |   Version   |
+-------------------+---------------+----------+-------------+
| 8c:16:45:c8:35:1b | 192.168.1.125 |  client  | v0.0.1alpha |
+-------------------+---------------+----------+-------------+
```

Rysunek 4.4: Przykładowy wynik komendy wyświetlającej wszystkich zarejestrowanych klientów

```
python fleetcontrol assign_image --mac-address 8c:16:45:c8:35:1b --image-name ubuntu22 --image-version v1.0.0
```

Rysunek 4.5: Przykładowe wywołanie komendy przypisującej obraz do klienta



Rysunek 4.6: Ekran wyboru maszyny wirtualnej do uruchomienia

Rozdział 5

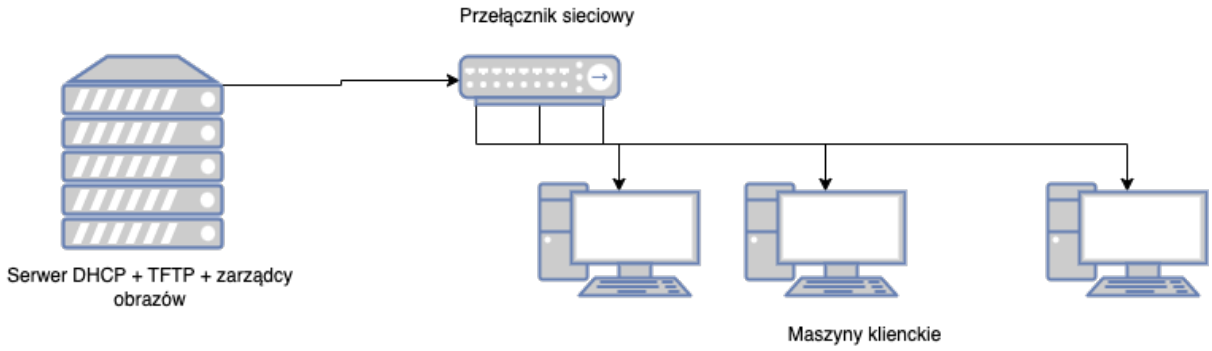
Specyfikacja wewnętrzna

Idea rozwiązania Zamysł proponowanego rozwiązania jest dość prosty i opiera się na kilku elementach:

- serwer przechowuje metadane o obrazach (nazwa, wersja, hash pliku, lokalizacja pliku), same pliki obrazów oraz dane o klientach (nazwa hosta, adres IP, adres MAC, wersja aplikacji klienckiej)
- dane o obrazach jak i same obrazy są ręcznie dodawane przez administratora
- klienci sami rejestrują się oraz aktualizują dane o sobie na serwerze
- klienci pobierają dane o przypisanych do siebie maszynach wirtualnych, pobierają także przypisane obrazy
- komunikacja odbywa się poprzez protokół HTTP
- do komunikacji z serwerem wymagana jest autoryzacja poprzez token JWT(JSON Web Token), uzyskiwany poprzez wysłanie loginu oraz hasła w formacie JSON do odpowiedniego punktu końcowego na serwerze
- po synchronizacji stanu pomiędzy klientem a serwerem, na kliencie uruchamia się okno wyboru obrazu do uruchomienia
- aplikacja klienta zbiera także podstawowe dane o maszynie: (zainstalowaną pojemność pamięci RAM, liczbę wątków procesora, adres IP oraz MAC karty sieciowej, która ma połączenie z siecią)
- po wybraniu obrazu, na podstawie danych o maszynie zebranych przez aplikację klienta generowany jest skrypt, uruchamiający program QEMU z zebranymi danymi jako argumentami wywołania
- oryginalny obraz maszyny wirtualnej jest kopiowany, a następnie uruchamiany jest skrypt włączający maszynę wirtualną

- po zakończonej pracy i wyłączeniu maszyny wirtualnej, skrypt podmienia modyfikowany obraz dysku maszyny wirtualnej na oryginalny, skopiowany w kroku wcześniej
- środowisko klienta składa się z bardzo okrojonego środowiska graficznego `i3`[1], które nie pozwala na wyjście z aplikacji klienckiej oraz emulatora terminala rxvt
- po uruchomieniu maszyny wirtualnej środowisko graficzne promuje jej okno do wypełnienia całego ekranu, jednocześnie przechwytując klawiaturę i myszkę, aby przekierować je do obsługi maszyny wirtualnej

Architektura sieciowa rozwiązania Proponowana architektura sieciowa dla rozwiązania została zaprezentowana na rys. 5.1. Zakłada ona, że serwer DHCP będzie na tej samej maszynie co serwer TFTP oraz zarządcy obrazów, jednak możliwe jest rozbicie każdego z tych narzędzi na osobną maszynę dedykowaną. Podobnie, jeżeli zagwarantowana zostanie poprawna konfiguracja, możliwe jest wykorzystanie dowolnego serwera DHCP, jednak przetestowana została wyłącznie architektura z serwerem DHCP w implementacji ISC.



Rysunek 5.1: Proponowana architektura sieciowa dla rozwiązania

Strona serwera Logika serwera zarządcy obrazów opiera się na koncepcie obrazu oraz klienta. Istnieje tylko jeden użytkownik, wspólny dla wszystkich klientów. Każdy klient może mieć przypisane do siebie wiele obrazów. Diagram klas wykorzystywanych przez aplikację przedstawiono na rys. 5.2.

Dane o obrazach i klientach przechowywane są w bazie danych SQLite3. Aplikacja korzysta z biblioteki ORM (Object-Relational Mapping) SQLAlchemy dla języka Python. Narzędzie to stworzyło 4 tabele o następujących kolumnach:

- clients:

[1] `https://i3wm.org/`

  - mac_address: VARCHAR NOT NULL PRIMARY KEY
  - ip_address: VARCHAR(16) NOT NULL
  - hostname: VARCHAR(100) NOT NULL
  - client_version: VARCHAR(100) NOT NULL
- vm_images:
  - image_id: INTEGER NOT NULL PRIMARY KEY
  - image_name: VARCHAR(100) NOT NULL
  - image_file: VARCHAR(500) NOT NULL
  - image_version: VARCHAR(100) NOT NULL
  - image_hash: VARCHAR(500) NOT NULL
  - image_name_version_combo: VARCHAR (600) NOT NULL
- users:
  - user_id: INTEGER NOT NULL PRIMARY KEY
  - username: VARCHAR NOT NULL
  - password_hash: VARCHAR NOT NULL
- client_image:
  - client_mac: FOREIGN KEY VARCHAR
  - mage_id: FOREIGN KEY INTEGER

Do stworzenia serwera wykorzystano następujące biblioteki:

- bcrypt - do tworzenia bezpiecznych „hashy” (wyników funkcji skrótu) haseł
- Flask - do stworzenia serwera HTTP
- pyaml - do odczytu i analizy plików o rozszerzeniu YAML
- PyJWT - do obsługi i tworzenia tokenów JWT
- prettytable - do formatowania danych w przejrzysty sposób
- SQLAlchemy - jako ORM do obsługi bazy danych
- argparse - do parsowania argumentów wywołania programu

Program został podzielony na kilka głównych modułów:

- obsługa bazy danych
- serwer HTTP, obsługa żądań i odpowiedzi
- obsługa pliku konfiguracyjnego i zmiennych środowiskowych
- moduł modeli danych dla bazy danych
- moduł obsługi autoryzacji poprzez tokeny JWT

Obsługa bazy danych została zaimplementowana w klasie `Database`. Jej zadaniem jest zestawienie połączenia z bazą danych oraz manipulacja wszystkimi modelami wykorzystywanymi w programie. Dodawanie, usuwanie, modyfikacja i pobieranie danych z bazy danych odbywa się poprzez zaimplementowane metody. Są one wysokopoziomowymi abstrakcjami podstawowych zapytań SQL:

- metody o formacie `get_[nazwa_modelu]_by_[nazwa_atrybutu]` odpowiadają zapytaniom `SELECT`
- metody o formacie `add_[nazwa_modelu]` odpowiadają zapytaniom `INSERT`
- metody o formacie `modify_[nazwa_modelu]` odpowiadają zapytaniom `UPDATE`
- metody o formacie `delete_[nazwa_modelu]` odpowiadają zapytaniom `DELETE`

Serwer HTPP został zbudowany w oparciu o bibliotekę Flask. Do obsługi zapytań HTTP stworzono klasę `Server`, która zawiera w sobie metody, które obsługują wszystkie punkty końcowe. Autoryzacja dostępu została zaimplementowana przez mechanizm dekoratorów, a dokładniej przez zaimplementowany dekorator `require_auth`. Sprawdza on, czy w nagłówkach zapytania HTTP klient wysłał odpowiedni token JWT i jeżeli nie był poprawny, zwraca odpowiedni komunikat do klienta. Jeżeli jednak był poprawny, przekazuje do funkcji obsługującej zapytanie informacje o zautoryzowanym użytkowniku. Przykład użycia tego dekoratora przedstawiono na listingu 5.3.

Serwer wystawia następujące punkty końcowe (ang. endpoints):

- GET `/` - przedstawia podstawowe dane o serwerze: nazwę, wersję oraz nazwę hosta, przykład odpowiedzi podano w listingu 5.4
- POST `/login` - pozwala na zalogowanie się poprzez wysłanie nazwy użytkownika i hasła, zwraca token JWT służący do autoryzacji, id użytkownika oraz jego nazwę jako potwierdzenie, przykład ciała zapytania logowania pokazano w listingu 5.5, przykłady możliwych odpowiedzi pokazano w listingu 5.6

- POST `/clients` - pozwala na rejestrację klienta, przykład wymaganego ciała zapytania pokazano w listingu 5.7
- PUT `/clients` - pozwala na modyfikację danych o już zarejestrowanym kliencie, zapytania i odpowiedzi są takie same, jak dla żądań rejestracji klientów
- GET `/clients/adres_mac` - pozwala na pobranie danych o zadanym kliencie z serwera, możliwe odpowiedzi pokazano w listingu 5.8
- GET `/clients/adres_mac/vms` - pozwala na pobranie listy numerów id obrazów przypisanych do klienta o podanym adresie MAC, możliwe odpowiedzi ukazano w listingu 5.9
- GET `/images/id_obrazu` - pozwala na pobranie danych o obrazie o zadanym numerze id, możliwe odpowiedzi pokazano w listingu 5.10
- GET `/images/id_obrazu/download` - pozwala na pobranie obrazu o zadanym numerze id

Ostatnim ważnym komponentem aplikacji jest klasa `ServerConfig`. Odpowiada ona za odczyt i parsowanie plików konfiguracyjnych oraz odczytanie wartości ustawionych zmiennych środowiskowych i nadpisanie odpowiednich wartości w konfiguracji. Do obsługi plików konfiguracyjnych wybrano bibliotekę `PYAML`.

Strona klienta Po stronie klienta jedyną strukturą danych jaka została wykorzystana, był model obrazu, wykorzystywany także po stronie serwera. Wykorzystano następujące biblioteki:

- picotui - do stworzenia interfejsu użytkownika
- getmac - do pobrania informacji o karcie sieciowej
- pyaml - do odczytu i parsowania plików konfiguracyjnych
- requests - do wysyłania zapytań do serwera
- psutil - do pobierania informacji o dostępnej liczbie rdzeni i pamięci operacyjnej urządzenia

Aplikacja klienta została podzielona na klika podstawowych modułów:

- moduł sprawdzający dane o maszynie, na której jest uruchomiony
- moduł obsługujący komunikację z serwerem
- moduł obsługujący uruchamianie maszyn wirtualnych oraz wyświetlanie interfejsu użytkownika

Sprawdzenie danych o maszynie odbywa się w kilku krokach: aby pobrać adres IP maszyny, program nawiązuje połączenie z serwerem DNS Google (adres IP: `8.8.8.8`), a następnie sprawdza nazwę gniazda (ang. socket), które połączyło się z siecią. Adres MAC jest sprawdzany przez bibliotekę `get_mac_address`, liczba rdzeni procesora przez bibliotekę `os`, a ilość pamięci RAM poprzez bibliotekę `psutil` (bibiloteka zwraca liczbę bajtów pamięci, a reszta aplikacji wymaga liczby gigabajtów, stąd wymagana była konwersja). Implementacja tych funkcjonalności, wraz z odczytem i parsowaniem plików konfiguracyjnych, znajduje się w klasie `MachineData`

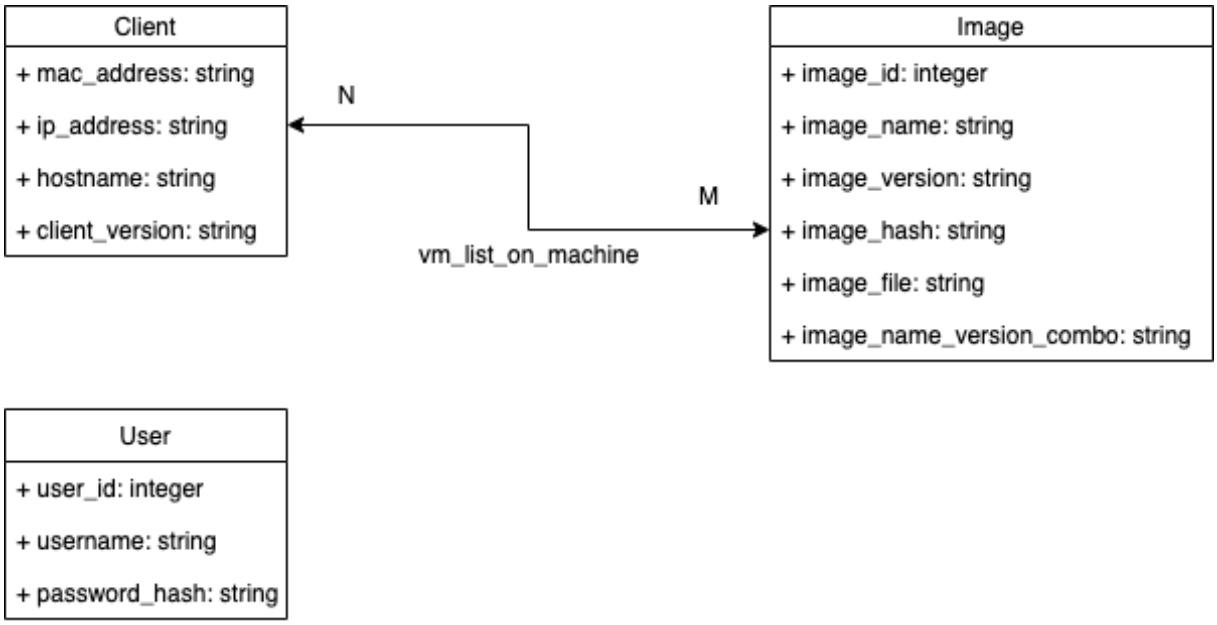
Obsługa komunikacji z serwerem została umieszczona w klasie `ValhallaServer`. Za pomocą biblioteki `requests` wysyłane są zapytania do serwera. Jeżeli wygaśnie token JWT, wysyłane jest zapytanie służące do zalogowania się do systemu, a następnie ponawiane jest poprzednie zapytanie. Dane z serwera są przekazywane w formacie JSON. Zaraz po uruchomieniu aplikacji klienckiej wysyłane jest żądanie rejestracji maszyny, lub, jeżeli maszyna już jest zarejestrowana, następuje aktualizacja danych o maszynie. Następnie pobierana jest lista obrazów przypisanych do konkretnej maszyny, a same obrazy są pobierane. Wszystkie te operacje są obsługiwane przez klasę `ValhallaServer`.

Obsługa uruchamiania maszyn wirtualnych sprowadza się do skopiowania pliku-wzorca skryptu uruchamiającego maszynę wirtualną QEMU do odpowiedniego katalogu, podmianie tymczasowych wartości liczby rdzeni, ilości pamięci RAM i ścieżki do obrazu dysku maszyny na prawidłowe wartości wyznaczone przez program w klasie `MachineData` oraz na podstawie danych obrazów pobranych z serwera, nadaniu odpowiednich uprawnień dla pliku skryptu (bit wykonywalności - execute) oraz na uruchomieniu skryptu i zablokowaniu wykonywania aplikacji. Po wyjściu z maszyny wirtualnej, ekran wyboru znów się uruchamia i znów pozwala na wybór maszyny do uruchomienia.

Pliki konfiguracyjne PXE Najważniejszy fragment pliku konfiguracyjnego serwera DHCP zawarto w listingu 5.11. Ustawia on wskaźnik na serwer TFTP serwujący pliki wymagane do uruchomienia środowiska instalacyjnego.

Z kolei w listingu 5.12 zawarto plik `grub.cfg`, który konfiguruje ekran wyboru instalatora oraz obsługuje uruchomienie minimalistycznego jądra systemu Linux, uruchamiającego pełny instalator systemu.

W listingu 5.13 ukazano konfigurację serwera Apache2, który serwuje obrazy `.iso` instalatorów.



Rysunek 5.2: Diagram klas wykorzystywanych przez serwer

```
class Server:
    @require_auth
    def register_new_client_to_database(request_user, self):
```

Rysunek 5.3: Przykład użycia dekoratora `require_auth`

```
HTTP200:
{
    "server_name": "VALHALLA",
    "server_version": "v0.0.1",
    "host": "127.0.0.1"
}
```

Rysunek 5.4: Przykład możliwej odpowiedzi na zapytanie pod punkt końcowy /

```
        {
            "username": "user",
            "password": "sekret_password"
        }
```

Rysunek 5.5: Przykład prawidłowego ciała żądania logowania

```
HTTP202 (poprawne dane):
{
    "token": "xxxxxxxxx",
    "user_id": "1",
    "username": "user"
}
HTTP401 (niepoprawne dane):
{
    "data": null,
    "error": "Auth error",
    "message": "Invalid login data"
}
```

Rysunek 5.6: Możliwe odpowiedzi na żądanie logowania od serwera

```
Poprawne zapytanie:
{
    "mac_address": "00:00:00:00:00:00:00:62",
    "ip_address": "192.168.1.12",
    "hostname": "test",
    "client_version": "v0.0.1alpha",
    "vm_list_on_machine": []
}
Odpowiedz: HTTP201
{
    "success": true
}
Niepoprawne zapytanie:
{
    "mac_address": "00:00:00:00:00:00:00:62",
    "ip_address": "192.168.1.12",
    "client_version": "v0.0.1alpha",
    "vm_list_on_machine": []
}
Odpowiedz: HTTP400
{
    "data": null,
    "error": "'hostname'",
    "message": "Internal server error"
}
```

Rysunek 5.7: Możliwe zapytania i odpowiedzi serwera na żądanie rejestracji klienta

```
HTTP200 (istniejacy adres MAC):
{
    "client_version": "v0.0.1alpha",
    "hostname": "test",
    "ip_address": "192.168.1.12",
    "mac_address": "00:00:00:00:00:00:00:62"
}
HTTP404 (nieistniejacy adres MAC):
{
    "data": null,
    "error": null,
    "message": "Client not found in database"
}
```

Rysunek 5.8: Możliwe odpowiedzi serwera na zapytanie o dane zadanego klienta

```
HTTP200 (istniejacy adres MAC):
[
    1
]
HTTP404 (nieistniejacy adres MAC):
{
    "data": null,
    "error": null,
    "message": "Client not found in database"
}
```

Rysunek 5.9: Mozliwe odpowiedzi serwera na zapytanie o obrazy przypisane do zadanego hosta

```
HTTP200 (instniejacy obraz):
{
    "image_file": "ubuntu.qcow2",
    "image_hash": "dbdc4d7f10511387ef89ffc503080b92",
    "image_id": "1",
    "image_name": "ubuntu22",
    "image_name_version_combo": "ubuntu22@v1.0.0",
    "image_version": "v1.0.0"
}
HTTP404 (nieistniejacy obraz):
{
    "data": null,
    "error": null,
    "message": "Image not found in database"
}
```

Rysunek 5.10: Możliwe odpowiedzi serwera na zapytanie o dane zadanego obrazu

```
allow bootp;
allow booting;
# PXE/iPXE BOOT CONFIGURATION FOR VALHALLA SERVER
next-server 192.168.1.25;
filename "bootx64.efi";
```

Rysunek 5.11: Fragment konfiguracji serwera DHCP

```
set default="0"
set timeout=-30

if loadfont unicode ; then
  set gfxmode=auto
  set locale_dir=\$prefix/locale
  set lang=en_US
fi
terminal_output gfxterm

set menu_color_normal=white/black
set menu_color_highlight=black/light-gray
if background_color 44,0,30; then
  clear
fi

function gfxmode {
        set gfxpayload="\${1}"
        if [ "\${1}" = "keep" ]; then
                set vt_handoff=vt.handoff=7
        else
                set vt_handoff=
        fi
}

set linux_gfx_mode=keep

export linux_gfx_mode

menuentry 'Install Ubuntu 22.04' {
        gfxmode \$linux_gfx_mode
        linux vmlinuz ip=dhcp url=http://192.168.1.25/images/
           ubuntu22/ubuntu.iso autoinstall ds=nocloud-net\;s=http
           ://192.168.1.25/ks/ cloud-config-url=/dev/null fsck.
           mode=skip
        initrd initrd
}
```

Rysunek 5.12: Plik konfiguracyjny `grub.cfg`

```
<VirtualHost 192.168.1.25:80>
    ServerAdmin root@192.168.1.25
    DocumentRoot /
    ServerName pxe-ks.nixenos.ovh
    ErrorLog \${APACHE_LOG_DIR}/ks-server.example.com-error_log
    CustomLog \${APACHE_LOG_DIR}/ks-server.example.com-access_log
        common
    <Directory /ks>
        Options Indexes MultiViews
        AllowOverride All
        Require all granted
    </Directory>
    <Directory /images>
        Options Indexes MultiViews
        AllowOverride All
        Require all granted
    </Directory>
</VirtualHost>
```

Rysunek 5.13: Fragment konfiguracji serwera Apache2

Rozdział 6

Weryfikacja i walidacja

Sposób testów Testy całości rozwiązania zostały przeprowadzone korzystając z komputera Lenovo Thinkpad T480. Posiada procesor Intel Core i5-8250U (4 rdzenie, 8 wątków), 24GB pamięci RAM oraz dysk SSD o pojemności 256GB. Pracuje pod kontrolą UEFI, wspiera Intel VT-d oraz PXE.

W ramach pierwszej fazy testów urządzenie było wielokrotnie czyszczone oraz konfigurowane ponownie. W ich trakcie przeporwadzono pomiary czasu jaki jest wymagany do pełnej konfiguracji maszyny.

W ramach drugiej fazy testów ręcznie wymuszano proces synchronizacji zmian z serwerem. Do testów przygotowano obraz maszyny wirtualnej pracującej pod kontrolą systemu Ubuntu Linux w wersji 22.04 LTS wraz ze środowiskiem graficznym Gnome. Obraz dysku zajmował 12GB. Podczas tych testów sprawdzano czas potrzebny na pobranie przykladowego obrazu, korzystając z gigabitowej karty sieciowej, dostępnej w maszynie.

Do pomiaru czasu wykorzystano narzędzie `time`, dostępne w większości systemów wywodzących się z rodziny UNIX.

W ramach trzeciej fazy testów, poddano ocenie wydajność i odczucia płynące z używania maszyny wirtualnej.

Pomiary czasu Pomiary czasu, w jakim urządzenie przeprowadzało instalację, powtórzono dziesięciokrotnie. Wyniki zostały przedstawione w tabeli 6.2.

Pomiary czasu pobierania obrazów także wykonano dziesięciokrotnie, a wyniki zamieszczono w tabeli 6.1.

Zauważono, że najdłuższym procesem jest ładowanie obrazu do pamięci serwera, samo jego przesyłanie trwa od 15 do 20% zmierzonego czasu.

Wykryte błędy Podczas testów wykryto kilka błędów. Pierwszym z nich było błędne ustawianie uprawnień pliku skryptu uruchamiającego maszynę wirtualną. Kolejny z nich dotyczył błędu w konfiguracji serwera PXE, który uniemożliwiał uruchomienie instalatora. Po zmianie konfiguracji tak, by podstawowe środowisko uruchomieniowe korzystało

z serwera HTTP do pobierania obrazów `.iso`, uruchamianie instalatora działało prawidłowo.

Wyniki badań użytkowych System uruchamiany poprzez maszynę wirtualną działa sprawnie i możliwe jest korzystanie z niego. Oczywiście wydajność, szczególnie graficzna, pozostawia wiele do życzenia. Problemy związane z niewystarczającą mocą obliczeniową do wirtualizacji karty graficznej były najbardziej widoczne podczas korzystania z przeglądarki internetowej Mozilla Firefox, gdzie przeglądanie stron WWW było możliwe, lecz wydajność przy otwartych 5-6 kartach znacząco malała i system stawał się niezdatny do użycia.

Tabela 6.1: Wyniki pomiaru czasu pobierania obrazu maszyny wirtualnej

Numer pomiaru	**Zmierzony czas**
1	4min 40.885s
2	4min 59.926s
3	5min 1.590s
4	5min 11.377s
5	5min 5.292s
6	4min 59.955s
7	5min 2.235s
8	4min 58.982s
9	5min 3.758s
10	5min 1.214s

Tabela 6.2: Wyniki pomiaru czasu konfiguracji systemu zarządcy klienta

Numer pomiaru	**Zmierzony czas**
1	15min 48.290s
2	13min 36.530s
3	10min 40.210s
4	10min 15.700s
5	10min 46.600s
6	9min 36.510s
7	9min 06.420s
8	9min 14.190s
9	9min 15.530s
10	10min 12.950s

Rozdział 7

Podsumowanie i wnioski

Uzyskano prawie pełną automatyzację procesu instalacji systemu nadzorcy. Administrator musi wykonać jedynie 4 manualne kroki: dodanie adresów MAC obsługiwanych maszyn do konfiguracji serwera DHCP i konfiguracji skryptów Ansible, wywołanie komendy wysyłającej „magic packet" do wyznaczonych urządzeń, potwierdzenie instalacji na każdym z urządzeń oraz wywołanie skryptu Ansible do konfiguracji urządzeń.

Dzięki wykorzystaniu QEMU udało się zapewnić użycie maszyn wirtualnych oraz łatwość tworzenia własnych obrazów dla nich. Osiągnięto także „bezstanowość", maszyny wirtualne po restarcie wracają do swojego pierwotnego stanu sprzed wszsytkich zmian wprowadzonych przez ostatniego użytkownika.

Użycie PXE, cloud-init oraz Ansible jako systemu automatyzującego instalację systemu nadzorcy pozwoliło na zapewnienie modularności i możliwości wbudowania rozwiązania w już istniejącą architekturę sieciową.

Niestety wydajność systemu zarządzania obrazami maszyn nie jest satysfakcjonująca. Największym problemem obecnego rozwiązania jest konieczność załadowania całego wysyłanego pliku do pamięci aplikacji, przez co wydajność jest mocno ograniczona. Dobrym rozwiązaniem tego problemu byłoby zintegrowanie serwera aplikacji z serwerem Apache2, lub Nginx, gdzie to dedykowany serwer HTTP wysyłałby duże pliki, a serwer zarządzający obsługiwałby tylko przechowywanie danych. Ciekawym rozwiązaniem tego problemu byłoby też użycie narzędzia rsync, którego zaawansowane wykrywanie różnic w plikach i podmiana jedynie tych właśnie róznic na pewno znacząco przyspieszyłoby działanie serwera obrazów. Kolejnym problemem jest kwestia bezpieczeństwa. Aby możliwe było pełne zautomatyzowanie procesu zdecydowano się na użycie tylko jednego użytkownika dla wszystkich klientów. Jest to duże ryzyko ze względów bezpieczeństwa, ponieważ potencjalny atakujący może wtedy odczytać adresy MAC wszystkich urządzeń i podszywać się pod nie.

Podobnie jak w przypadku serwera, wydajność systemu w maszynach wirtualnych nie jest satysfakcjonująca. Największym problemem jest wydajność graficzna. Rozwiązaniem tego problemu byłoby wprowadzenie mechanizmu przekazywania kontroli nad kartą graficzną systemowi gościa maszyny wirtualnej. Aby takie podejście było możliwe w przypadku posiadania tylko jednej karty graficznej, wymagane byłoby jej wcześniejsze odpięcie od systemu zarządcy. Istnieje kilka projektów[13], które wykorzystują mechanizm „zaczepów" (ang. hooks) do odłączania urządzeń PCI przy uruchamianiu maszyny wirtualnej i podpinania ich z powrotem do systemu zarządcy po wyłączeniu maszyny wirtualnej. Jednak to rozwiązanie wymaga skomplikowanej konfiguracji na maszynach klienckich, między innymi wybrania odpowiednich urządzeń PCI do przekazania do maszyny wirtualnej, gdzie adresy tych urządzeń będą różne na każdej z maszyn, a także ta konfiguracja byłaby różna dla każdego typu urządzeń. Napisanie programu, który w dynamiczny sposób tworzyłby odpowiednie pliki konfiguracyjne, byłoby na pewno skomplikowane, ale możliwe.

Potencjalnym kierunkiem rozwoju tej aplikacji byłoby przeprojektowanie metody synchronizacji obrazów. Optymalną opcją wydaje się użycie narzędzia rsync, aby zapewnić inkrementalne aktualizacje obrazów, bez potrzeby pobierania całości. Sporym ułatwieniem pracy dla administratora byłoby także przygotowanie graficznego interfejsu użytkownika do zarządzania serwerem. Po stronie klienta dobrym torem rozwoju byłoby opracowanie metody na przekazywanie kontroli nad kartą graficzną systemowi gościa w sposób w pełni zautomatyzowany. Ważnym krokiem byłoby także dopracowanie interfejsu użytkownika.

Powyżej przedstawione propozycje rozwoju są tylko sugestią, ponieważ architektura aplikacji pozwala na łatwą integrację z innymi rozwiązaniami automatyzującymi. Dzięki temu przedstawiony projekt jest silnie rozwojowy, a przez użycie rozwiązań na licencjach Wolnego Oprogramowania możliwe jest wykorzystanie go w placówkach naukowych bardzo niskim kosztem.

Bibliografia

[1] AMD. *AMD64 Technology; AMD64 Architecture Programmer's Manual Volume 2: System Programming, Rev. 3.3*. Architecture Specification. 2022. DOI: 24593.

[2] AMD. *Magic Packet Technology*. Whitepaper. 1995. URL: https://www.amd.com/system/files/TechDocs/20213.pdf.

[3] Diane Barrett i Gregory Kipper. „1 - How Virtualization Happens". W: *Virtualization and Forensics*. Red. Diane Barrett i Gregory Kipper. Boston: Syngress, 2010, s. 3–24. ISBN: 978-1-59749-557-8. DOI: https://doi.org/10.1016/B978-1-59749-557-8.00001-1. URL: https://www.sciencedirect.com/science/article/pii/B9781597495578000011.

[4] Canonical. *cloud-init documentation*. 2022. URL: https://cloudinit.readthedocs.io/en/latest/ (term. wiz. 01. 01. 2023).

[5] Mozilla MDN Contributors. *An overview of HTTP*. ostatnia modyfikacja: Nov 22, 2022. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview (term. wiz. 31. 12. 2022).

[6] Mozilla MDN Contributors. *Evolution of HTTP*. ostatnia modyfikacja: Oct 28, 2022. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/Evolution_of_HTTP#http1.1_%E2%80%93_the_standardized_protocol (term. wiz. 31. 12. 2022).

[7] Mozilla MDN Contributors. *HTTP request methods*. ostatnia modyfikacja: Sep 9, 2022. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods (term. wiz. 31. 12. 2022).

[8] Ralph Droms. *Dynamic Host Configuration Protocol*. RFC 2131. Mar. 1997. DOI: 10.17487/RFC2131. URL: https://www.rfc-editor.org/info/rfc2131.

[9] Roy T. Fielding i in. *Hypertext Transfer Protocol – HTTP/1.1*. RFC 2068. Sty. 1997. DOI: 10.17487/RFC2068. URL: https://www.rfc-editor.org/info/rfc2068.

[10] Joseph Galbraith i Oskari Saarenmaa. *SSH File Transfer Protocol*. Internet-Draft draft-ietf-secsh-filexfer-13. Work in Progress. Internet Engineering Task Force, lip. 2006. 60 s. URL: https://datatracker.ietf.org/doc/draft-ietf-secsh-filexfer/13/.

[11] Steve Newsted (Red Hat). *Keeping Linux files and directories in sync with rsync*. 2021. URL: https://www.redhat.com/sysadmin/sync-rsync (term. wiz. 31. 12. 2022).

[12] Intel. *Intel® Virtualization Technology for Directed I/O; Architecture Specification, rev. 4.0*. Architecture Specification. 2022. DOI: D51397-015.

[13] Joe Knockenhauer. *Single GPU Passthrough on Linux*. 4/05/2021. URL: https://github.com/joeknock90/Single-GPU-Passthrough (term. wiz. 06. 01. 2023).

[14] KVM. *Main page — KVM,* 2016. URL: https://www.linux-kvm.org/index.php?title=Main_Page&oldid=173792 (term. wiz. 31. 12. 2022).

[15] Chris M. Lonvick i Tatu Ylonen. *The Secure Shell (SSH) Protocol Architecture*. Request for Comments 4251. 2006. 30 s. DOI: 10.17487/RFC4251.

[16] Microsoft. *AD DS Getting Started*. 7/29/2021. URL: https://learn.microsoft.com/en-us/windows-server/identity/ad-ds/ad-ds-getting-started?source=recommendations (term. wiz. 01. 01. 2023).

[17] Microsoft. *Kits and tools overview; ADK for Windows 11*. 5/24/2022. URL: https://learn.microsoft.com/en-us/windows-hardware/get-started/kits-and-tools-overview (term. wiz. 01. 01. 2023).

[18] N/A. *How Rsync Works; A Practical Overview*. N/A. URL: https://rsync.samba.org/how-rsync-works.html (term. wiz. 31. 12. 2022).

[19] N/A. *OpenSSH Users*. N/A. URL: https://www.openssh.com/users.html (term. wiz. 31. 12. 2022).

[20] N/A. *SSH File Transfer Protocol (SFTP): Get SFTP client and server*. N/A. URL: https://www.ssh.com/academy/ssh/sftp-ssh-file-transfer-protocol (term. wiz. 31. 12. 2022).

[21] QEMU. *QEMU; A generic and open source machine emulator and virtualizer*. N/A. URL: https://www.qemu.org (term. wiz. 31. 12. 2022).

[22] Dr. Karen R. Sollins. *The TFTP Protocol (Revision 2)*. RFC 1350. Lip. 1992. DOI: 10.17487/RFC1350. URL: https://www.rfc-editor.org/info/rfc1350.

[23] Intel with special contributions from SystemSoft. *Preboot Execution Environment (PXE) Specification, rev. 2.1*. Architecture Specification. 1999.

[24] Andrew Tridgell i Paul Mackerras. „The rsync algorithm". W: (1996).

[25] Andrew Tridgell i in. „Efficient algorithms for sorting and synchronization". W: (1999).

Dodatki

Spis skrótów i symboli

SFTP Protokół Bezpiecznego Przesyłu Plików (ang. *Secure File Transfer Protocol*)

SSH Bezpieczna Powłoka (ang. *Secure SHell*)

HTTP Protokół Przesyłu HiperTekstu (ang. *HyperText Transfer Protocol*)

WWW OgólnoŚwiatowa Sieć (ang. *World Wide Web*)

CERN Europejska Organizacja Badań Jądrowych (fr. *Organisation Européenne pour la Recherche Nucléaire*)

TCP Protokół Sterowania Transmisją (ang. *Transmission Control Protocol*)

KVM Maszyna Wirtualna Jądra (ang. *Kernel Virtual Machine*)

MMU Jednostka Zarządzająca Pamięcią (ang. *Memory Management Unit*)

IOMMU Jednostka Zarządzająca Pamięcią Wejścia Wyjścia (ang. *Input Output Memory Management Unit*)

DHCP Protokół Dynamicznego Konfigurowania Hostów (ang. *Dynamic Host Configuration Protocol*)

BOOTP Protokół Bootstrap (ang. *BOOTstrap Protocol*)

TFTP Trywialny Protokół Przesyłu Plików (ang. *Trivial File Transfer Protocol*)

PXE Przeduruchomieniowe Środowisko Uruchomieniowe (ang. *Preboot Execution Environment*)

WOL Uruchom Poprzez LAN (ang. *Wake On LAN*)

WISM ang. *Windows System Image Manager*

DISM ang. *Deployment Image Servicing and Management*

AD ang. *Active Directory*

JWT ang. *JSON Web Token*

JSON Notacja Obiektów JavaScript (ang. *JavaScript Object Notation*)

ORM Mapowanie obiektowo-relacyjne (ang. *Object-Relational Mapping*)

Lista dodatkowych plików, uzupełniających tekst pracy

W systemie do pracy dołączono dodatkowe pliki zawierające:

- źródła programów oraz pliki konfiguracyjne i skrypty
- film pokazujący działanie opracowanego oprogramowania

Spis rysunków

Spis tabel