

Politechnika Warszawska

W Y D Z I A Ł M A T E M A T Y K I
I N A U K I N F O R M A C Y J N Y C H



Praca dyplomowa inżynierska

na kierunku Informatyka i Systemy Informacyjne

System do zdalnej pracy w środowisku graficznym wykorzystujący
maszyny wirtualne QEMU z akceleracją sprzętową

Krzysztof Smogór

Numer albumu 298906

Piotr Widomski

Numer albumu 298919

promotor

dr inż. Marek Kozłowski

WARSZAWA 2022

.....

podpis promotora

.....

podpisy autorów

Streszczenie

System do zdalnej pracy w środowisku graficznym wykorzystujący maszyny wirtualne QEMU z akceleracją sprzętową

Celem projektu jest stworzenie systemu do zdalnej pracy, za pomocą zdalnego pulpitu, wykorzystującego maszyny wirtualne. Jednym z głównych wyróżników projektu jest możliwość użycia maszyn z akceleracją sprzętową oraz integracja z zewnętrznym systemem katalogowym.

Na wstępie projektu przedstawiony został problem wynikający ze zdalnej pracy oraz wizja systemu, który ma go niwelować wraz z wymaganiami, które musi spełniać. W następnym rozdziale znajduje się dokładny opis rozwiązania z wyróżnieniem poszczególnych modułów oraz procesów. Zawarte zostały również wykorzystane technologie, instrukcja uruchomienia systemu oraz sposób użytkowania. Kolejny rozdział przedstawia analizę rozwiązania skupiającą się na sposobie testowania poprawności działania systemu oraz jego wynikach. Zakończenie opisuje ostateczny stan projektu z wyróżnieniem elementów, które naszym zdaniem wyróżniają się gorszą jakością. Wymienione zostały aspekty o widocznych możliwościach rozwoju oraz znane problemy.

Słowa kluczowe: maszyny wirtualne, zdalny pulpit, libvirt, chmura, praca zdalna

Abstract

Environment for remote work with Graphical User Interface using QEMU virtual machines with hardware acceleration

Goal of thesis is to create environment for remote work, using remote desktop, based on virtual machines. One of main traits is ability to use machines with hardware acceleration and integration with external directory system.

Introduction describes problem addressed by thesis and proposed environment, aiming to ease it. Chapter also contains technical requirements, which the project must fulfil. Next chapter describes in detail the designed environment, including created modules and business processes, used technologies and deployment and use instruction. Following chapter analyses the created solution focusing on testing operations and fulfillment of requirements. Final chapter describes final state of solution with emphasis on components that stand apart in quality. Possible development paths and open problems were presented.

Keywords: virtual machines, remote desktop, libvirt, cloud, remote work

Załącznik nr 1 do zarządzenia nr 109 /2021

Rektora PW z dnia 9 listopada 2021 r.

załącznik nr 5 do zarządzenia nr 42 /2020 Rektora PW



Politechnika Warszawska

.....
miejscowość i data

.....
imię i nazwisko studenta

.....
numer albumu

.....
kierunek studiów

OŚWIADCZENIE

Świadomy/-a odpowiedzialności karnej za składanie fałszywych zeznań oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie, pod opieką kierującego pracą dyplomową.

Jednocześnie oświadczam, że:

- niniejsza praca dyplomowa nie narusza praw autorskich w rozumieniu ustawy z dnia 4 lutego 1994 roku o prawie autorskim i prawach pokrewnych (Dz.U. z 2021 r., poz. 1062) oraz dóbr osobistych chronionych prawem cywilnym,
- niniejsza praca dyplomowa nie zawiera danych i informacji, które uzyskałem/-am w sposób niedozwolony,
- niniejsza praca dyplomowa nie była wcześniej podstawą żadnej innej urzędowej procedury związanej z nadawaniem dyplomów lub tytułów zawodowych,
- wszystkie informacje umieszczone w niniejszej pracy, uzyskane ze źródeł pisanych i elektronicznych, zostały udokumentowane w wykazie literatury odpowiednimi odnośnikami,
- znam regulacje prawne Politechniki Warszawskiej w sprawie zarządzania prawami autorskimi i prawami pokrewnymi, prawami własności przemysłowej oraz zasadami komercjalizacji.

.....
czytelny podpis studenta

Załącznik nr 3 do zarządzenia nr 109 /2021

Rektora PW z dnia 9 listopada 2021 r.

załącznik nr 9 do zarządzenia nr 42 /2020 Rektora PW



Politechnika Warszawska

.....
miejsowość i data

.....
imię i nazwisko studenta

.....
numer albumu

.....
Wydział i kierunek studiów

Oświadczenie studenta w przedmiocie udzielenia licencji
Politechnice Warszawskiej

Oświadczam, że jako autor/współautor* pracy dyplomowej pt.
..... udzielam/nie udzielam* Politechnice Warszawskiej
nieodpłatnej licencji na niewyłączne, nieograniczone w czasie, umieszczenie pracy dyplomowej w elek-
tronicznych bazach danych oraz udostępnianie pracy dyplomowej w zamkniętym systemie bibliotecznym
Politechniki Warszawskiej osobom zainteresowanym.

Licencja na udostępnienie pracy dyplomowej nie obejmuje wyrażenia zgody na wykorzystywanie pracy
dyplomowej na żadnym innym polu eksploatacji, w szczególności kopiowania pracy dyplomowej w całości
lub w części, utrwalania w innej formie czy zwielokrotniania.

.....
czytelny podpis studenta

* niepotrzebne skreślić

Spis treści

1. Wstęp	11
1.1. Opis problemu	11
1.2. Podobne rozwiązania	11
1.3. Wizja systemu	12
1.4. Istotne pojęcia	13
1.5. Wymagania funkcjonalne	15
1.5.1. Nadzorca	15
1.5.2. Serwer wirtualizacji	17
1.5.3. Panel administratora	19
1.6. Wymagania niefunkcjonalne	21
1.7. Analiza ryzyka	22
1.7.1. Omówienie zagrożeń	22
1.8. Podział pracy	24
2. Opis rozwiązania	25
2.1. Architektura systemu	25
2.1.1. Model systemu	25
2.1.2. Nadzorca	27
2.1.3. Serwer wirtualizacji	28
2.1.4. Aplikacja kliencka	28
2.1.5. Panel administratora	28
2.1.6. Broker wiadomości	28
2.2. Stany biznesowe	29
2.2.1. Maszyna wirtualna	29
2.2.2. Serwer wirtualizacji	31
2.2.3. Użytkownik	32
2.3. Procesy biznesowe	32
2.3.1. Uzyskanie sesji	32

2.3.2.	Kończenie sesji	33
2.3.3.	Rozpoczęcie pracy serwera wirtualizacji	34
2.4.	Komunikacja	36
2.4.1.	Komunikacja wewnętrzna	36
2.4.2.	Komunikacja zewnętrzna	36
2.5.	Sekwencje komunikacji	38
2.5.1.	Utworzenie sesji	38
2.5.2.	Zakończenie sesji	39
2.5.3.	Aktualizacja stanu	40
2.5.4.	Włączenie maszyny	40
2.5.5.	Wyłączenie maszyny	41
2.6.	Wykorzystane technologie	41
2.6.1.	Technologie realizacji strony klienckiej	41
2.6.2.	Technologie realizacji strony serwerowej	42
2.6.3.	Technologie testowania	43
2.6.4.	Utrzymanie kodu	43
2.7.	Wymagania systemu	43
2.7.1.	Komunikacja sieciowa między modułami	44
2.7.2.	Wymagania aplikacji klienckiej	45
2.7.3.	Wymagania aplikacji nadzorcy i serwera panelu administracyjnego	46
2.7.4.	Wymagania serwera wirtualizacji	46
2.7.5.	Automatyzacja konfiguracji	47
2.7.6.	Wymagania szablonu maszyny wirtualnej	48
2.8.	Uruchomienie systemu	49
2.8.1.	Pozyskanie aplikacji klienckiej	49
2.8.2.	Budowanie kontenerów	49
2.8.3.	Budowanie modułów	50
2.8.4.	Konfiguracja aplikacji klienckiej	51
2.8.5.	Konfiguracja panelu administracyjnego	51
2.8.6.	Konfiguracja nadzorcy	51
2.8.7.	Konfiguracja serwera wirtualizacji	54
2.8.8.	Procedura uruchomienia	59
2.8.9.	Parametry uruchomienia kontenera panelu administracyjnego	59
2.8.10.	Parametry uruchomienia kontenera nadzorcy	60

2.8.11.	Parametry uruchomienia kontenera serwera wirtualizacji	61
2.8.12.	Przykład minimalnego systemu	62
2.9.	Interakcja z systemem	62
2.9.1.	Typowe działanie systemu	62
2.9.2.	Funkcje panelu administracyjnego	62
2.9.3.	Funkcje aplikacji klienckiej	65
3.	Analiza rozwiązania	71
3.1.	Testowana funkcjonalność	71
3.1.1.	Brak komunikacji z nadzorcą	71
3.1.2.	Utrata komunikacji z nadzorcą	71
3.1.3.	Standardowe użycie systemu przez użytkownika	72
3.1.4.	Standardowe użycie systemu przez użytkownika przy awarii nadzorcy	72
3.1.5.	Podłączenie nowego serwera wirtualizacji	73
3.1.6.	Podłączenie nowego nadzorcy	73
3.1.7.	Odnotowanie utraty serwera wirtualizacji	73
3.1.8.	Utrata komunikacji przy działającej sesji	74
3.1.9.	Odzyskanie komunikacji przy działającej sesji	74
3.2.	Środowisko testowe	75
3.3.	Wykonane testy	75
3.4.	Wyniki testów	76
4.	Podsumowanie	77
4.1.	Końcowy stan systemu	77
4.2.	Możliwe ścieżki rozwoju	78
4.2.1.	Panel administratora	78
4.2.2.	Użycie konkretnych kart graficznych	78
4.2.3.	Wsparcie innych protokołów zdalnego pulpitu	78
4.3.	Otwarte problemy	79
4.3.1.	Konfiguracja połączenia RabbitMQ	79
4.3.2.	Monitorowanie stanu połączenia klienta	79
4.3.3.	Znane problemy wyścigów	79

1. Wstęp

1.1. Opis problemu

Można aktualnie zaobserwować dużą zmianę na rynku pracy. Z powodu globalnej epidemii wiele firm zdecydowało się na zmianę pracy stacjonarnej na zdalną. W 2020 roku około 70% pracowników zatrudnionych na pełen etat w Stanach Zjednoczonych pracowało z domu [1]. Nawet po złagodzeniu obostrzeń znaczna część miejsc pracy pozostała przy takim trybie lub przyjęło hybrydową formę pracy. Około 89% europejskich firm planuje pozostać czy hybrydowym trybie pracy nawet po zakończeniu epidemii [2]. Taka forma pracy prowadzi ma jednak własne utrudnienia. W pewnych przypadkach pracownicy muszą łączyć się za pomocą funkcji zdalnego pulpitu z komputerami znajdującymi się w biurze. Może to wynikać z niewystarczającej wydajności sprzętu pracownika lub dostępu do specyficznych programów albo zasobów. W takich sytuacjach komputer, z którym łączy się użytkownik, musi być uruchomiony, a w przypadku awarii - zrestartowany. Dodatkowo taki dostęp może być wymagany jedynie przez ograniczony czas, co powoduje, że dużą część czasu maszyna spędza włączona, ale nieużywana.

Możliwym sposobem na złagodzenie tego problemu jest użycie zmniejszonej liczby stacji roboczych, które mogą być używane przez większą liczbę pracowników jednocześnie za pośrednictwem maszyn wirtualnych. Zmniejsza to liczbę działających maszyn fizycznych, a zarządzanie wirtualnymi może być wykonywane zdalnie.

Tworzony przez nas system umożliwi zmniejszenie liczby stanowisk na rzecz jednego *przezroczystego komputera*. Pracownicy będą mogli poprosić o zdalny dostęp do maszyny z zasobami systemowymi dopasowanymi do ich potrzeb. System sam zadba o udostępnienie i zarządzanie tymi zasobami.

1.2. Podobne rozwiązania

Podobnymi rozwiązaniami oferującymi możliwość zdalnej pracy przy użyciu wirtualnych pulpików są między innymi *Amazon WorkSpaces* (<https://aws.amazon.com/workspaces/>), *Para-*

rells RAS (url <https://www.parallels.com/products/ras/remote-application-server/>) oraz *Citrix* (<https://www.citrix.com/pl-pl/>). Usługa *Citrix Virtual Apps and Desktops* [14] oferowana przez firmę Citrix jest bardzo podobna do naszego rozwiązania. Przy zakupie dostępu można wybrać czy interesuje nas udostępnianie całego pulpitu zdalnego jako serwisu (tzw. DaaS - Desktop as a Service), czy zdalny dostęp do wybranego komputera.

System oferowany przez Citrixa ten posiada także zbiór komputerów łączonych w klastrer oraz aplikacje balansującą, która równomiernie rozkłada zużycie wszystkich maszyn w klastrze. Możliwe jest skorzystanie z gotowej infrastruktury zaoferowanej przez Citrixa albo wybudowanie własnego klastra.

Jedną z ważniejszych różnic Citrixa od naszego systemu jest większa wirtualizacja zasobów podłączonych do maszyny wirtualnej. W naszym systemie możliwe jest bardziej statyczne przyporządkowanie zasobów (np. karta graficzna) co pozwala na uzyskanie lepszej wydajności do nietypowych zastosowań biurowych (np. symulacje numeryczne w programach typu CAD). Możliwe jest przekazanie do maszyny wirtualnej dowolnego urządzenia PCI zdefiniowanego w konfiguracji. Umożliwia to pracę z bardzo nietypowymi urządzeniami, nawet zdalnie.

Dodatkowo nasz system pozwala łatwo wykorzystać istniejące stacje robocze do stworzenia klastra. Konfiguracja każdego komputera w klastrze może być ustawiona oddzielnie. System odpowiednio zarządzi różnymi typami maszyn. Może to obniżyć koszty, jeżeli pracownicy często zmieniają swój tryb pracy.

1.3. Wizja systemu

Tworzony system ma za zadanie umożliwiać zdalną pracę za pomocą protokołu zdalnego pulpitu. Rozwiązanie pozwala na wykorzystanie istniejących stacji roboczych do tworzenia wirtualnych stanowisk.

Użytkownikami końcowymi są pracownicy, którzy za pomocą okienkowej aplikacji klienckiej mogą uzyskać sesję do pracy zdalnej. Użytkownik podczas pracy łączy się za pomocą protokołu zdalnego pulpitu z maszyną wirtualną uruchamiającą obraz systemu GNU/Linux. Uruchamianie i zarządzanie maszynami jest zadaniem aplikacji działającej na rzeczywistej stacji roboczej, która udostępnia swoje zasoby maszynom wirtualnym. Aplikacja ta, oraz rzeczywista maszyna uruchamiająca ją, nazywana jest dalej serwerem wirtualizacji. Serwery działają niezależnie od siebie i nie ma teoretycznego ograniczenia na ich liczbę w systemie. Komunikacją z użytkownikami oraz zarządzaniem systemem zajmuje się aplikacja nadzorcza. Ilość jej instancji również jest teoretycznie nieograniczona co umożliwia balansowanie obciążeniem oraz zwiększa odporność systemu na

1.4. ISTOTNE POJĘCIA

awarie.

System pozwala na tworzenie maszyn wirtualnych różnych typów, czyli kombinacji zasobów systemowych udostępnianych dla maszyny wirtualnej oraz faktu czy ma ona bezpośredni dostęp do karty graficznej maszyny, na której pracuje. Do używania systemu użytkownik musi posiadać konto w systemie katalogowym, na które loguje się podczas użytkowania systemu. Foldery domowe pracowników montowane są wewnątrz maszyny z zewnętrznego dysku sieciowego i umożliwiają przechowywanie danych między połączeniami.

System udostępnia panel administracyjny w postaci strony WWW umożliwiający podgląd obciążenia i stanu systemu przez upoważnione osoby. Komunikacja aplikacji klienckiej z aplikacją nadzorczą oraz panel administratora wykorzystują komunikację za pomocą protokołu HTTP. Możliwe jest użycie szyfrowanego protokołu HTTPS [5], pod warunkiem użycia poprawnych certyfikatów SSL/TSL.

1.4. Istotne pojęcia

- **Aplikacja kliencka** - aplikacja okienkowa uruchamiana na komputerze użytkownika, która umożliwia komunikację z systemem oraz uruchomienie zewnętrznego programu implementującego protokół RDP. Działa pod systemem operacyjnym Windows 10 oraz GNU/Linux.
- **Aplikacja nadzorcza (Nadzorca)** - aplikacja, która przetwarza zapytania od aplikacji klienckiej oraz komunikuje się ze wszystkimi serwerami wirtualizacji. Na podstawie tych informacji buduje model zajętości każdego z serwerów wirtualizacji oraz decyduje kiedy, i na którym serwerze, trzeba uruchomić nowe maszyny wirtualne. Decyduje również, do której wirtualnej maszyny ma podłączyć się użytkownik proszący o utworzenie sesji. Aplikacja może działać na dowolnym komputerze spełniającym wymagania.
- **Serwer wirtualizacji** - komputer, który udostępnia swoje zasoby (rdzenie procesora, karty graficzne, pamięć RAM oraz przestrzeń dyskową) w postaci uruchamianych na nim maszyn wirtualnych. Komputer ten uruchamia aplikację, która odpowiada na zapytania aplikacji nadzorczej oraz wykonuje operacje na maszynach wirtualnych (uruchamianie i wyłączanie). Komputer może uruchamiać co najwyżej jedną aplikację, dlatego zarówno maszynę, jak i aplikację, nazywamy serwerem wirtualizacji. Uruchamiana na stacji roboczej, która ma być częścią klastra.
- **Broker wiadomości** - aplikacja przekazująca wiadomości pomiędzy połączonymi z nią innymi aplikacjami.

- **Maszyna wirtualna CPU** - maszyna systemowa emulująca, lub para-emulująca, sprzęt i służąca do uruchamiania systemu operacyjnego. Udostępnia użytkownikowi podstawowe zasoby (procesor, pamięć RAM i przestrzeń dyskowa). Uruchamiana jest na serwerze wirtualizacji z liczbą zasobów określoną w konfiguracji. Maszyna wirtualna uruchamia system operacyjny GNU/Linux (ArchLinux - <https://archlinux.org/>).
- **Maszyna wirtualna GPU** - maszyna analogiczna do maszyny wirtualnej CPU. Wyróżnia się przekazaną na wyłączność, za pośrednictwem mechanizmu PCI Passthrough, kartą graficzną podłączoną do serwera wirtualizacji.
- **RDP** - protokół zdalnego dostępu do pulpitu od firmy Microsoft [23]. Maszyny wirtualne uruchamiają serwer RDP (xrdp - <http://xrdp.org/>), który umożliwia pracę za pośrednictwem protokołu zdalnego dostępu.
- **Sesja** - jednorazowy dostęp użytkownika do systemu oraz maszyny wirtualnej. Utworzenie sesji wiąże się z przypisaniem do użytkownika konkretnej maszyny wirtualnej, na której będzie pracować. Sesja kończy się w przypadku, gdy użytkownik poinformuje system o zakończeniu pracy lub gdy minie czas oczekiwania na odzyskanie połączenia po jego utracie.
- **Vagrant Box [21]** - przygotowany wcześniej obraz maszyny wirtualnej, który umożliwia zmianę dostępnych zasobów. Uruchamiają się z obrazu pierwotnego w środowisku programu Vagrant. Obrazy te używane są do tworzenia maszyn wirtualnych.
- **Ansible playbook [17]** - skrypt konfiguracyjny dla systemu operacyjnego, który umożliwia parametryzację oraz wykonywanie określonych akcji po uruchomieniu Vagrant Boxa.
- **Panel administratora** - aplikacja przeglądarkowa, która umożliwia administratorowi systemu podgląd statusu serwerów wirtualizacji znajdujących się w systemie oraz zajętość zasobów.
- **Konto użytkownika** - profil użytkownika w systemie, do którego ma dostęp na każdej maszynie wirtualnej. Używane do logowania się do systemu oraz maszyn wirtualnych. Przechowywane jest w zewnętrznym systemie katalogowym.
- **Katalog użytkownika** - prywatny folder dostępny dla użytkownika na każdej maszynie wirtualnej. Przechowywany na zewnętrznym dysku sieciowym oraz montowane na maszynach wirtualnych.
- **Konfiguracja stała** - konfiguracja maszyny wirtualnej, która nie zmienia się w zależności od miejsca uruchomienia. Zapisana jest w Vagrant Boxie. W razie potrzeby można ją także

1.5. WYMAGANIA FUNKCJONALNE

zdefiniować w odpowiednim Ansible playbooku.

- **Konfiguracja zmienna** - konfiguracja maszyny wirtualnej, która zmienia się w zależności od miejsca uruchomienia. Jest definiowana w odpowiednim Ansible playbooku uruchamianym przy każdym włączeniu maszyny.

1.5. Wymagania funkcjonalne

1.5.1. Nadzorca

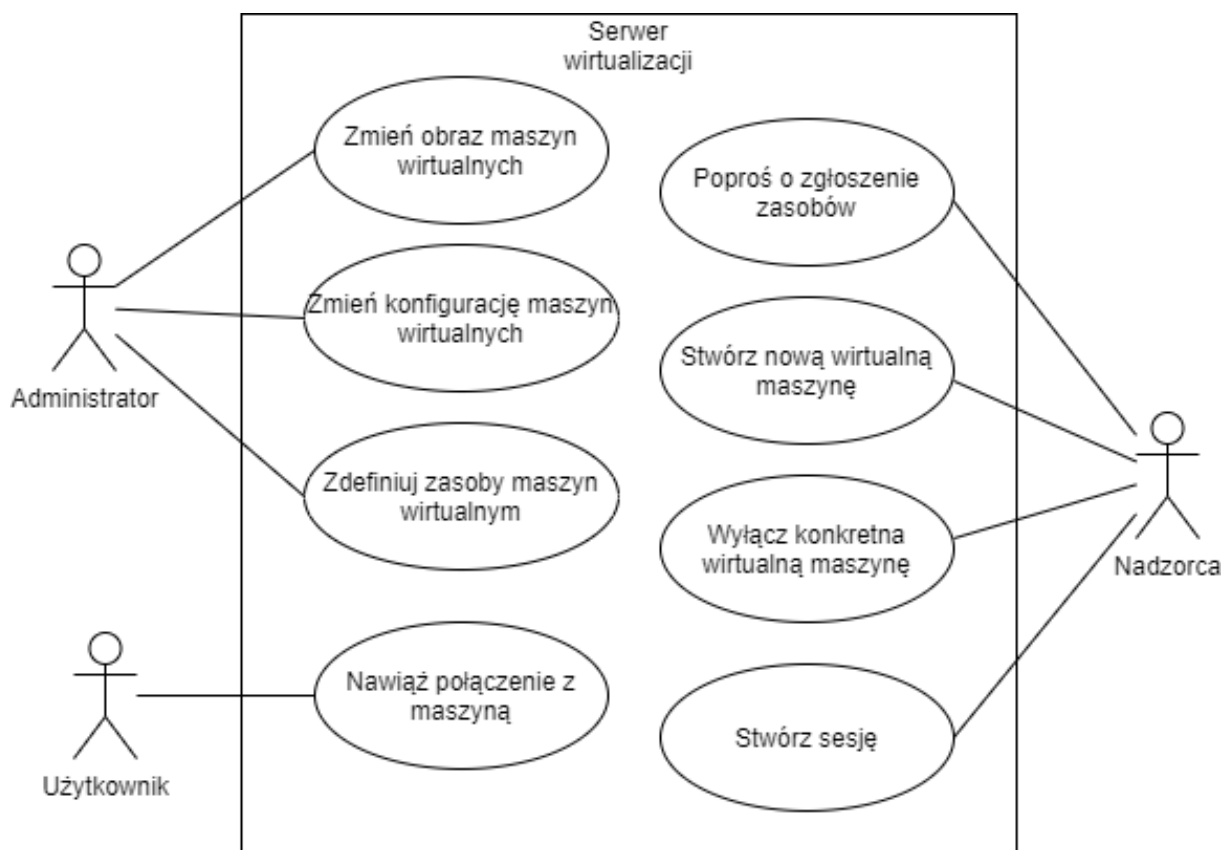


Rysunek 1.1: Przypadki użycia aplikacji nadzorczej

Tablica 1.1: Przypadki użycia aplikacji nadzorczej

Aktor	Nazwa	Opis	Odpowiedź systemu
Użytkownik	Uzyskanie sesji do pracy	Uzyskanie sesji do pracy na maszynie wirtualnej odpowiedniego typu.	Do użytkownika zostaje przydzielona maszyna wirtualna oraz zestawione połączenie RDP. Jeżeli przypisana do użytkownika sesja nie została jeszcze zakończona, to system przydziela mu tę samą maszynę.
	Poznanie ilości dostępnych maszyn	Wyświetlanie szacowanej ilości dostępnych maszyn każdego typu.	Użytkownikowi zostaje wyświetlona szacowana liczba dostępnych maszyn obliczona na podstawie informacji o dostępnych zasobach każdego z serwerów wirtualizacji.
Server wirtualizacji	Zgłoszenie dostępnych zasobów	Serwer zgłasza nadzorcy dostępne zasoby.	Nadzorca wykorzystuje zgłoszone zasoby do wyliczenia szacowanej liczby dostępnych maszyn oraz balansowania obciążeniem serwerów wirtualizacji.
Panel administratora	Podgląd stanu modelu	Nadzorca udostępnia panelowi administratora stan zasobów systemu.	Panel administratora wykorzystuje uzyskane dane do wyświetlenia administratorowi raportu o stanie systemu.

1.5.2. Serwer wirtualizacji

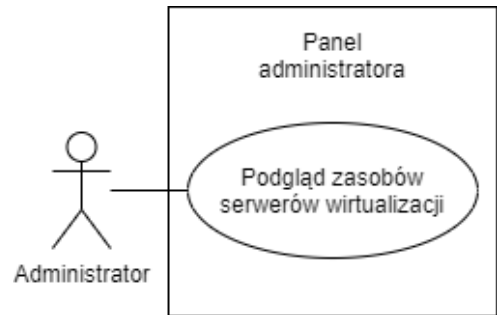


Rysunek 1.2: Przypadki użycia serwera wirtualizacji

Tablica 1.2: Przypadki użycia serwera wirtualizacji

Aktor	Nazwa	Opis	Odpowiedź systemu
Użytkownik	Nawiązanie połączenia z maszyną	Użytkownik nawiązuje połączenie z maszyną wirtualną.	Maszyna wirtualna zostaje zajęta przez użytkownika; serwer wirtualizacji rozpoczyna monitorowanie, czy sesja wciąż trwa.
Nadzorca	Poproś o zgłoszenie zasobów	Nadzorca wysyła do wszystkich serwerów wirtualizacji prośbę o zgłoszenie swoich używanych i wolnych zasobów.	Serwer wirtualizacji informuje nadzorcę o stanie swoich zasobów.
	Stwórz nową maszynę wirtualną	Nadzorca prosi serwer wirtualizacji o stworzenie nowej maszyny wirtualnej dla danego użytkownika na wybranym typie maszyny.	Serwer wirtualizacji tworzy maszynę wirtualną wybranego typu i udostępnia możliwość połączenia się z nią.
	Wyłącz konkretną maszynę wirtualną	Nadzorca prosi serwer wirtualizacji aby wyłączył konkretną maszynę wirtualną.	Serwer wirtualizacji wyłącza konkretną maszynę wirtualną oraz pilnuje aby na pewno się wyłączyła.
Administrator	Zmień obraz maszyn wirtualnych	Zmiana obrazu źródłowego maszyn wirtualnych.	Zdefiniowany przez administratora Vagrant Box jest używany przez serwery wirtualizacji.
	Zmień konfigurację maszyn wirtualnych	Zmiana zmiennej konfiguracji maszyn wirtualnych.	Zmodyfikowany ansible playbook jest używany przez serwery wirtualizacji.
	Zdefiniuj zasoby maszyn wirtualnych	Zmiana łącznej ilości zasobów przeznaczonych na maszyny.	Zmodyfikowana konfiguracja zasobów będzie wykorzystywana przez serwer wirtualizacji przy kolejnym uruchomieniu.

1.5.3. Panel administratora



Rysunek 1.3: Przypadki użycia panelu administratora

Tablica 1.3: Przypadki użycia panelu administratora

Aktor	Nazwa	Opis	Odpowiedź systemu
Administrator	Podgląd zasobów serwerów wirtualizacji	Wyświetlanie wolnych oraz zajętych zasobów serwerów wirtualizacji.	Wyświetlenie zasobów poszczególnych serwerów wirtualizacji, liczby zajętych maszyn oraz szacowanej liczby wolnych maszyn.

1.6. Wymagania niefunkcjonalne

Tablica 1.4: Wymagania niefunkcjonalne

Grupa wymagań	Nr wymagania	Opis
Użytkowanie (Usability)	1	Aplikacja kliencka ma działać na systemach operacyjnych MS Windows (Windows 10) oraz GNU/Linux (Arch Linux). Aplikacja na systemach GNU/Linux wymaga zainstalowanego klienta RDP zgodnego z XRDP [11].
	2	Aplikacja kliencka musi udostępniać możliwość użycia własnego klienta RDP do nawiązania połączenia z maszyną wirtualną.
	3	Maszyny wirtualne muszą mieć dostęp do systemu przechowującego konta użytkowników wraz z ich katalogami domowymi.
Nieawodność (Reliability)	4	System musi być odporny na awarie poszczególnych serwerów wirtualizacji i kontynuować działanie w sposób niezauważalny dla użytkowników nie używających danego. serwera.
	5	Awaria nadzorca może spowodować uniemożliwienie rozpoczęcia nowych sesji, ale nie może przerwać istniejących sesji.
Wydajność (Performance)	6	Łącznie zużywane zasoby przez maszyny wirtualne na poszczególnym serwerze wirtualizacji nie mogą przekroczyć wcześniej zdefiniowanych. limitów
	7	Nadzorca musi balansować obciążeniem serwerów wirtualizacji.
	8	W systemie zawsze musi istnieć jedna działająca maszyna wirtualna nie połączona z żadną sesją, aby można było ją szybko przydzielić użytkownikowi.
	9	Zwolnione maszyny wirtualne, które nie są wykorzystywane jako zapas, muszą być wyłączane
Utrzymanie (Supportability)	10	Możliwe jest działanie więcej niż jednego nadzorca w systemie, w celu zwiększenia dostępności lub przeprowadzenia prac utrzymaniowych

1.7. Analiza ryzyka

Tablica 1.5: Analiza ryzyka

<p>Mocne strony</p> <ul style="list-style-type: none"> • Łatwa skalowalność pod względem liczby sesji w systemie. • Wiele rozwiązań Open Source. • Elastyczność pod względem konfiguracji. • Tańsze rozwiązanie niż kupno stacji roboczych. 	<p>Słabości</p> <ul style="list-style-type: none"> • System trudny w konfiguracji. • Potrzeba wymiany sprzętu komputerowego. • Krótki czas rozwoju systemu. • Ograniczone doświadczenie twórców systemu. • Małe prawdopodobieństwo dalszego wsparcia projektu po zakończeniu prac.
<p>Okazje</p> <ul style="list-style-type: none"> • Grupa docelowa to firmy z dużą ilością stacji roboczych. • Zwiększenie zapotrzebowania na pracę zdalną na rynku pracy. 	<p>Zagrożenia</p> <ul style="list-style-type: none"> • Istnienie konkurencji ugruntowanej na rynku. • System w dużej mierze oparty o oprogramowanie rozwijane przez inne organizacje.

1.7.1. Omówienie zagrożeń

- System trudny w konfiguracji - wysoko prawdopodobne

Można temu zaradzić poprzez udostępnienie dokładnej dokumentacji lub ścisłą współpracę z klientem przy wdrażaniu systemu.

Waga: duża

- Potrzeba wymiany sprzętu komputerowego - średnio prawdopodobne

Klient może potrzebować wymienić aktualne stacje robocze na terminale oraz zainwestować w sprzęt serwerowy. Jednak gdy klientami będą firmy, które mają dużo pracowników pracujących spoza biura, lub dopiero tych pracowników pozyskują, to kupno terminali i

1.7. ANALIZA RYZYKA

serwerów powinno być bardziej zachęcające niż kupno stacji roboczych.

Waga: średnia.

- Krótki czas rozwoju systemu - wysoko prawdopodobne

Czas rozwoju systemu jest bardzo ograniczony. Aby pomimo tego ograniczenia działał on w sposób akceptowalny powinniśmy skupić się na dobrym przedyskutowaniu i opisanu kluczowych modułów systemu. W czasie projektu należy pilnować aby nie dodawać nadmiarowych funkcjonalności do systemu. W czasie implementacji konieczne będzie dokładne zaplanowanie aplikacji pod kątem testowania automatycznego. Ułatwi to wyłapywanie prostych błędów jeszcze we wczesnej fazie projektu.

Waga: wysoka

- Ograniczone doświadczenie twórców systemu - pewne

Jedynym sposobem na ograniczenie ryzyka jest rozważna implementacja.

Waga: średnia

- Małe prawdopodobieństwo wsparcia projektu po zakończeniu prac - wysoko prawdopodobne

Trudno teraz przewidzieć co się stanie z projektem po zakończeniu prac. Jednak prawdopodobnie twórcy systemu zajmą się innymi projektami. Można jedynie dokładnie komentować kod i pokrywać jak najwięcej jego części testami. Wtedy inne osoby będą w stanie szukać błędów albo próbować w taki sposób uzupełnić brakującą wiedzę o systemie.

Waga: niska

- Istnienie konkurencji ugruntowanej na rynku - bardzo prawdopodobne

Konkurencyjne systemy oferujące podobne rozwiązania są już dobrze ugruntowane na rynku i przetestowane. Nasz system może spróbować konkurować jedynie z nimi ceną implementacji oraz elastycznością.

Waga: średnia

- System w dużej mierze oparty o oprogramowanie rozwijane przez inne organizacje - nisko prawdopodobne

W czasie życia systemu mogą pojawić się błędy w oprogramowaniu nie rozwijanym w ramach naszego systemu. Naprawa takich błędów może trwać bardzo długo. Pewnym sposobem rozwiązania takiego problemu jest własnoręczne poprawianie błędów w zewnętrznym oprogramowaniu i zgłaszanie ich do odpowiednich organizacji. Do czasu zastosowania po-

prawki jest możliwość korzystania z wersji, na którą nanieśliśmy własną poprawkę.

Waga: wysoka

1.8. Podział pracy

W czasie realizacji systemu podzieliśmy się pracą:

- Krzysztof Smogór - aplikacja serwera wirtualizacji (w tym integracja z Vagrantem oraz libvirtem) oraz aplikacja nadzorcy (przetwarzanie informacji o systemie oraz udostępnienie ich aplikacjom klienckim)
- Piotr Widomski - aplikacja kliencka (wraz z integracją z klientami RDP), aplikacja panelu administracyjnego, struktura modelu systemu, komunikacja pomiędzy modułami oraz integracja z systemem katalogowym i dyskiem sieciowym.

W czasie pisania pracy dyplomowej podział był następujący:

- Krzysztof Smogór - Analiza rozwiązania (rozdział 3) oraz część opisu rozwiązania związana z konfiguracją i wymaganiami (rozdział od 2.6 do 2.9).
- Piotr Widomski - Wstęp (rozdział 1), podsumowanie (rozdział 4) oraz teoretyczna część opisu rozwiązania (rozdział od 2.1 do 2.5)

2. Opis rozwiązania

2.1. Architektura systemu

Opracowywany system składa się z następujących modułów:

- nadzorcy,
- serwera wirtualizacji,
- aplikacji klienckiej,
- panelu administratora,
- brokera wiadomości,
- systemu katalogowego,
- dysku współdzielonego.

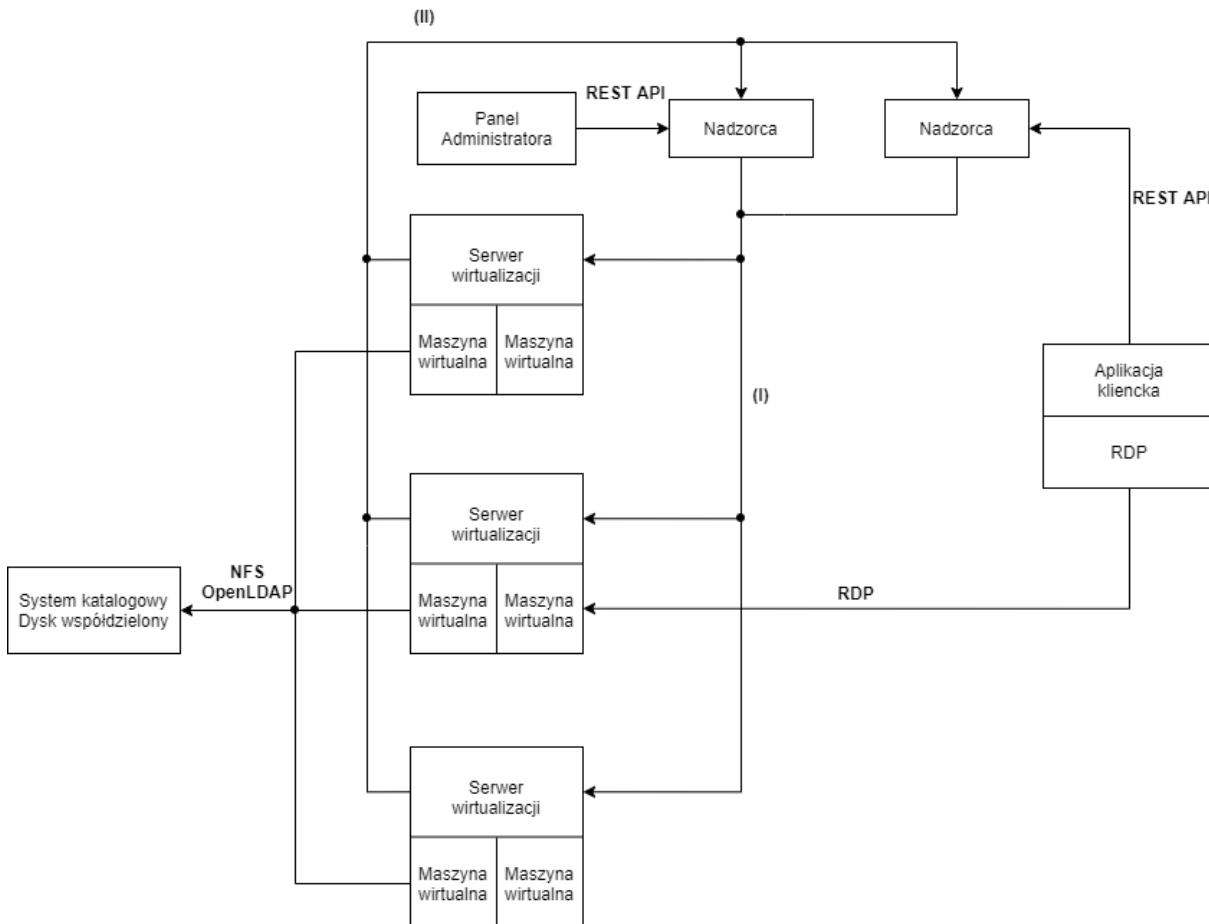
Schematyczny obraz systemu przedstawiony został na rysunku 2.1.

Połączenia oznaczone liczbami rzymskimi oznaczają kolejki komunikacji za pośrednictwem brokera wiadomości, które opisane zostały w sekcji 2.1.6. Z założenia system powinien móc skalować się w dwóch wymiarach, to znaczy:

- Zwiększanie liczby serwerów wirtualnych - zwiększenie liczby istniejących jednocześnie sesji.
- Zwiększenie liczby nadzorców - zwiększenie liczby obsługiwanych jednocześnie klientów oraz niezawodności systemu.

2.1.1. Model systemu

W celu zarządzania systemem każdy z nadzorców musi posiadać dokładną wiedzę o jego aktualnym stanie. Informacje te przechowuje w strukturze nazywanej dalej modelem systemu. Ważne jest, że klasy modelu nie wykonują żadnych akcji poza modyfikacją przechowywanych danych. Wszystkie metody służą jedynie do zmiany stanu przechowywanego modelu w celu dopasowania



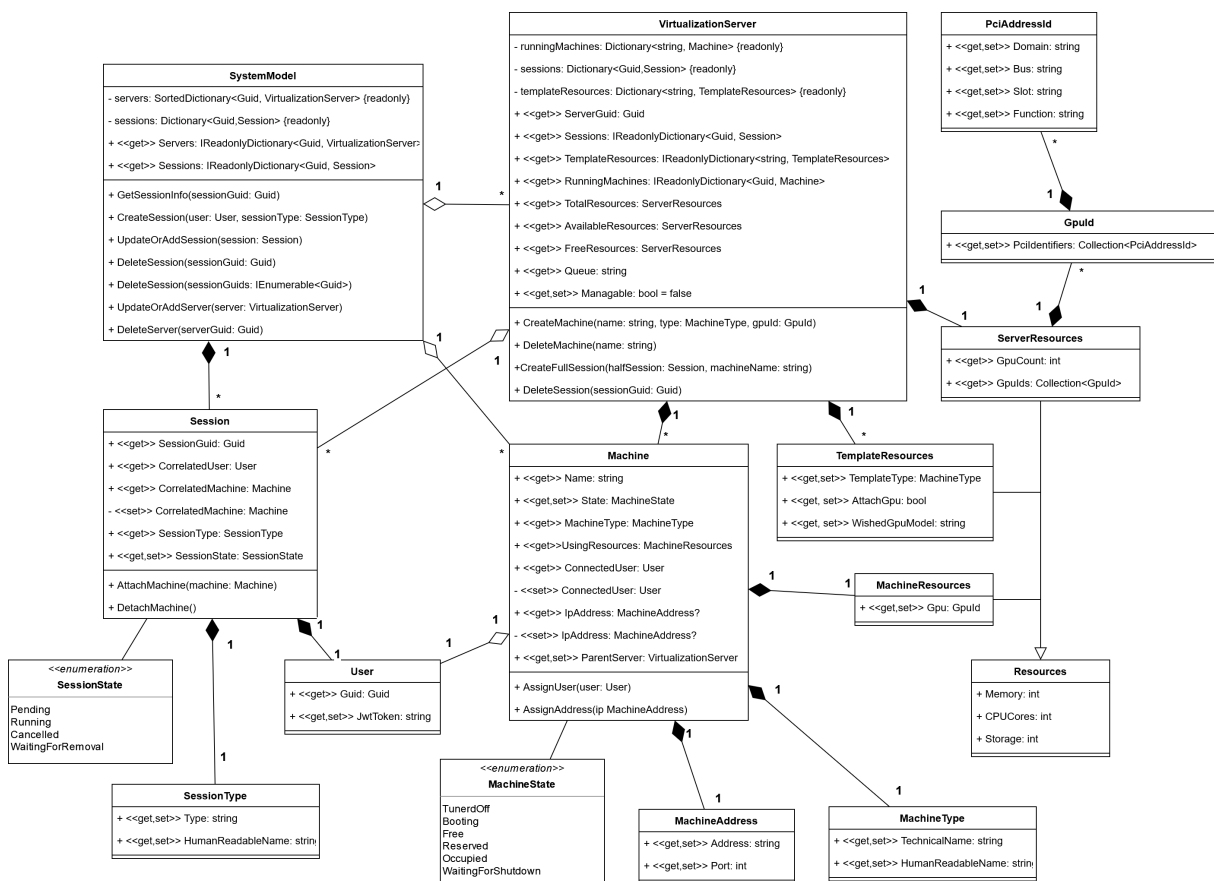
Rysunek 2.1: Schematyczna architektura systemu

go do rzeczywistego stanu systemu, na podstawie otrzymanych danych. Przyczyną takiego rozwiązania jest pierwsze z założeń komunikacji opisanych w sekcji 2.4.1. Sprawia ono, że zmiana modelu musi być następstwem pewnej akcji wykonanej przez serwer wirtualizacji. Schemat klas przedstawia rysunek 2.2.

Główną klasą modelu jest **SystemModel** zawierający informacje o aktualnie działających serwerach wirtualizacji oraz aktywnych sesjach. Klasa **VirtualizationServer** modeluje pojedynczy serwer wirtualizacji, jego zasoby, maszyny wirtualne oraz obsługiwane sesje. Przechowuje również informacje wymagane do komunikacji z daną instancją serwera.

Klasa **Resources** oraz klasy pochodne opisują zasoby systemowe i są używane do przedstawienia zarówno zasobów maszyny, jak i całego serwera wirtualizacji. Klasa pochodna **Template Resources** służy do przechowywania informacji o zasobach potrzebnych do utworzenia maszyny danego typu.

2.1. ARCHITEKTURA SYSTEMU



Rysunek 2.2: Schemat klas modelu systemu

2.1.2. Nadzorca

Aplikacja mająca za zadanie obsługiwać komunikację z aplikacjami klienckimi oraz wysyłać polecenia do serwerów wirtualizacji. Udostępnia REST API (sekcja 2.4.2), służące do komunikacji z aplikacjami klienckimi. Do komunikacji z serwerami wirtualizacji wykorzystuje kolejki wiadomości (sekcja 2.1.6).

Nadzorca przechowuje wewnętrznie model systemu zawierający informację o działających serwerach wirtualizacji i stanie ich maszyn. Na podstawie tego modelu moduł stwierdza, do której maszyny przypisać nowo utworzoną sesję. Wewnętrzne procesy skupione są wokół zmian modelu. Jeżeli proces wysłał do serwera wirtualizacji prośbę o zmianę stanu, to dalsze przetwarzanie odbywa się, gdy stan modelu został zaktualizowany i na tej podstawie podejmowane są decyzje.

Dzięki zastosowaniu kolejek wiadomości (sekcja 2.1.6) oraz zasad komunikacji (sekcja 2.4.1) w systemie może istnieć więcej niż jeden nadzorca. Instancje nadzorców działają niezależnie od siebie i przechowują identyczny model systemu. Dzięki temu uzyskujemy retencję i możemy zmniejszyć obciążenie poszczególnych nadzorców.

2.1.3. Serwer wirtualizacji

Zadaniem serwera wirtualizacji jest uruchamianie i zarządzanie maszynami wirtualnymi, z którymi łączy się użytkownik systemu. Komunikuje się on z nadzorcami i wykonuje operacje na maszynach wirtualnych zgodnie z żądaniami.

Moduł ten nie jest w stanie funkcjonować samodzielnie. Z tego powodu aplikacja nie uruchomi się, jeżeli nie jest w stanie nawiązać połączenia z aplikacją nadzorczą. Aplikacja zakończy się, gdy ostatni nadzorca w systemie zakończy działanie, pod warunkiem że nie ma żadnych działających sesji.

Serwer wirtualizacji jest częścią systemu, która przechowuje fizyczne zasoby udostępniane użytkownikom. System zaprojektowany jest w taki sposób aby teoretycznie nie było ograniczenia na liczbę serwerów wirtualizacji działających jednocześnie.

2.1.4. Aplikacja kliencka

Aplikacja okienkowa umożliwiająca użytkownikowi autoryzację, uzyskanie sesji oraz automatyczne rozpoczęcie połączenia. Komunikuje się z nadzorcą za pomocą REST API (rozdział 2.4.2).

Proces uzyskania sesji z perspektywy aplikacji klienckiej zawiera:

1. uzyskanie informacji o dostępnych typach i liczbie maszyn,
2. wybór typu maszyny,
3. oczekiwanie na utworzenie sesji,
4. nawiązanie połączenia RDP,
5. utrzymanie i monitorowanie stanu połączenia,

2.1.5. Panel administratora

Prosta aplikacja internetowa umożliwiająca administratorowi systemu podgląd stanu zużycia zasobów serwerów wirtualizacji.

2.1.6. Broker wiadomości

Komunikacja wewnątrz systemu, czyli pomiędzy serwerami wirtualizacji oraz nadzorcami, realizowana jest poprzez kolejki wiadomości. W tym celu użyty został system RabbitMQ, który zajmuje się transportem wiadomości wewnątrz systemu oraz niezawodnością komunikacji między modułami.

Zdefiniowane zostały następujące kolejki wiadomości (numeracja odpowiada rysunkowi 2.1):

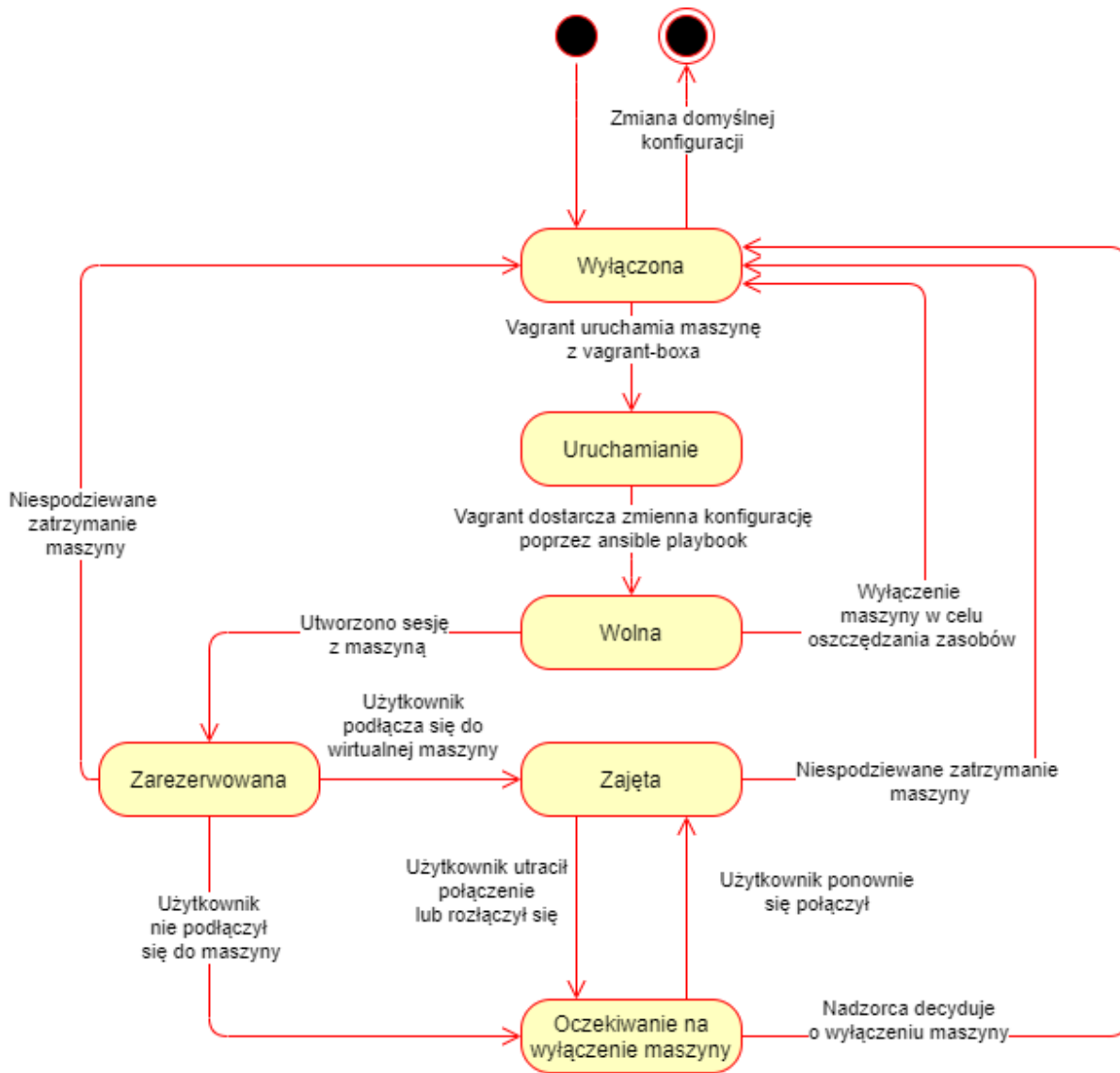
- (I) Kolejka kończąca się na każdym z serwerów wirtualizacji powielająca wiadomości między nimi. Służy ona do wysyłania niespersonalizowanych próśb od nadzorców do serwerów wirtualizacji.
- (II) Kolejka kończąca się na każdym z nadzorców powielająca wiadomości między nimi. Służy ona do przesyłania informacji do nadzorców o zmianach wewnątrz serwerów wirtualizacji.
- (III) Kolejka kończąca się wyłącznie na pojedynczym serwerze wirtualizacji. Liczba kolejek zgadza się z liczbą serwerów wirtualizacji aktywnych w systemie. Służą one do przesyłania spersonalizowanych wiadomości oraz sprawdzania, czy serwer wirtualizacji nadal pracuje po drugiej stronie. Wykorzystana została funkcjonalność kolejek na wyłączność (Exclusive Queue [11]).
- (IV) Kolejka kończąca się na aktualnie podłączonym do maszyny wirtualnej kliencie. Liczba kolejek odpowiada liczbie aktywnych użytkowników. Celem kolejki jest sprawdzenie, czy aplikacja kliencka jest nadal podłączona do wirtualnej maszyny (mechanizm Exclusive Queue). W celach bezpieczeństwa będą one definiowane na oddzielnym procesie brokera, który będzie można w razie potrzeby udostępnić poza sieć lokalną.

Powyższe 4 grupy kolejek umożliwią prawidłowe działanie systemu. W trakcie uruchamiania każdy z modułów tworzy kolejki, z których odbiera wiadomości. Jedynym wymogiem prawidłowego uruchomienia komunikacji jest dostępny dla wszystkich serwerów wirtualizacji oraz nadzorców proces brokera.

2.2. Stany biznesowe

2.2.1. Maszyna wirtualna

Najważniejszym obiektem biznesowym w systemie jest maszyna wirtualna, do której będą podłączać się użytkownicy.

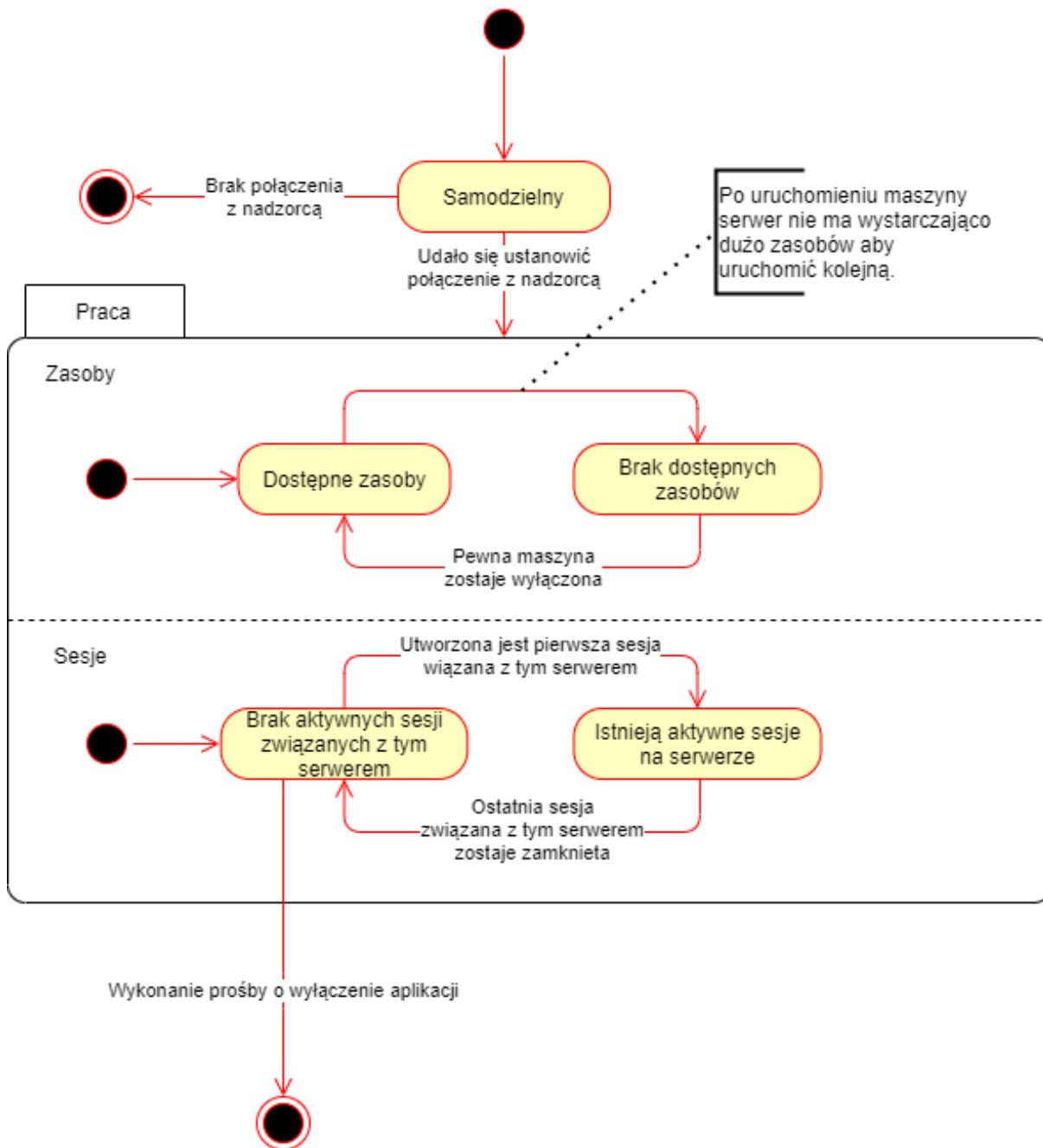


Rysunek 2.3: Diagram stanów maszyny wirtualnej

Maszyna, aby być całkowicie uruchomiona, musi zostać zaopatrzona we wszystkie konfiguracje. Stan *Wolna* oznacza możliwość przypisania sesji. Po przypisaniu do sesji, maszyna przechodzi ona w stan oczekiwania na użytkownika. Czas oczekiwania jest konfigurowalny, a po jego upływie przechodzi w stan oczekiwania na wyłączenie. W tym stanie oczekuje na ponowne połączenie, pozwalając użytkownikowi na bezproblemowy powrót do sesji w przypadku nieoczekiwanego utracenia połączenia. Jeżeli użytkownik nie powróci do sesji, to system wyłącza maszynę. Maszyna jest w stanie *Zajęta*, gdy aktualnie pracuje na niej użytkownik. Monitorowanie zajętości realizowane jest przy użyciu odpowiedniej kolejki wiadomości (rozdział 2.1.6, kolejka (IV)).

2.2.2. Serwer wirtualizacji

Serwer wirtualizacji monitoruje zasoby zużywane przez uruchomione na nim maszyny wirtualne oraz fakt stan połączenia użytkowników.



Rysunek 2.4: Diagram stanów serwera wirtualizacji

Przy starcie serwer wirtualizacji oczekuje na działającego w sieci nadzorcę. W przypadku jego braku serwer kończy działanie zwracając błąd. Serwer może mieć wolne zasoby na uruchomienie maszyny lub ich brak. Jednak ważniejszym stanem z perspektywy działania serwera są podłączeni do niego użytkownicy. W przypadku gdy podłączony jest do niego przynajmniej je-

den użytkownik, serwer nie może poprawnie zakończyć pracy aż użytkownik nie skończy używać maszyny.

2.2.3. Użytkownik



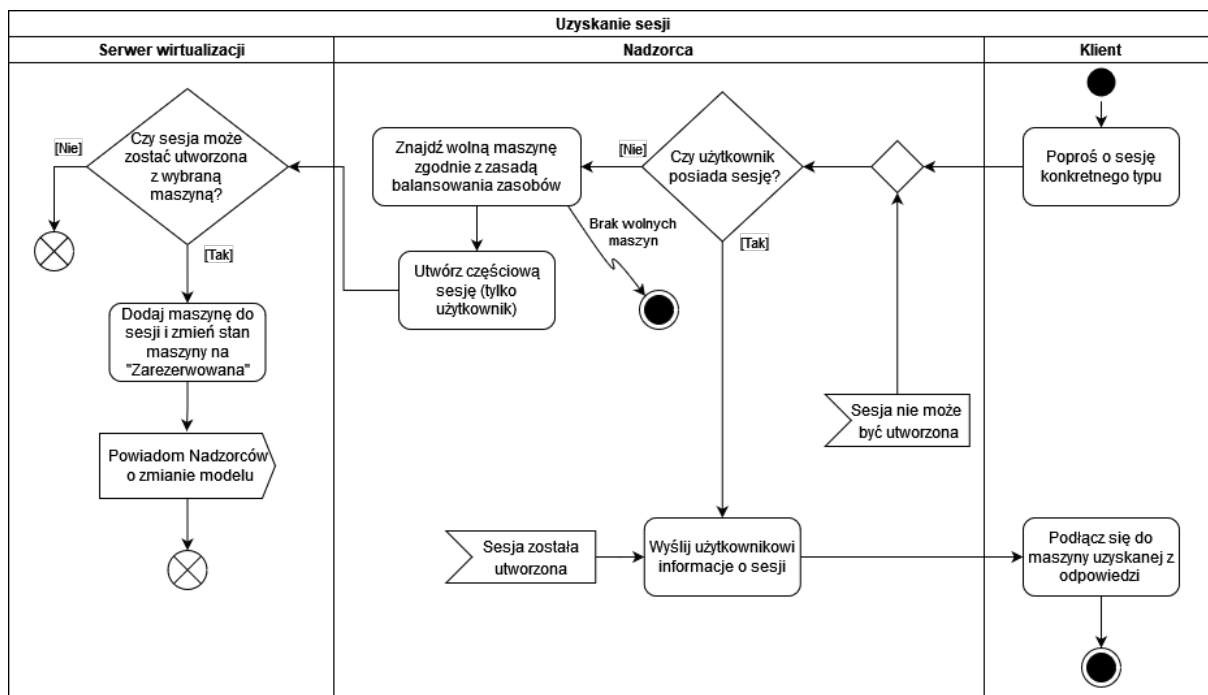
Rysunek 2.5: Diagram stanów użytkownika

Użytkownik z perspektywy systemu po zalogowaniu może być w dwóch stanach: pracuje w ramach swojej sesji lub też nie. W stanie *Połączony* aplikacja kliencka powiadamia serwer wirtualizacji, że ciągle jest obecny. Przy zmianie stanu do innego informowanie musi ustać, aby serwer mógł wykryć odłączenie się użytkownika.

2.3. Procesy biznesowe

2.3.1. Uzyskanie sesji

Proces opisuje prośbę klienta o ustanowienie dla niego sesji. Sesja może już istnieć lub zostać dopiero utworzona.



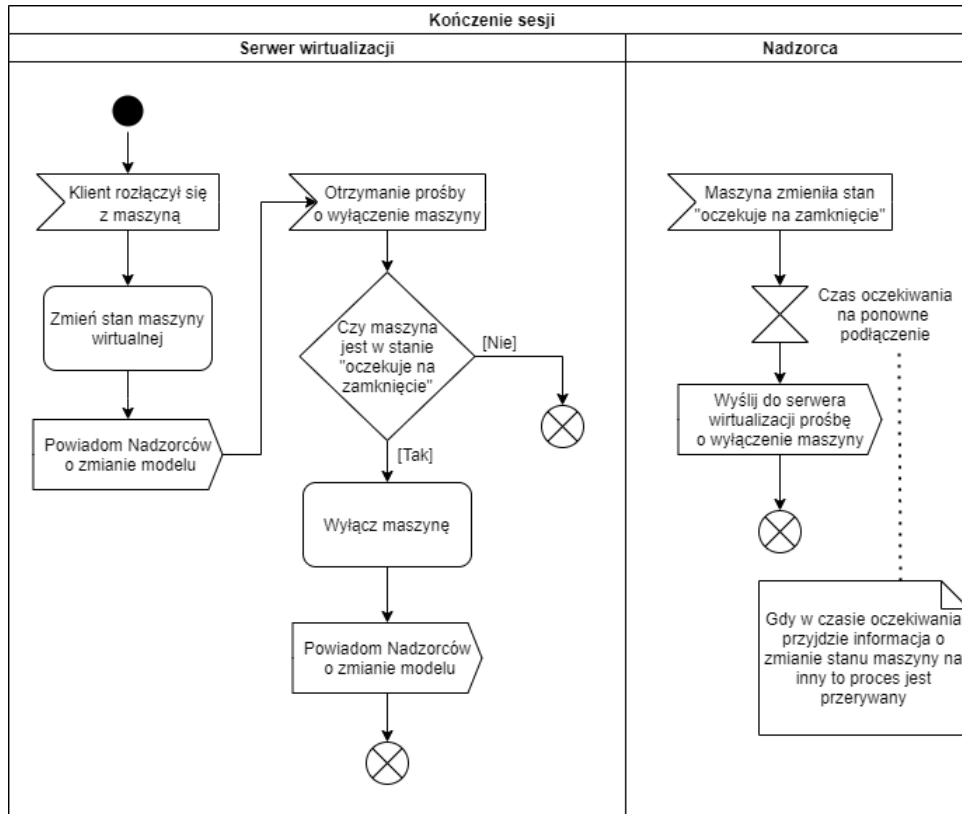
Rysunek 2.6: Proces uzyskania sesji

Jeżeli sesja już istnieje, to zostaje zwrócona użytkownikowi. W przeciwnym razie nadzorca, na podstawie modelu systemu, wybiera pewną wolną maszynę i wysyła do serwera wirtualizacji, na którym działa wybrana maszyna, prośbę o utworzenie sesji. Możliwość działania wielu nadzorców wymaga, aby proces ten był powtarzalny, czyli dla konkretnego stanu modelu musi zawsze zostać wybrana ta sama maszyna. W Przypadku braku wolnej maszyny użytkownikowi zgłoszony jest błąd. Z założenia taka sytuacja może zajść jedynie, gdy wszystkie maszyny zostały zajęte i brakuje zasobów na utworzenie nowych. Wynika to z tego, że system powinien w miarę możliwości trzymać pewien zapas wolnych maszyn. Może się zdarzyć, że model jest nieaktualny i nie można utworzyć sesji z wcześniej wybraną maszyną. Taka prośba zostaje odrzucona przez serwer wirtualizacji, ale odświeżenie modelu spowoduje powtórzenie procesu, tym razem wybierając inną maszynę.

Uzyskanie sesji przez użytkownika zrealizowane jest asynchronicznie. Użytkownik oddzielnym zapytaniem prosi o uzyskanie sesji, po czym używając otrzymanego identyfikatora sesji prosi o jej dane. Obiekt jest w pełni utworzony, gdy odpowiedź nadzorca zawiera sesję w stanie gotowym oraz adres przypisanej maszyny.

2.3.2. Kończenie sesji

Proces ma za zadanie zakończyć sesję oraz wyłączyć skojarzoną z nią wirtualną maszynę w celu zwolnienia zasobów.

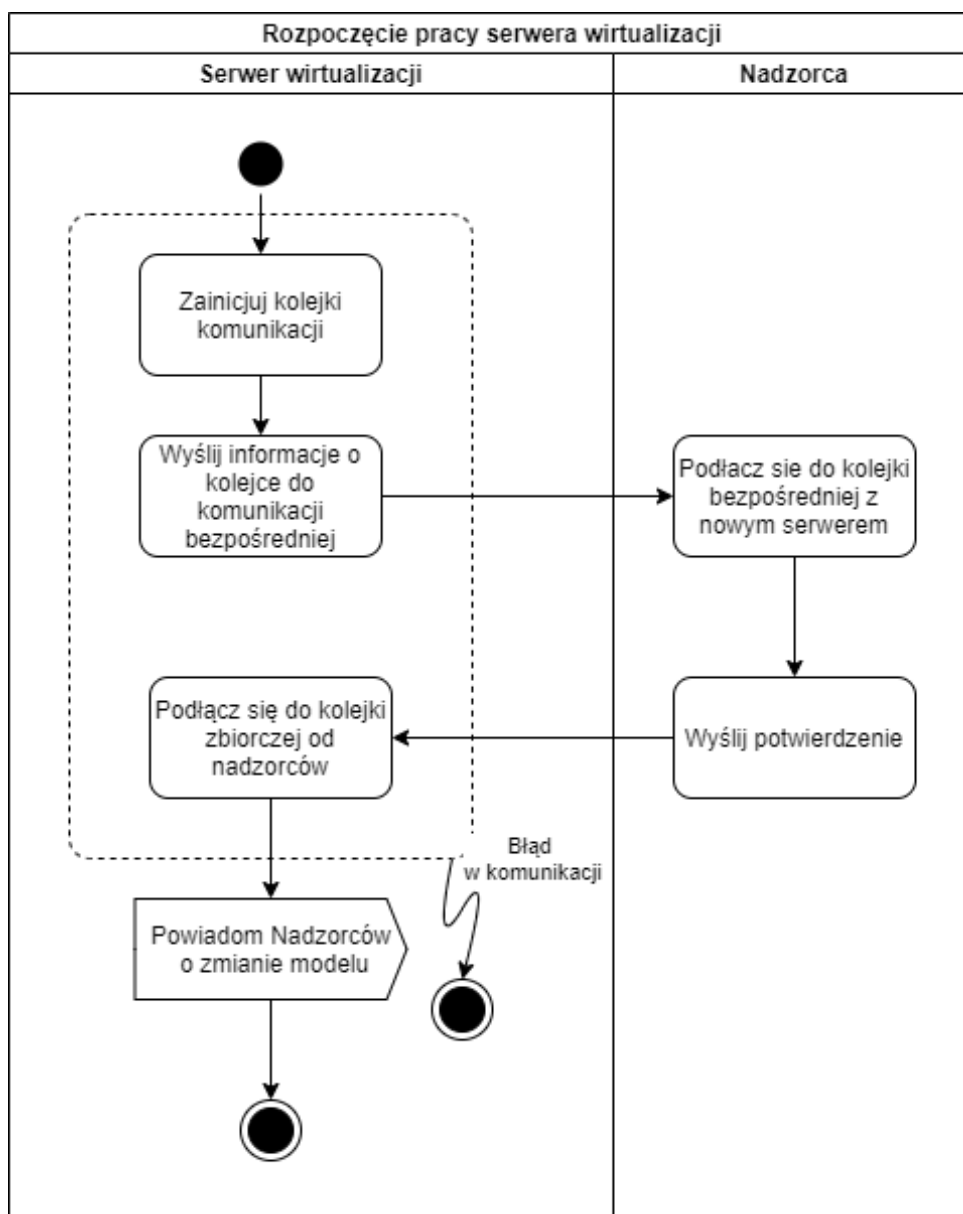


Rysunek 2.7: Proces zakończenia sesji

Proces rozpoczyna się w momencie, gdy użytkownik odłączy się od systemu lub utraci połączenie. Po upływie ustalonego czasu, jeżeli użytkownik nie podłączył się ponownie, maszyna zostaje wyłączona. Serwer wirtualizacji informuje nadzorcę o utracie połączenia lub odłączeniu się użytkownika poprzez zmianę modelu. Decyzję o wyłączeniu maszyny podejmuje nadzorca. Powoduje to, że zmiana konfiguracji aplikacji nadzorczych będzie oznaczać spójną reakcję całego systemu.

2.3.3. Rozpoczęcie pracy serwera wirtualizacji

Proces opisuje przyjęcie nowego serwera wirtualizacji do systemu.



Rysunek 2.8: Proces rozpoczęcia pracy serwera wirtualizacji

Serwer wirtualizacji bez działającego nadzorcy nie jest w stanie obsługiwać użytkowników. Oznacza to, że jeśli przy starcie nie wykryje brokera wiadomości lub nadzorcy po drugiej stronie kolejek [13] to się wyłączy. Jeżeli jednak komunikacja z nadzorcą jest możliwa, to serwer podłączy się do wspólnych kolejek (rozdział 2.1.6, kolejka (I)) oraz prześle informację o kolejce bezpośredniej (rozdział 2.1.6, kolejka (III)). Gdy komunikacja będzie ustanowiona bezwarunkowo wyśle stan swojego modelu do nadzorców.

2.4. Komunikacja

2.4.1. Komunikacja wewnętrzna

Komunikacja wewnątrz systemu opiera się na kolejkach opisanych w rozdziale 2.1.6. W celu uniknięcia wyścigów i utrzymania spójności modelu systemu pomiędzy nadzorcami ustalone zostały następujące zasady:

- Nadzorca może zmienić stan systemu jedynie w reakcji na odpowiedź serwera wirtualizacji. Odpowiedzi te wysyłane są do wszystkich nadzorców, dzięki czemu każdy nadzorca ma taki sam model systemu.
- Wiadomości przetwarzane są przez serwer wirtualizacji w sposób atomowy. Pojedyncza wiadomość musi zostać w pełni obsłużona zanim program przejdzie do obsługi kolejnej.
- Serwer wirtualizacji odpowiada na wiadomości wysyłając nowy stan maszyn. Jeżeli żądanie nie może być spełnione z powodu jego nieaktualności, to serwer na nie nie odpowiada. Wyjątkiem jest żądanie o wysłanie aktualnego stanu maszyn.
- Z powodu asynchroniczności wiadomości moduły nie oczekują na odpowiedź. W przypadku nadzorczy dalsze przetwarzanie zostanie uruchomione przez zmianę modelu.
- Do monitorowania utrzymania połączenia z brokerem użyty jest mechanizm zwracania wiadomości, które nie mogą zostać dostarczone [12]. Używając go nadzorcy mogą wykryć, kiedy poszczególne serwery wirtualizacji przestaną działać, a serwery wirtualizacji - kiedy wszyscy nadzorcy przestaną działać.

Opisane wyżej założenia pozwalają zminimalizować problem hazardu i wyścigów. Jeżeli wiele nadzorców wyśle do serwera wirtualizacji tą samą prośbę (np. o stworzenie sesji na konkretnej maszynie), to z atomowości obsługi sesja zostanie stworzona tylko dla pierwszego z nich. Serwer wirtualizacji wyśle wiadomość o aktualizacji stanu maszyn i zignoruje pozostałe prośby. Nadzorcy otrzymają zmianę stanów, co spowoduje wywołanie odpowiednich procedur. Dla pierwszego będzie to dalsza część procesu tworzenia sesji, a pozostali nadzorcy pozostaną w procesie wyszukiwania maszyny do sesji.

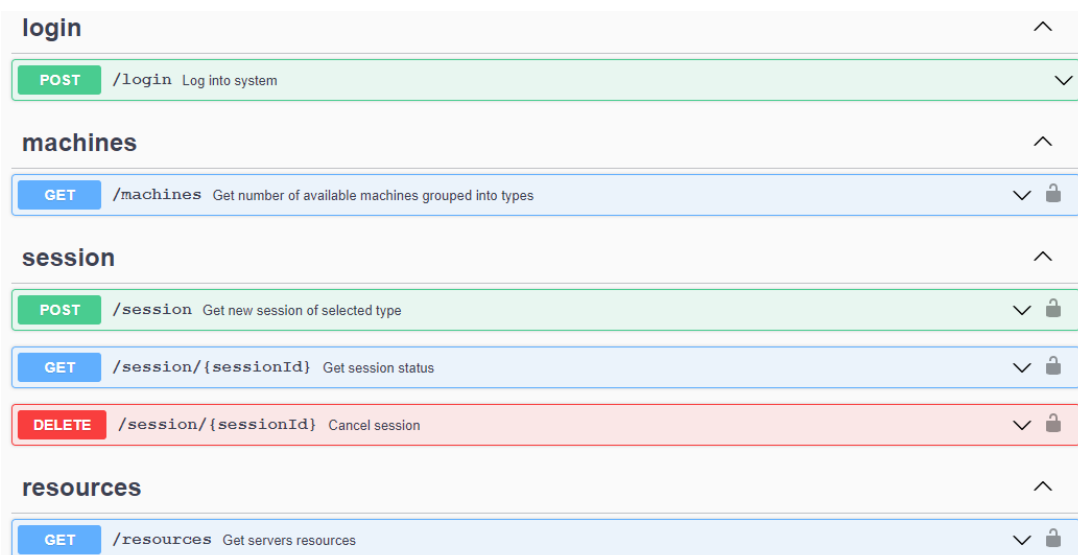
2.4.2. Komunikacja zewnętrzna

Komunikacja aplikacji klienckiej oraz panelu administratora z systemem - nadzorcą - rozwiązana jest za pomocą REST API (<https://restfulapi.net/>). W zależności od konfiguracji

2.4. KOMUNIKACJA

nadzorcy wiadomości mogą być wysyłane za pomocą protokołu HTTPS, który zapewnia ich szyfrowanie. W tym celu wymagane jest aby na adres, pod którym udostępniony będzie system, wystawiony był odpowiedni certyfikat [15] gwarantujący jego tożsamość.

Całość specyfikacji API umieszczona jest w załączniku. Poniżej znajduje się zestawienie oraz krótki opis dostępnych endpointów.



login		
POST	/login	Log into system
machines		
GET	/machines	Get number of available machines grouped into types
session		
POST	/session	Get new session of selected type
GET	/session/{sessionId}	Get session status
DELETE	/session/{sessionId}	Cancel session
resources		
GET	/resources	Get servers resources

Rysunek 2.9: Endpointy API

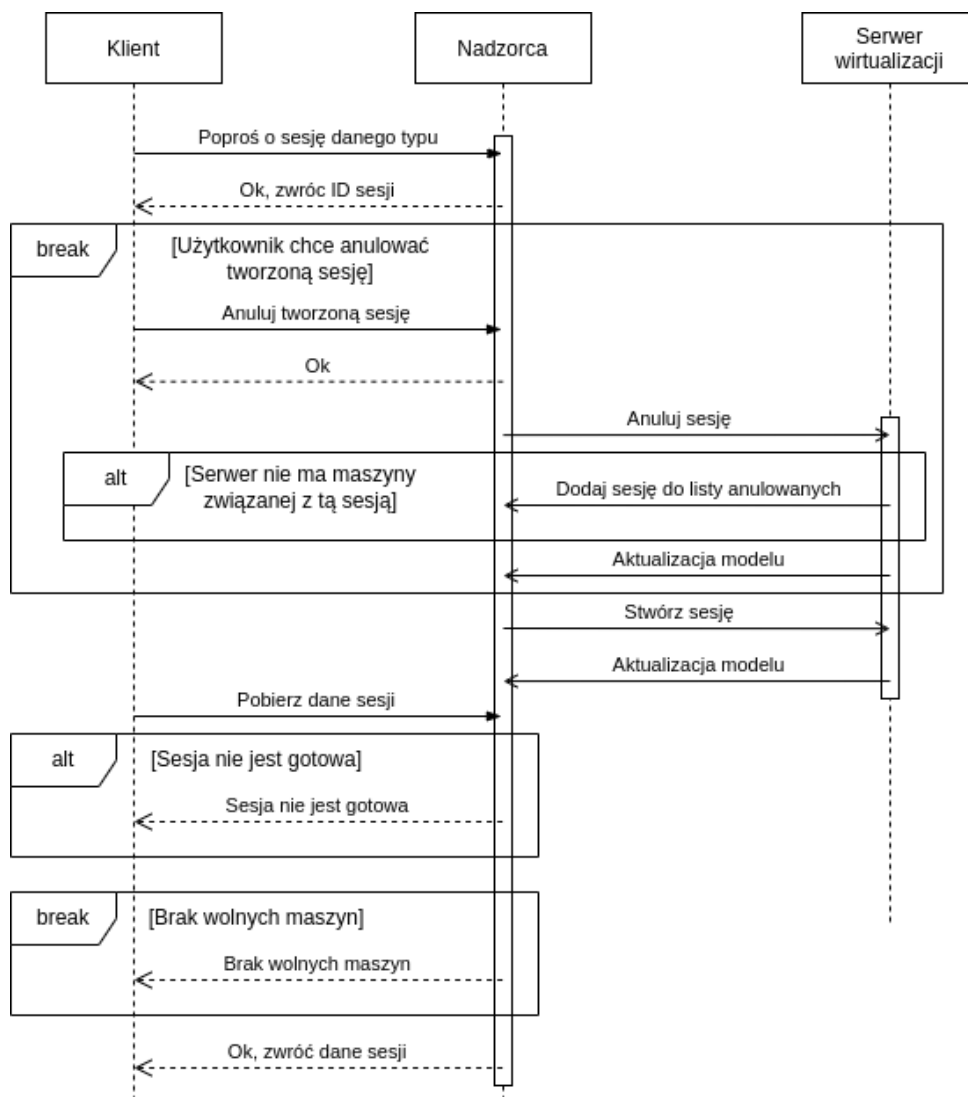
- Login - służy do logowania do systemu; współdzielony przez aplikację kliencką oraz panel administracyjny. Poprawne zalogowanie zwraca token do dalszej autoryzacji.
- Machines - służy do pobierania przez aplikację informacji o typach i ilości dostępnych maszyn. Wymaga autoryzacji poprzez umieszczenie tokenu otrzymanego podczas logowania w odpowiednim nagłówku wiadomości. Dostępny tylko dla użytkowników.
- Session - pozwala na wysłanie prośby o uzyskanie sesji, pobranie stanu sesji oraz jej anulowanie. Utworzenie sesji jest możliwe poprzez operację **POST** z typem maszyny. W odpowiedzi użytkownik dostaje częściowo wypełniony obiekt sesji zawierający id umożliwiające dalsze zapytania. Operacja **GET** zwraca obiekt sesji z aktualnym stanem. Jeżeli sesja jest gotowa, to zawiera on też adres, z którym należy nawiązać połączenie RDP. **DELETE** umożliwia anulowanie sesji. Wymaga autoryzacji poprzez umieszczenie tokenu otrzymanego podczas logowania w odpowiednim nagłówku wiadomości. Dostępny tylko dla użytkowników.
- Resources - udostępnia informację o zasobach działających serwerów wirtualizacji. Dostępny jedynie dla administratora. Wymaga autoryzacji poprzez umieszczenie tokenu otrzymanego podczas logowania w odpowiednim nagłówku wiadomości.

Ważną informacją jaką musi posiadać system jest fakt, czy użytkownik rzeczywiście jest podłączony do maszyny wirtualnej. System uzyskuje tę informację komunikując się z brokerem wiadomości odpowiedzialnym za komunikację z użytkownikami. Każda aplikacja kliencka, po uzyskaniu gotowej sesji, tworzy kolejkę o takiej nazwie jak uzyskany identyfikator sesji. Serwer wirtualizacji sprawdza co jakiś czas czy na końcu kolejki istnieje jakikolwiek konsument. Gdy użytkownik się rozłączy to aplikacja kliencka usuwa kolejkę, co pozwala serwerowi wykryć zakończenie pracy.

2.5. Sekwencje komunikacji

2.5.1. Utworzenie sesji

Celem tej sekwencji komunikacji jest odnalezienie istniejącej już sesji lub stworzenie nowej. Zakładamy, że w systemie istnieje wolna maszyna. W przeciwnym przypadku zgłaszamy użytkownikowi błąd.



Rysunek 2.10: Sekwencja komunikacji utworzenia sesji

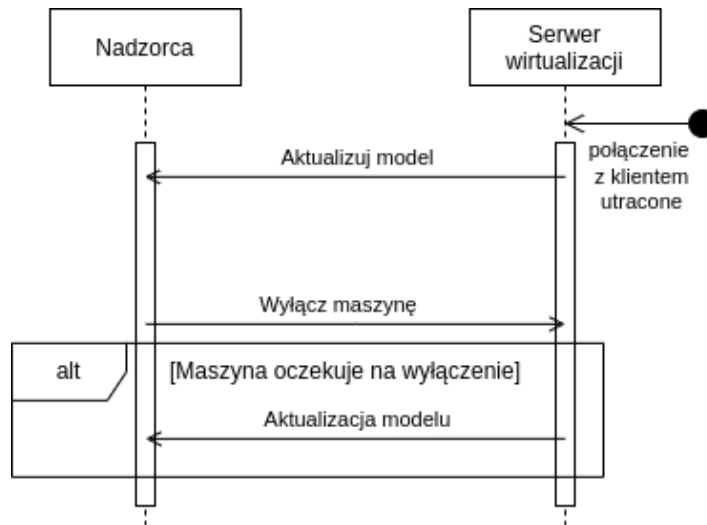
Po prośbie użytkownika nadzorca znajduje wolną maszynę i prosi odpowiadający serwer wirtualizacji, aby spróbował utworzyć z nią sesję dla użytkownika. W przypadku sukcesu serwer wysyła informację o zmianie modelu. W przeciwnym przypadku nadzorca powtarza wyszukiwanie wolnej maszyny. Stan otrzymanej maszyny decyduje, czy nadzorca musi powtórzyć wyszukiwanie (m.in. czy sesja do niej przypisana należy do tego użytkownika).

Może się zdarzyć, że użytkownik anuluje wyszukiwanie. Jeżeli maszyna jest już przydzielona użytkownikowi, to serwer wirtualizacji jest powiadamiany o anulowaniu sesji. Jeżeli nie została jeszcze utworzona, to nadzorca przerywa wyszukiwanie lub wyłącza ją.

2.5.2. Zakończenie sesji

Sekwencja ta zainicjowana jest poprzez utracenie połączenia z użytkownikiem maszyny. Serwer powiadamia o tym fakcie nadzorców, po czym oczekuje na polecenie wyłączenia maszyny

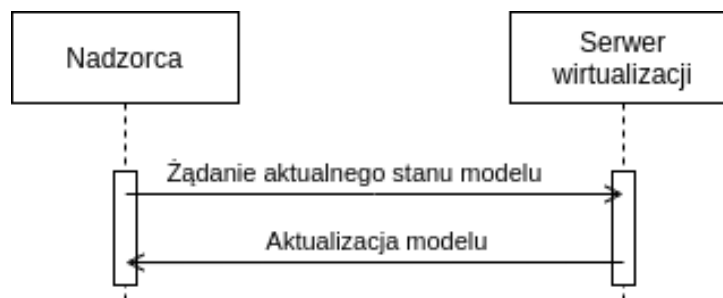
przesłane przez nadzorcę. Serwer może odmówić z powodu różnic modelu, lub jeżeli maszyna znów jest używana przez użytkownika, ignorując wiadomość.



Rysunek 2.11: Sekwencja komunikacji zakończenia sesji

2.5.3. Aktualizacja stanu

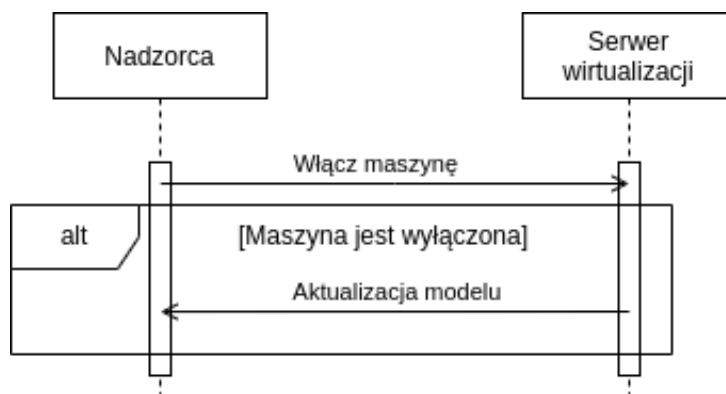
Nadzorca może w każdej chwili poprosić wszystkie serwery wirtualizacji o przestanie ich aktualnego stanu poprzez wspólną kolejkę do serwerów wirtualizacji (rozdział 2.1.6, kolejka (I)). Serwery muszą bezwarunkowo odpowiedzieć aktualnym stanem do wspólnej kolejki zwrotnej (rozdział 2.1.6, kolejka (II)).



Rysunek 2.12: Sekwencja komunikacji aktualizacji stanu systemu

2.5.4. Włączenie maszyny

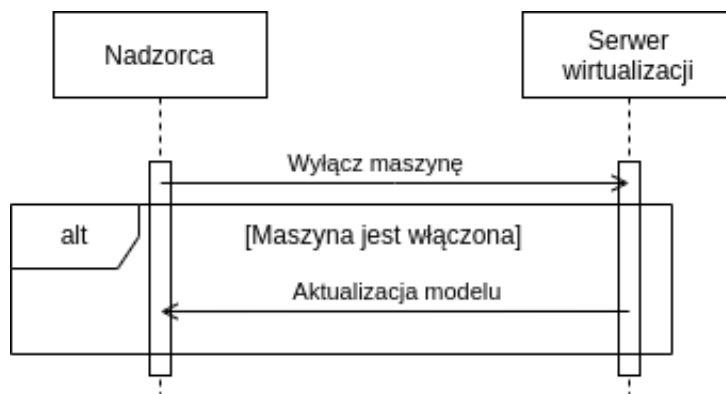
Nadzorca może poprosić konkretny serwer wirtualizacji, aby utworzył maszynę o konkretnej nazwie. Jeżeli maszyna nie istnieje, to zostanie uruchomiona a serwer odeśle powiadomienie o zmianie modelu zbiorczą kolejką (rozdział 2.1.6, kolejka (II)). W przeciwnym wypadku nie zrobi nic.



Rysunek 2.13: Sekwencja komunikacji włączenia maszyny

2.5.5. Wyłączenie maszyny

Nadzorca może poprosić konkretny serwer wirtualizacji, aby wyłączył konkretną maszynę wirtualną. Jeżeli maszynę można wyłączyć to zostanie ona wyłączona. Następnie serwer wirtualizacji odeśle powiadomienie o zmianie modelu zbiorczą kolejką (rozdział 2.1.6, kolejka (II)). W przeciwnym wypadku nie zrobi nic.



Rysunek 2.14: Sekwencja komunikacji wyłączenia maszyny

2.6. Wykorzystane technologie

2.6.1. Technologie realizacji strony klienckiej

Realizacja panelu administracyjnego oraz aplikacji klienckiej, przez które użytkownicy i administratorzy korzystają z systemu, zostały wykonane przy użyciu platformy programistycznej Angular (<https://angular.io/>). Wykorzystaliśmy ją z powodu znajomości technologii, co umożliwiło sprawną implementację aplikacji. W każdym przypadku skorzystaliśmy z języka TypeScript (<https://www.typescriptlang.org/>), który jest wykorzystywany

przez Angulara oraz kompiluje się do JavaScriptu (<https://www.ecma-international.org/publications-and-standards/standards/ecma-262/>).

Aplikacja kliencka uruchamiana jest na systemie operacyjnym użytkownika poprzez technologię Electron (<https://www.electronjs.org/>), która umożliwia wyświetlanie aplikacji internetowej w postaci okna. Electron korzysta z silnika Chromium w celu uruchomienia aplikacji webowej jako aplikacji okienkowej. Uprościło to tworzenie aplikacji na wiele systemów operacyjnych. Dodatkowym plusem korzystania z Electrona jest możliwość odwołania się do silnika NodeJS (<https://nodejs.org/en/>) w celu dostępu do zasobów systemu operacyjnego. Wykorzystaliśmy go do uruchamiania podprocesu klienta RDP, aby po uzyskaniu sesji od razu zestawić połączenie z wybraną maszyną wirtualną.

Aby ułatwić implementację API (rozdział 2.4.2)) do komunikacji pomiędzy aplikacjami zewnętrznymi oraz nadzorcą, skorzystaliśmy z generatora kodu OpenAPI Generator (<https://openapi-generator.tech/>). Dzięki niemu generowaliśmy gotowe szablony klas do komunikacji ze strony klienta oraz nadzorcy. Zminimalizowało to błędy związane z implementacją API.

2.6.2. Technologie realizacji strony serwerowej

W przypadku aplikacji nadzorcy oraz serwera wirtualizacji zdecydowaliśmy się na środowisko uruchomieniowe .NET 5.0 (<https://dotnet.microsoft.com/en-us/>) z językiem C# (<https://docs.microsoft.com/en-us/dotnet/csharp/>). Ekosystem .NET jest przez nas dobrze znany co przyspiesza naszą pracę. Głównym ułatwieniem była duża baza bibliotek zgromadzona w menadżerze pakietów Nuget (<https://www.nuget.org/>), do którego dodaliśmy też własne pakiety, aby wydzielić wspólną część kodu aplikacji serwerowych.

Do zarządzania i monitorowania maszyn wirtualnych skorzystaliśmy z libvirta (<https://libvirt.org/>). W serwerze wirtualizacji wykorzystaliśmy rozszerzoną przez nas bibliotekę odwołującą się do oryginalnych bibliotek libvirta. Jest to technologia powszechnie znana i dobrze przez nas poznana. Był to najprostszy sposób do realizacji zarządzania maszynami wirtualnymi. Libvirt umożliwia współpracę z technologią Vagrant (<https://www.vagrantup.com/>), która pozwala na proste powielanie maszyn wirtualnych z wzorca dostarczonego przez użytkownika. Ułatwia to administratorom tworzenie obrazów uruchamianych systemów operacyjnych.

Serwer wirtualizacji komunikuje się z Vagrantem poprzez specjalnie przygotowany plik wykonywalny z kodem w Ruby (<https://www.ruby-lang.org/en/>). Vagrant wykonywany jest jako podproces serwera wirtualizacji w postaci poleceń powłoki. Parametry przekazywane są poprzez zmienne środowiskowe ustawiane przez serwer wirtualizacji chwile przed uruchomieniem procesu. Do dodatkowych konfiguracji maszyn wirtualnych użyliśmy technologii Ansible

(<https://www.ansible.com/>). Wybraliśmy ją, ponieważ jest znacznie mniej bezawaryjna niż konfigurowanie systemu przez skrypty powłoki oraz posiada predefiniowane akcje obudowujące standardowe wywołania systemowe. Skorzystaliśmy z niej również do konfiguracji komputerów przed uruchomieniem systemu.

Do komunikacji pomiędzy modułami po stronie serwerowej skorzystaliśmy z asynchronicznego brokera wiadomości RabbitMQ (<https://www.rabbitmq.com/>). Umożliwił nam on bezproblemową implementację komunikacji pomiędzy modułami nadzorcy i serwera wirtualizacji.

2.6.3. Technologie testowania

Do testów jednostkowych aplikacji od strony klienckiej skorzystaliśmy z platformy Jest (<https://jestjs.io/>). Ułatwia ona pisanie testów przy niepełnym obrazie systemu. Przy rozbudowanej komunikacji w naszym systemie było to bardzo przydatne. W przypadku aplikacji serwerowych skorzystaliśmy z platformy NUnit (<https://nunit.org/>). Zналиśmy ją z wcześniejszych projektów, więc mogliśmy od razu przystąpić do pracy. Do testów komunikacji poprzez brokera wiadomości oraz integracji z libvirtem także skorzystaliśmy z NUnita.

Do testów integracyjnych aplikacji Angularowych korzystaliśmy z platformy Cypress (<https://www.cypress.io/>). Zdecydowaliśmy się na nią, ponieważ wykonuje ona zrzut widoku podczas każdego kroku testów. Wcześniej znane nam rozwiązania nie posiadały takich funkcji, przez co bardzo trudno było szukać problemów w testach. Dodatkowe testy API wystawianego przez aplikację nadzorcy wykonaliśmy Postmanem (<https://www.postman.com/>).

2.6.4. Utrzymanie kodu

Do sprawnego zarządzania projektem wykorzystaliśmy strukturę wielu repozytoriów dostępnych na stronie github. Wspólne części kodu wydzielaliśmy w postaci pakietów NPM (<https://www.npmjs.com/>) oraz Nuget. Aby zautomatyzować proces publikacji pakietów skorzystaliśmy ze środowiska ciągłej integracji Github Actions (<https://github.com/features/actions>). Umożliwiło to praktycznie bezproblemową realizację komunikacji pomiędzy nadzorcą, a aplikacją kliencką i panelem administratora. Tworzyliśmy automatycznie generowane pakiety z częściową implementacją obiektów transportowych oraz struktury zapytań.

2.7. Wymagania systemu

System do poprawnej pracy wymaga konfiguracji środowiska na wielu płaszczyznach. Poniższy rozdział zawiera informacje o konfiguracji systemu wymaganej do uruchomienia każdego z

modułów.

2.7.1. Komunikacja sieciowa między modułami

System do poprawnego działania z pełną funkcjonalnością musi składać się z:

- przynajmniej jednego nadzorcy (*),
- serwerów wirtualizacji,
- maszyn wirtualnych uruchamianych na serwerach wirtualizacji,
- serwera HTTP udostępniającego panel administracyjny,
- brokera wiadomości do komunikacji wewnętrznej (*),
- brokera wiadomości do komunikacji zewnętrznej (może być tym samym brokerem co wewnętrzny),
- dowolnej liczby aplikacji klienckich,
- systemu katalogowego przechowującego użytkowników
- dysku sieciowego zawierającego katalogi domowe użytkowników

Elementy wymagane do poprawnego uruchomienia pozostałych modułów oznaczone zostały symbolem (*).

Poniżej opisane zostały wymagania osiągalności połączeń sieciowych dla poszczególnych modułów.

Dostępność wewnętrznego brokera

Broker wewnętrzny powinien być dostępny dla każdego nadzorcy oraz każdego serwera wirtualizacji. Jest to wymagane do prawidłowej komunikacji pomiędzy nadzorcami i serwerami wirtualizacji.

Dostępność zewnętrznego brokera

Broker zewnętrzny powinien być osiągalny przez każdy serwer wirtualizacji oraz każdego klienta łączącego się z systemem. Jest to wymagane do stwierdzenia czy użytkownik nadal jest podłączony do maszyny wirtualnej.

Dostępność nadzorców

Nadzorcy powinni być widoczni przez aplikacje klienckie, które poprzez nich komunikują się z resztą systemu. Administrator podczas używania panelu administracyjnego z poziomu serwera HTTP powinien móc wysyłać zapytania do nadzorców.

Dostępność serwerów wirtualizacji

Serwery wirtualizacji nie muszą być widoczne przez żaden z innych modułów.

Dostępność serwera http z panelem administracyjnym

Serwer HTTP powinien być dostępny dla każdego z administratorów.

Dostępność maszyn wirtualnych

Maszyny wirtualne powinny być osiągalne przez każdego z klientów. Jest to potrzebne do pracy na nich poprzez protokół RDP.

Dostępność zewnętrznych modułów

System katalogowy powinien być dostępny dla każdego nadzorcy oraz maszyny wirtualnej w celu autoryzacji. Dysk sieciowy powinien być widoczny przez maszyny wirtualne, aby mogły używać katalogów domowych użytkowników.

2.7.2. Wymagania aplikacji klienckiej

Aplikacja kliencka wspiera system Windows 10 lub nowszy oraz GNU/Linux w dystrybucjach opartych na 64 bitowych systemach Ubuntu 18.04+ oraz Arch Linux.

Dla systemu Windows aplikacja kliencka, jako plik wykonywalny pobrany ze strony z oficjalnymi wydaniem [9], powinna uruchomić się bez żadnych wcześniejszych konfiguracji. Do uruchomienia wymagany jest jednak plik konfiguracji użytkownika dostępny w oficjalnych wydaniach. Do działania korzysta ona z klienta RDP dostarczonego przez firmę Microsoft wraz z systemem Windows. Aplikacja była testowana dla systemu Windows 10. Dla starszych wersji systemu Windows aplikacja może nie działać prawidłowo.

W przypadku systemu Linux należy posiadać środowisko graficzne oraz mieć zainstalowanego klienta FreeRDP (<https://www.freerdp.com/>). Pozostałe zależności są dostarczone wewnątrz pliku `.appimage`. Działanie aplikacji było testowane na dystrybucjach opartych na Debianie oraz Arch Linuxie. Dla innych dystrybucji aplikacja może nie działać prawidłowo.

2.7.3. Wymagania aplikacji nadzorcy i serwera panelu administracyjnego

Wymagania sprzętowe

Aplikacja nadzorcy i serwer panelu administracyjnego może być uruchomiony na dowolnym komputerze z możliwością uruchomienia usługi Dockera. System operacyjny, pod którym pracuje komputer, nie ma znaczenia.

Wymagania systemowe

Do uruchomienia aplikacji nadzorcy i serwera HTTP z panelem administracyjnym potrzeba zainstalowanego Dockera w systemie. Każdy z tych dwóch modułów można zbudować do kontenera, w którym wszystkie zależności zostaną spełnione.

Do ręcznego uruchomienia aplikacji nadzorcy, bez pośrednictwa Dockera, wymagany jest .NET 5. W przypadku panelu administratora do kompilacji wymagany jest NodeJS w wersji 16.13.2. Zbudowany moduł można udostępnić za pomocą dowolnego serwera HTTP.

2.7.4. Wymagania serwera wirtualizacji

Wymagania sprzętowe

Serwer wirtualizacji zadziała prawidłowo na komputerze działającym pod kontrolą systemu operacyjnego z grupy GNU/Linux. Maszyna musi być podłączona do sieci poprzez przewodowy interfejs sieciowy (jest to wymagane przez interfejs typu bridge). Wydajnościowo system musi być w stanie uruchomić wirtualne maszyny z usługą Dockera jednocześnie. Jednak zalecamy aby system był wyposażony przynajmniej w dwurdzeniowy procesor, 4GB pamięci operacyjnej oraz przynajmniej 100GB przestrzeni dyskowej. Jeżeli system spełnia takie wymagania, to powinien być w stanie uruchomić maszynę wirtualną pod kontrolą systemu bez przekazanego urządzenia PCI.

W razie chęci wykorzystania funkcjonalności PCI Passthrough [19], aby przekazać urządzenia do maszyny wirtualnej komputer musi spełniać dodatkowe wymagania. Po aktywowaniu modułu IOMMU (I/O Memory Management Unit) [3] każde z urządzeń, które chcemy przekazać, musi znajdować się w innej grupie IOMMU. Prawidłowa izolacja urządzeń przekazywanych do maszyny wirtualnej od pozostałych urządzeń uruchamianych na serwerze jest kluczowa z punktu bezpieczeństwa.

To czy prawidłowa izolacja urządzeń jest możliwa zależy od używanej w komputerze płyty głównej. Większość chipsetów przeznaczonych do komputerów biurowych używa tych samych linii komunikacji dla wielu urządzeń. Należy więc skorzystać z płyt przeznaczonych do procesorów

rów dla entuzjastów (np. linia AMD Threadripper z chipsetami z linii X) albo profesjonalnych (np. linia AMD Epyc). Posiadają one wiele linii PCI, które zwykle nie są współdzielone między urządzeniami.

Alternatywnym wyborem jest modyfikacja jądra systemu i dodanie ACS Override patch [24]. Spowoduje to rozdzielenie wspólnych grup IOMMU na oddzielne wirtualne grupy. Jednakże z tak skonfigurowaną maszyną istnieje niebezpieczeństwo wykonania wyłomu poza wirtualną maszynę oraz zdestabilizowania pracy systemu. Odradzamy wykorzystywanie takiego rozwiązania w przypadku publicznego dostępu do systemu. W przypadku braku odpowiedniego sprzętu oraz ograniczenia dostępu do systemu można spróbować z niego skorzystać.

Wymagania systemowe

Serwer wirtualizacji, oprócz działającej usługi Dockera, wymaga usługi zarządcy maszyn wirtualnych libvirt w wersji 7.10+. W przypadku ręcznego uruchamiania aplikacji, bez pośrednictwa Dockera, wymagany jest Vagrant w wersji 2.2.11+ oraz .NET 5.

Aby prawidłowo uruchomić maszynę wirtualną potrzebna jest uruchomiona usługa zapory sieciowej oraz pakiet `dnsmasq`. Wymagane jest także utworzenie struktur usługi Vagrant dla użytkownika uruchamiającego serwer wirtualizacji. Chodzi konkretnie o utworzenie folderu `.vagrant.d` w katalogu domowym użytkownika.

Aby uruchamiane maszyny wirtualne były dostępne dla innych urządzeń potrzebne jest utworzenie interfejsu sieciowego w trybie bridge. Nazwa interfejsu powinna być przekazana do pliku konfiguracyjnego serwera wirtualizacji. Maszyny wirtualne uzyskają wtedy dostęp do sieci w taki sposób, jakby były fizycznymi komputerami w sieci.

W niektórych przypadkach, podczas uruchamiania maszyny wirtualnej przy użyciu Vagranta z działającym Dockerem, komunikacja sieciowa z maszyną wirtualną poprzez podłączony interfejs w trybie bridge może zostać ograniczona. Należy wtedy dopilnować, by w trakcie działania systemu zapora sieciowa posiadała zasadę na samej górze łańcucha **FORWARD**, która będzie bezwarunkowo zezwalać trasować pakiety z urządzeń w trybie bridge. Przy uruchomieniu serwera wirtualizacji w kontenerze dockerowym trzeba dopilnować aby z sieci, w której pracuje kontener, można było osiągnąć sieć konfiguracyjną Vagranta. W przeciwnym wypadku próba uruchomienia maszyny wirtualnej zakończy się z błędem połączenia SSH.

2.7.5. Automatyzacja konfiguracji

Przykładowa konfiguracja oraz narzędzia do automatycznej konfiguracji przed uruchomieniem dostępne są w module `configuration` [8]. Składa się on ze skryptów konfiguracyjnych Ansible,

nazywanych dalej *playbook*, oraz zmiennych opisujących konfigurowane komputery.

By uruchomić skrypt dla pewnego systemu operacyjnego, który chcemy skonfigurować, musi on spełniać wymogi opisane w dokumentacji Ansible [22].

Grupy i zmienne

Konfiguracja podzielona jest na 2 grupy: *overseer* i *virtsrv*. Odpowiadają one za reprezentację systemów przygotowanych odpowiednio dla nadzorców oraz serwerów wirtualizacji. Dodatkowo wytyczona jest sztuczna grupa *all* opisująca wszystkie konfigurowane systemy.

Jedyną wspólną zmienną dla wszystkich maszyn jest nazwa użytkownika, który będzie uruchamiał i odpowiadał za zasoby systemu OneClickDesktop.

Dla każdej maszyny z osobna należy zdefiniować dane dostępowe do komunikacji z konfigurowanym systemem. Dodatkowo należy podać hasło umożliwiające dostęp do uprawnień superużytkownika.

Zmienne dla serwera wirtualizacji

Playbook dla serwera wirtualizacji wykona wszystkie kroki opisane w rozdziale 2.7.4. Aby tego dokonać należy zdefiniować dane dla tworzonego interfejsu typu *bridge*. Należy podać nazwę interfejsu sieciowego, który zostanie połączony do nowo tworzonego urządzenia *bridge* o nazwie zawartej w pliku. Playbook korzysta z NetworkManagera (<https://networkmanager.dev/>) do zmiany konfiguracji, zatem trzeba przekazać nazwę *connection* (jednostka logiczna w Network-Managerze) skojarzonego z początkowo wybranym interfejsem sieciowym.

Zmienne dla nadzorcy

Aby uruchomić nadzorcę wystarczą wspólne wymagania dla wszystkich grup.

2.7.6. Wymagania szablonu maszyny wirtualnej

Maszyna wirtualna uruchamiana w ramach systemu OneClickDesktop musi być dostarczona w postaci Vagrant Boxa. Przykładowy box został utworzony w czasie rozwoju systemu i dostępny jest w chmurze Vagrant pod nazwą *smogork/archlinux-rdp* [6]. Box używa dystrybucji Arch Linux oraz spełnia wszystkie wymagania aby zostać uruchomionym w ramach systemu.

Do przygotowania szablonu proponujemy, aby skorzystać właśnie z tego obrazu. Jeżeli jednak potrzebne jest utworzenie niestandardowego szablonu to musi on:

- Spełniać minimalne wymagania opisane w dokumentacji Vagranta [16].

2.8. URUCHOMIENIE SYSTEMU

- Mieć zainstalowany menadżer okienek. Proponowany menadżer okienek, zawarty w przykładowym szablonie, to XFCE w wersji 4.16. (<https://xfce.org/>)
- Przy uruchomieniu udostępniać usługę zdalnego pulpitu RDP. Przykładowy szablon korzysta z implementacji xrdp w wersji 0.9.17-1
- Każdy nowy interfejs sieciowy powinien być tak skonfigurowany, aby uzyskać adres IP z usługi DHCP.
- Przy starcie systemu uruchamiać usługę `qemu-guest-agent`.
- Autoryzować użytkowników za pomocą systemu katalogowego [10]. Dokładne dane do ustanowienia połączenia mogą być przekazane za pomocą playbooka.
- Montować katalogi domowe użytkowników. Może to być również rozwiązane za pomocą playbooka.

Nie ma ograniczenia co do systemu operacyjnego uruchamianego wewnątrz systemu OneClick-Desktop, tak długo jak jest on w stanie spełnić powyższe wymagania.

Aby zbudować własny szablon należy skonsultować się z dokumentacją wtyczki vagrant-libvirt [20].

2.8. Uruchomienie systemu

W tym podrozdziale zakładamy, że każdy system operacyjny został skonfigurowany poprawnie zgodnie z opisem w rozdziale 2.7.

2.8.1. Pozyskanie aplikacji klienckiej

Zalecany sposób pozyskania aplikacji klienckiej jest udanie się do sekcji z oficjalnymi wydaniem [9] modułu aplikacji klienckiej oraz pobranie najnowszej wersji dla wybranego systemu operacyjnego. Przy pierwszym uruchomieniu trzeba pobrać archiwum plików zawierające aplikację oraz plik konfiguracyjny. Bez pliku konfiguracyjnego aplikacja nie uruchomi się.

2.8.2. Budowanie kontenerów

Moduły: panelu administracyjnego, nadzorca i serwera wirtualizacji można uruchomić pod postacią kontenera dockerowego. Jest to zalecany sposób uruchamiania tych trzech modułów.

Ujednolicony sposób budowania kontenerów

Każdy z tych 3 modułów ma przygotowany skrypt `build.sh`, który powinien prawidłowo zbudować kontener. Skrypt ten wykona polecenie do budowania i oznaczy kontener odpowiednią nazwą. Należy wykonać go zawsze z poziomu głównego folderu repozytorium modułu. Każdy kontener przy starcie wykona skrypt `assets/entry_point.sh`.

Budowanie kontenera serwera wirtualizacji

Kontener serwera wirtualizacji wymaga specjalnie przygotowanego wcześniej kontenera zawierającego wszystkie potrzebne zależności do uruchomienia aplikacji. Aby go zbudować należy skorzystać z pliku `runtime_container/Dockerfile` i oznaczyć go nazwą `click-desktop/virtualization-server-runtime`. Kontener zawierający aplikację w czasie budowania będzie oczekiwał, że taki obraz istnieje.

Po zbudowaniu kontenera z zależnościami można przystąpić do budowania kontenera głównego. Należy do tego celu skorzystać z dostarczonego pliku `Dockerfile` w głównym folderze repozytorium.

Przy uruchomieniu skryptu `build.sh` kontener jest budowany z nazwą `one-click-desktop/virtualization-server`.

Budowanie kontenera nadzorcy i panelu administratora

W przypadku tych dwóch modułów nie trzeba wykonać żadnych dodatkowych przygotowań. Wystarczy skorzystać z dostarczonego pliku `Dockerfile` w głównym folderze repozytorium.

Przy uruchomieniu skryptu `build.sh` kontenery są budowane z nazwami odpowiednio `one-click-desktop/overseer` oraz `one-click-desktop/admin-panel`.

2.8.3. Budowanie modułów

Każdy z modułów posiada dokładną instrukcję budowania w pliku `README.md`. Uruchamianie zbudowanych modułów zaleca się tylko w przypadku rozwoju aplikacji. Przy wdrażaniu systemu najszybciej i najbezpieczniej jest skorzystać z metod opisanych w sekcjach 2.8.1 i 2.8.2.

Moduły napisane w technologii .NET

W przypadku budowania modułu nadzorcy i serwera wirtualizacji potrzebne są narzędzia deweloperskie .NET 5.0. W trakcie budowania aplikacji wszystkie użyte biblioteki zostaną pobrane przy użyciu menadżera pakietów Nuget. Przy uruchomieniu serwera wirtualizacji potrzebny jest,

poza wymaganiami zdefiniowanymi w rozdziale 2.7.4, zainstalowany Vagrant wraz z wtyczką vagrant-libvirt w wersji przynajmniej 0.7.0.

Moduły napisane w technologii Node

W przypadku budowania modułów aplikacji klienckiej oraz panelu administracyjnego potrzebny będzie pakiet Node. Przed zbudowaniem aplikacji należy pobrać wszystkie wymagane biblioteki przy użyciu menadżera pakietów npm poleceniem `npm install`.

2.8.4. Konfiguracja aplikacji klienckiej

Aplikacja kliencka wymaga pliku konfiguracyjnego o nazwie `config.json` w tym samym folderze co plik wykonywalny. W pliku konfiguracyjnym użytkownik musi podać:

- Adres jednego z nadzorców lub serwera dostępowego (`basePath`) - adres musi być w formie URI.
- Adres zewnętrznego brokera wiadomości (`rabbitPath`) - adres musi być w formie URI.
- Czy aplikacja powinna podczas połączenia RDP używać danych dostępowych takich samych jak przy logowaniu do systemu (`useRdpCredentials`) - `true` albo `false`.
- Czy aplikacja powinna uruchamiać zintegrowanego klienta RDP (`startRdp`) - `true` albo `false`. Przydaje się to przy wykorzystaniu własnego klienta RDP. Aplikacja po uzyskaniu sesji wyświetli dane dostępowe do przypisanej maszyny wirtualnej.

2.8.5. Konfiguracja panelu administracyjnego

Przy uruchomieniu panelu administracyjnego trzeba przekazać adres jednego z nadzorców lub serwera dostępowego (jeżeli taki jest używany). Aby tego dokonać należy ustawić zmienną środowiskową `API_URL` na adres dostępowy jednego z nadzorców lub serwera dostępowego.

2.8.6. Konfiguracja nadzorcy

Nadzorca posiada wiele parametrów związanych z działaniem całego systemu. Kontroluje on między innymi czas oczekiwania na powrót użytkownika do porzuconej sesji. Wszystkie parametry należy przekazać poprzez plik konfiguracyjny.

Nazwa pliku konfiguracyjnego musi spełniać wzorzec `appsettings.${Nazwa_srodowiska}.ini`. W przypadku uruchamiania aplikacji wewnątrz kontenera ustawione jest środowisko o nazwie `Production`. Aplikacja nadzorcy domyślnie przeszukuje folder `./config` w poszukiwaniu

plików konfiguracyjnych. Można zmienić ścieżkę do folderu konfiguracyjnego poprzez parametr `-c path/to/other/config/folder/`.

W pliku konfiguracyjnym nadzorca odczytuje dwie sekcje:

- **JwtSettings** - przechowuje parametr wykorzystywany do generowania unikatowych tokenów autoryzacyjnych
- **OneClickDesktop** - przechowuje parametry związane z działaniem systemu OneClickDesktop.

Poza tymi sekcjami można tam umieścić dowolne sekcje, które będzie przetwarzać ASP.NET [4]. Jednak warte uwagi są następujące sekcje:

- **Logging** - parametry loggera dostarczonego przez Microsoft (<https://www.nuget.org/packages/microsoft.extensions.logging>).
- **Kestrel** - parametry serwera HTTP udostępniającego aplikację. Aplikacja nie zadziała w trybie HTTPS bez skonfigurowanego certyfikatu.

Pozostają jeszcze 2 ważne parametry bez sekcji zaimplementowane przez ASP.NET:

- **AllowedHosts** - lista adresów z których zapytania będą obsługiwane.
- **urls** - lista adresów, na których aplikacja będzie nasłuchiwać zapytań.

Parametry sekcji JwtSettings

Sekcja ta zawiera jedynie parametr **Secret**. Należy go ustawić na dowolny napis. W przypadku uruchamiania więcej niż jednej instancji nadzorcy taka sama wartość parametru pozwala na zmianę nadzorcy bez konieczności ponownej autoryzacji.

Parametry sekcji OneClickDesktop

Sekcja zawiera parametry związane z komunikacją pomiędzy jednostkami systemu oraz kilka parametrów określających interwały czasowe zdarzeń. W dostarczonej przykładowej konfiguracji wykomentowane wartości oznaczają wartości domyślne.

- **OverseerId** - identyfikator nadzorcy używany w komunikacji poprzez wewnętrzny brokera wiadomości. Powinna być unikatowa w skali jednej instancji systemu OneClickDesktop. Domyślnie przyjmuje wartość **overseer-test**.

- **RabbitMQHostname** - adres dostępowy wewnętrznego brokera wiadomości. Domyślnie przyjmuje wartość `localhost`. Bez działającego procesu brokera pod tym adresem aplikacja wyłączy się z błędem zaraz po uruchomieniu.
- **RabbitMQPort** - port dostępowy wewnętrznego brokera wiadomości. Domyślnie przyjmuje wartość `5672`. Bez działającego procesu brokera pod tym portem aplikacja wyłączy się z błędem zaraz po uruchomieniu.
- **ModelUpdateInterval** - ilość sekund co ile nadzorca prosi serwery wirtualizacji o aktualizację modelu. Domyślnie przyjmuje wartość `60`. Zalecane jest aby wartość tego parametru opisywała czas od 30 do 60 sekund.
- **DomainShutdownTimeout** - ilość minut jaka musi upłynąć od oznaczenia maszyny wirtualnej stanem *Oczekiwanie na wyłączenie maszyny* do jej wyłączenia. Domyślnie przyjmuje wartość `15`.
- **DomainShutdownCounterInterval** - ilość sekund co ile nadzorca sprawdza czy maszyna wirtualna nadal oczekuje na zamknięcie. Zaleca się, aby ten czas dzielił czas z parametru **DomainShutdownTimeout** na równe części. Takich części nie powinno być mniej niż 10, ale także nie więcej niż 100. Każde takie sprawdzenie zużywa czas procesora.

Przykładowa konfiguracja

Konfiguracja ta umożliwia użycie protokołu HTTPS dzięki przekazaniu zabezpieczonego hasłem certyfikatu oraz nasłuchiowaniu na adresie obsługującym ten protokół.

```
AllowedHosts=*
```

```
urls=http://*:5000;https://*:5001
```

```
[Kestrel:Certificates:Default]
```

```
Password=password
```

```
Path=/overseer/overseer.pfx
```

```
[JwtSettings]
```

```
Secret=MySecretForJWTTokensPleaseChangeMe
```

```
[Logging.LogLevel]
```

```
Default=Information
```

```
Microsoft=Warning
Microsoft.Hosting.Lifetime=Information
```

```
[OneClickDesktop]
OverseerId=overseer-instance-123
RabbitMQHostname=1.2.3.4
RabbitMQPort=5678
ModelUpdateInterval=60
DomainShutdownTimeout=15
DomainShutdownCounterInterval=30
```

2.8.7. Konfiguracja serwera wirtualizacji

Serwer wirtualizacji posiada wiele parametrów związanych z zasobami przekazanymi do dyspozycji systemu. Wszystkie parametry należy przekazać poprzez pliki konfiguracyjne znajdujące się w jednym folderze.

Głównym plikiem konfiguracyjnym jest `virtsrv.ini`. Zawiera on parametry związane z komunikacją wewnątrz i na zewnątrz systemu. Dodatkowo znajdują się tam informacje o zasobach udostępnionych dla serwera wirtualizacji. Dodatkowe pliki konfiguracyjne reprezentują typy maszyn wirtualnych uruchamianych na tym serwerze wirtualizacji. Jedynym odstępstwem od reguły jest plik konfiguracyjny dla pakietu NLog. Musi się on znajdować w tym samym folderze co aplikacja oraz nazywać `NLog.config`.

Aplikacja serwera wirtualizacji domyślnie przeszukuje folder `./config` w poszukiwaniu plików konfiguracyjnych. Można zmienić ścieżkę do folderu konfiguracyjnego poprzez parametr `-c path/to/other/config/folder/`.

Główny plik konfiguracyjny

W głównym pliku konfiguracyjnym możemy wyróżnić 2 sekcje:

- **OneClickDesktop** - przechowuje parametry związane z działaniem systemu OneClickDesktop.
- **ServerResources** - przechowuje parametry określające wszystkie udostępnione zasoby.

W sekcji **OneClickDesktop** występują parametry:

- **VirtualizationServerId** - identyfikator serwera wirtualizacji używany w komunikacji poprzez wewnętrznego brokera wiadomości. Powinna być unikatowa w skali jednej instancji

systemu OneClickDesktop. Domyślnie przyjmuje wartość `virtsrv-test`.

- **OverssersCommunicationShutdownTimeout** - po upływie tylu sekund bez komunikacji od jakiegokolwiek nadzorcy serwer wirtualizacji uznaje, że zabrakło nadzorców w systemie. Nastąpi wtedy wyłączenie aplikacji serwera wirtualizacji. Domyślnie parametr przyjmuje wartość 120. Zaleca się aby wartość parametru była większa niż dwukrotność parametru `ModelUpdateInterval` z konfiguracji nadzorcy opisanej w sekcji 2.8.6.
- **LibvirtUri** - adres do połączenia z usługą libvirta. Vagrant jak i biblioteka libvirta skorzysta z tego adresu do połączenia. Specyfikacja libvirta dokładnie opisuje format tego adresu [18].
- **VagrantFilePath** - ścieżka do specjalnie przygotowanego pliku wsadowego dla Vagranta. Wykorzystywany jest przez system do uruchamiania i wyłączania maszyn wirtualnych. Aplikacja nie zadziała prawidłowo bez ustawionej wartości parametru.
- **PostStartupPlaybook** - skrypt wykonywany przy pomocy Ansible'a zaraz po uruchomieniu maszyny wirtualnej. Koniecznie należy przetestować czy konfiguracja wykonuje się prawidłowo. Przy każdym błędzie wykonania skryptu maszyna wirtualna zostanie usunięta. Domyślnie przyjmuje wartość `res/poststartup_playbook.yml`.
- **VagrantboxUri** - identyfikator szablonowej maszyny wirtualnej. Musi ona spełniać szereg wymogów opisanych w sekcji 2.7.6. W przeciwnym wypadku system nie będzie w stanie uruchomić takiego szablonu. Aplikacja nie zadziała prawidłowo bez ustawionej wartości parametru.
- **BridgeInterfaceName** - nazwa interfejsu sieciowego w skonfigurowanego w trybie bridge, do którego zostanie podłączona każda uruchomiona maszyna wirtualna. Domyślnie przyjmuje wartość `br0`.
- **BridgedNetwork** - adres sieci, do której podłączona zostanie maszyna wirtualna poprzez `BridgeInterfaceName`, podany w formacie CIDR. Jeżeli maszyna nie będzie posiadać adresu z podanej sieci po uruchomieniu to zostanie wyłączona. Aplikacja nie zadziała prawidłowo bez ustawionej wartości parametru.
- **InternalRabbitMQHostname** - adres dostępowy wewnętrznego brokera wiadomości. Domyślnie przyjmuje wartość `localhost`. Bez działającego procesu brokera pod tym adresem aplikacja wyłączy się z błędem zaraz po uruchomieniu.

- **InternalRabbitMQPort** - port dostępowy wewnętrznego brokera wiadomości. Domyślnie przyjmuje wartość **5672**. Bez działającego procesu brokera pod tym portem aplikacja wyłączy się z błędem zaraz po uruchomieniu.
- **ExternalRabbitMQHostname** - adres dostępowy zewnętrznego brokera wiadomości. Domyślnie przyjmuje wartość **localhost**. Bez działającego procesu brokera pod tym adresem aplikacja wyłączy się z błędem zaraz po uruchomieniu.
- **ExternalRabbitMQPort** - port dostępowy zewnętrznego brokera wiadomości. Domyślnie przyjmuje wartość **5673**. Bez działającego procesu brokera pod tym portem aplikacja wyłączy się z błędem zaraz po uruchomieniu.
- **ClientHeartbeatChecksForMissing** - liczba nieudanych sprawdzeń czy aplikacja kliencka nadal jest połączona do maszyny wirtualnej. Po tej liczbie prób maszyna wirtualna oznaczana jest jako oczekująca na zamknięcie. Domyślnie przyjmuje wartość **2**.
- **ClientHeartbeatChecksDelay** - czas w milisekundach pomiędzy sprawdzeniami, czy aplikacja kliencka nadal jest połączona do maszyny wirtualnej. Domyślnie przyjmuje wartość **10000**.

W sekcji **ServerResources** występują parametry:

- **Cpus** - liczba wątków, które może wykorzystać system. Domyślnie przyjmuje wartość **2**.
- **Memory** - ilość pamięci operacyjnej w MiB jaką może wykorzystać system. Domyślnie przyjmuje wartość **2048**.
- **Storage** - ilość przestrzeni dyskowej w GiB jaką może wykorzystać system. Domyślnie przyjmuje wartość **100**.
- **GPUsCount** - ilość kart graficznych przekazanych do systemu. Domyślnie przyjmuje wartość **0**.
- **MachineTypes** - lista typów maszyn wirtualnych. Każda nazwa typu jest oddzielona od sąsiednich przecinkiem. Nazwa typu musi składać się ze znaków opisanych następującym wyrażeniem regularnym **[a-zA-Z0-9\ -]**. Dla każdej z nazw wyszukiwany jest plik konfiguracyjny o nazwie zaczynającej się nazwą typu i kończącą się **_template.ini**.

Gdy $n = \text{GPUsCount}$ jest większy od 0, wtedy w pliku muszą znaleźć się sekcje od **ServerGPU.1** do **ServerGPU.n** włącznie. W przeciwnym wypadku serwer wirtualizacji zakończy się z błędem zaraz po uruchomieniu.

2.8. URUCHOMIENIE SYSTEMU

Każda z tych sekcji zawiera parametr `AddressCount`, który określa jak duża jest grupa IOMMU w której znajduje się przekazywane urządzenie PCI. W przypadku $m = \text{AddressCount}$ w tej sekcji muszą znaleźć się parametry od `Address_1` do `Address_m` włącznie. W przeciwnym wypadku serwer wirtualizacji zakończy się z błędem zaraz po uruchomieniu. Każdy z tych parametrów opisuje adres pojedynczego urządzenia na magistrali PCI. Adres na magistrali PCI musi zostać opisany w formacie uzyskanym z polecenia `lspci -` `${domain:4}:${bus:2}:${slot:2}.${function:1}`.

Przykładowa zawartość głównego pliku konfiguracyjnego:

```
[OneClickDesktop]
VirtualizationServerId=virtsrv-instance-123
OverssersCommunicationShutdownTimeout=120
LibvirtUri=qemu:///system
VagrantFilePath=res/Vagrantfile
PostStartupPlaybook=res/poststartup_playbook.yml
VagrantboxUri=smogork/archlinux-rdp

BridgeInterfaceName=br0
BridgedNetwork=1.2.3.0/24

InternalRabbitMQHostname=1.2.3.4
InternalRabbitMQPort=5678

ExternalRabbitMQHostname=1.2.3.4
ExternalRabbitMQPort=8765

ClientHeartbeatChecksForMissing=2
ClientHeartbeatChecksDelay=10000

[ServerResources]
Cpus=6
Memory=4096
Storage=200
GPUsCount=1
MachineTypes=cpu,cpu-memory
```

```
[ServerGPU.1]
AddressCount=2
Address_1=0000:03:00.0
Address_2=0000:03:00.1
```

Pliki konfiguracyjne opisujące typy maszyn wirtualnych

W pliku opisującym typy zawarte jest o zasobach wymaganych do uruchomienia maszyny danego typu.

Każdy typ ma przypisaną nazwę. Oznaczmy ją jako `${template_name}`. Nazwa typu powinna być unikatowa w skali systemu. Jeżeli chcemy udostępnić ten sam typ na wielu serwerach wirtualizacji, należy nadać mu takie same zasoby. System może nieprawidłowo zarządzać zasobami, gdy uzyska dwa różne wzorce zasobów dla tego samego typu maszyny. Plik opisujący typ musi mieć nazwę `${template_name}_template.ini` oraz znajdować się w folderze z pozostałymi plikami konfiguracyjnymi. Jeżeli dla jakiegokolwiek nazwy typu aplikacja nie znajdzie pliku konfiguracyjnego to zakończy się z błędem zaraz po uruchomieniu.

W znalezionym pliku musi być sekcja o nazwie `${template_name}_template` z parametrami:

- **HumanReadableName** - nazwa wyświetlana w aplikacji klienckiej. Domyślnie przyjmuje wartość `${template_name}`.
- **Cpus** - liczba wątków potrzebna do uruchomienia maszyny. Domyślnie parametr przyjmuje wartość 2.
- **Memory** - ilość pamięci operacyjnej w MiB potrzebnej do uruchomienia maszyny. Domyślnie parametr przyjmuje wartość 512.
- **Storage** - ilość przestrzeni dyskowej w GiB potrzebnej do uruchomienia maszyny. Domyślnie parametr przyjmuje wartość 20.
- **AttachGpu** - informacja czy maszyna powinna mieć przekazaną kartę graficzną. Domyślnie parametr przyjmuje wartość `false`.

Przykładowa zawartość pliku konfiguracyjnego typu maszyny:

```
[gpu_template]
HumanReadableName=Efficiency machine with GPU attached
Cpus=1
```

2.8. URUCHOMIENIE SYSTEMU

Memory=2048

Storage=55

AttachGpu = true

2.8.8. Procedura uruchomienia

Prawidłowy start systemu powinien być wykonany w następującej kolejności:

1. Uruchomić zewnętrznego i wewnętrznego brokera wiadomości.
2. Poczekać na prawidłowy start brokerów wiadomości.
3. Uruchomić wymaganą liczbę nadzorców.
4. Poczekać na prawidłowy start przynajmniej jednego z nadzorców.
5. Uruchomić serwer HTTP udostępniający panel administracyjny.
6. Uruchomić wszystkie serwery wirtualizacji.
7. Poczekać aż w systemie znajdą się w pełni uruchomione maszyny wszystkich zarejestrowanych typów.

Po opisanych krokach system jest gotowy do użytkowania przez użytkowników. Ważne jest, aby przed uruchomieniem jakiegokolwiek nadzorcy brokerzy wiadomości byli już prawidłowo zainicjalizowani i gotowi na przyjęcie komunikacji. W przeciwnym wypadku aplikacja nadzorcy zakończy się z błędem. Serwer wirtualizacji także zakończy się z błędem, jeżeli zabraknie brokerów wiadomości lub nadzorców.

2.8.9. Parametry uruchomienia kontenera panelu administracyjnego

Kontener zawierający panel administracyjny udostępnia aplikację poprzez serwer HTTP nginx. Do komunikacji wystawia on porty 80 (HTTP) oraz 443 (HTTPS), które należy przekierować na odpowiednie porty uruchamiającego systemu.

Adres dostępu do nadzorców

Aplikacja panelu administracyjnego oczekuje ustalonego adresu dostępu do jakiegokolwiek nadzorcy. Przy uruchomieniu kontenera należy ustawić zmienną środowiskową `API_URL` zgodnie z opisem w sekcji 2.8.5.

Certyfikat SSL

W celu uruchomienia panelu w trybie HTTPS należy przekazać certyfikat wraz ze zmodyfikowaną konfiguracją serwera nginx.

```
-v {PATH_TO_CERT}:{PATH_TO_CERT_IN_CONTAINER}
-v {PATH_TO_KEY}:{PATH_TO_KEY_IN_CONTAINER}
-v {PATH_TO_CONF}:/etc/nginx/conf.d/default.conf
```

gdzie `PATH_TO_CERT`, `PATH_TO_KEY` i `PATH_TO_CONF` są ścieżkami bezwzględnymi na systemie uruchamiającym, a `PATH_TO_CERT_IN_CONTAINER` oraz `PATH_TO_KEY_IN_CONTAINER` są ścieżkami bezwzględnymi zgodnymi ze ścieżkami zdefiniowanymi w konfiguracji.

2.8.10. Parametry uruchomienia kontenera nadzorcy

Aplikacja nadzorcy udostępnia API poprzez serwer HTTP Kestrel. Do komunikacji wystawia porty 5000 (HTTP) oraz 5001 (HTTPS), które można zmienić w dostarczonej konfiguracji. Należy przekierować je na odpowiednie porty uruchamiającego systemu.

Plik konfiguracyjny

Aplikacja nadzorcy poszukuje plików konfiguracyjnych w lokalizacji `/overseer/config/`. Należy przekazać folder zawierający konfiguracje z systemu uruchamiającego (`PATH_TO_CONFIGS`) do wnętrza kontenera pod dokładnie tę lokalizację pod postacią woluminu.

```
-v {PATH_TO_CONFIGS}:/overseer/config
```

Certyfikat SSL

W celu uruchomienia nadzorcy w trybie HTTPS należy przekazać certyfikat SSL w formacie `.pfx` przy pomocy woluminów oraz plik konfiguracyjny zgodny z opisem z rozdziału 2.8.6. Przy konfiguracji używającej protokołu HTTPS i braku certyfikatów aplikacja zakończy się z błędem zaraz po uruchomieniu.

```
-v {PATH_TO_CERT}:{PATH_TO_CERT_IN_CONTAINER}
```

gdzie `PATH_TO_CERT` jest ścieżką bezwzględną do certyfikatu na systemie uruchamiającym oraz `PATH_TO_CERT_IN_CONTAINER` jest ścieżką bezwzględną wewnątrz kontenera zgodną z konfiguracją.

2.8.11. Parametry uruchomienia kontenera serwera wirtualizacji

Serwer wirtualizacji zarządza maszynami wirtualnymi na systemie uruchamiającym. Wymaga nietypowo podłączonych woluminów aby prawidłowo funkcjonować.

Zasoby libvirta

W celu komunikacji z usługą libvirta działającą na systemie uruchamiającym musimy przekazać gniazdo sieciowe do wnętrza kontenera. Musi ono udawać, że jest usługą libvirta uruchomioną we wnętrzu kontenera. Dodatkowo nowo tworzone maszyny powinny zapisywać się pomiędzy uruchomieniami kontenera. W tym celu trzeba przekazać także cały folder z danymi libvirta. Taki efekt można uzyskać przy pomocy woluminów przekazując

```
-v /var/run/libvirt/libvirt-sock:/var/run/libvirt/libvirt-sock
-v /var/lib/libvirt/:/var/lib/libvirt/
```

Zasoby vagranta

Przy starcie kontenera zasoby vagranta są puste. Oznacza to, że przy każdym pierwszym uruchomieniu maszyny wirtualnej będzie ona musiała być pobrana i rozpakowana do zasobów libvirta. Zabiera to czas i przestrzeń dyskową. Aby zapewnić trwałość Vagrant Boxów pomiędzy uruchomieniami, oraz zarządzanie nimi z poziomu systemu uruchamiającego, należy przekazać do kontenera główny folder vagranta. Folder domowy użytkownika wykonawczego powinien być wykorzystany do tego celu.

```
-v ${HOME}/.vagrant.d/boxes:/root/.vagrant.d/boxes/
```

gdzie HOME to ścieżka do folderu domowego użytkownika wykonawczego.

Plik konfiguracyjny

Aplikacja serwera wirtualizacji poszukuje plików konfiguracyjnych w lokalizacji `/app/config/docker-test/`. Należy przekazać folder zawierający konfiguracje z systemu uruchamiającego (`PATH_TO_CONFIGS`) do wnętrza kontenera pod dokładnie tą lokalizację pod postacią woluminu.

```
-v {PATH_TO_CONFIGS_DIR}:/app/config/
```

Można także zmienić lokalizację folderu z konfiguracją ustawiając zmienną środowiskową `CONFIG`. Może być ona względna do ścieżki `/app`.

Nie można zapomnieć o wyjątku konfiguracji NLoga, którą trzeba przekazać do folderu `/app`.

2.8.12. Przykład minimalnego systemu

Uruchamiając kontenery z modułami należy podać odpowiednie parametry, aby aplikacje wewnątrz pracowały prawidłowo. Przykładowy minimalny system wraz z parametrami można zobaczyć w module `demonstration` [7]. Znajduje się tam system składający się z jednego brokera wiadomości (zewnętrzna i wewnętrzna komunikacja w jednym), panelu administracyjnego, jednego nadzorcy i jednego serwera wirtualizacji. Konta użytkowników zdefiniowane są w systemie katalogowym, a ich katalogi domowe udostępniane poprzez serwer NFS. W celu umożliwienia uruchomienia większej ilości nadzorców stworzony został prosty serwer dostępowy. System zawarty w tym repozytorium udostępnia pełną funkcjonalność. Można znaleźć tam przykłady zarówno konfiguracji modułów jak i parametrów startowych dla kontenerów.

2.9. Interakcja z systemem

2.9.1. Typowe działanie systemu

Po uruchomieniu opisanym w rozdziale 2.8.8 system powinien być w pełni funkcjonalny. Typowym zachowaniem systemu będzie cykliczne wysyłanie próśb przez nadzorców do serwerów wirtualizacji. W logach serwerów wirtualizacji, jak i nadzorców, powinny cyklicznie pojawiać się informacje o prośbie lub odpowiedzi na zapytanie o model.

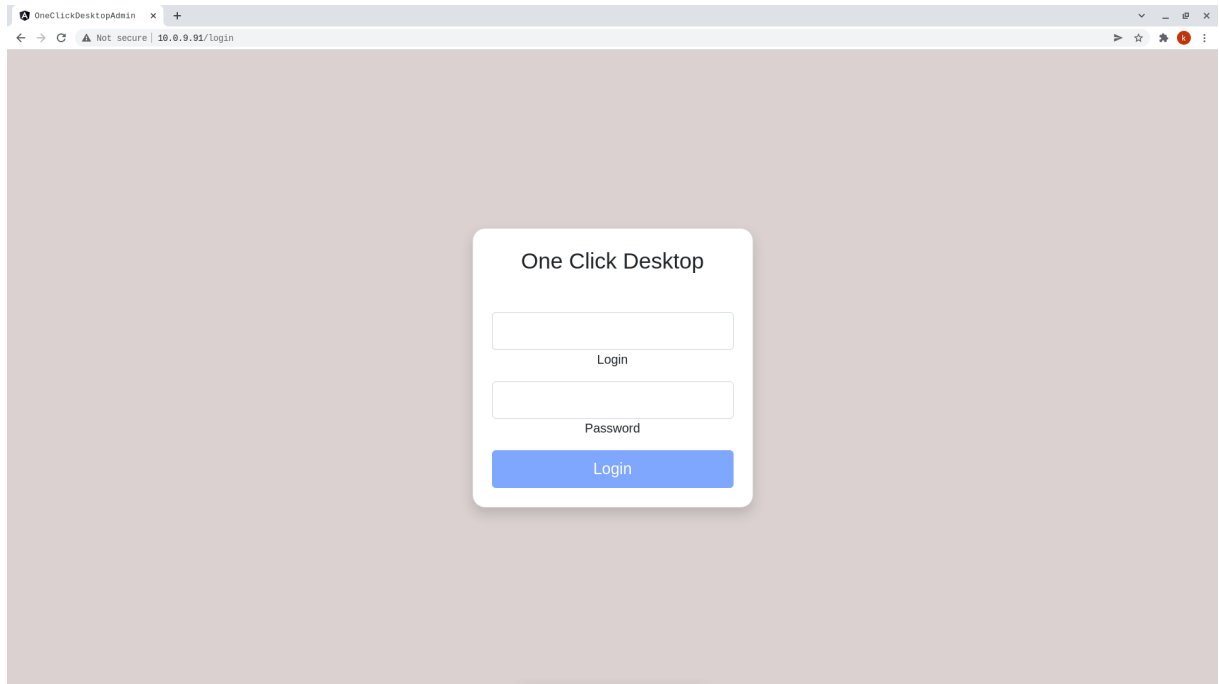
Zaraz po starcie systemu powinny pojawić się prośby o włączenie maszyn wirtualnych na odpowiednich serwerach wirtualizacji. Na standardowym wyjściu aplikacji powinny być widoczne informacje od uruchamiającego się Vagranta. Przy każdej nowo utworzonej sesji powinny wybudzać się następne maszyny wirtualne. Często problemy z uruchomieniem maszyn wirtualnych będą oznaczać nieskończone próby ich uruchomienia, które zawsze będą kończyć się niepowodzeniem. Serwer wirtualizacji powinien wszystkie informacje o błędach zapisywać do pliku z logami we wnętrzu kontenera w folderze roboczym aplikacji `/app`.

Przy wyłączaniu serwera wirtualizacji mogą pozostać nadal działające maszyny wirtualne wytworzone przez niego. Maszyny takie należy ręcznie wyłączyć lub usunąć. Nazwy maszyn można wyczytać z logów pracy serwera wirtualizacji.

2.9.2. Funkcje panelu administracyjnego

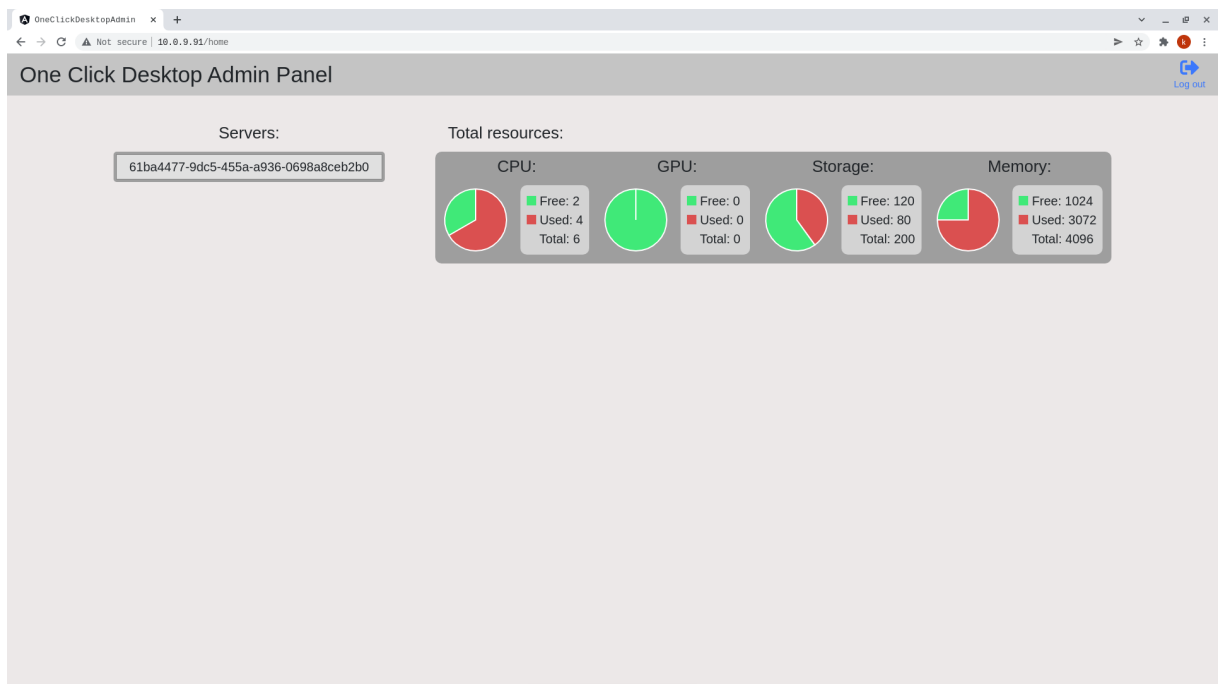
Po wejściu na stronę panelu administracyjnego, administrator musi podać nazwę użytkownika i hasło (rysunek 2.15). Jeżeli rzeczywiście podane konto jest kontem administratora przejdzie dalej do panelu. W przeciwnym wypadku zostanie zwrócony błąd dostępu.

2.9. INTERAKCJA Z SYSTEMEM



Rysunek 2.15: Okno logowania panelu administracyjnego

Po zalogowaniu administrator ma dostęp do głównego panelu z podsumowaniem wszystkich dostępnych zasobów. Po lewej stronie ma on dostępną listę wszystkich aktywnych serwerów wirtualizacji w systemie (rysunek 2.16).



Rysunek 2.16: Widok wszystkich zasobów systemu

Po naciśnięciu na wybrany serwer wirtualizacji na środku pojawiają się szczegółowy opis

zasobów konkretnego serwera (rysunek 2.17). Administrator, oprócz zasobów, może sprawdzić ile aktualnie jest uruchomionych maszyn (sekcja *Running*) oraz ile jeszcze można uruchomić (sekcja *Free*).

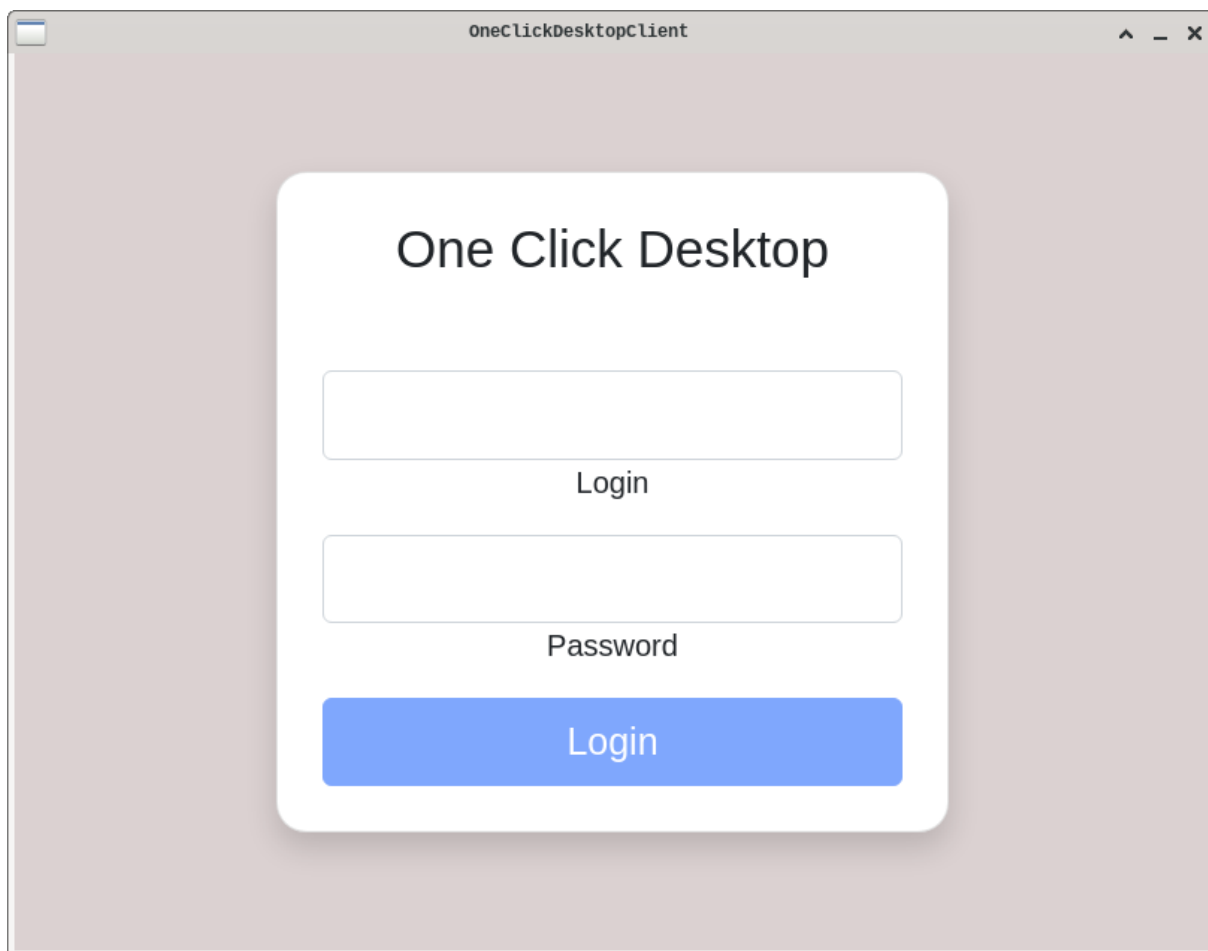


Rysunek 2.17: Widok szczegółowych zasobów serwera wirtualizacji

Zawartość panelu odświeża się automatycznie.

2.9.3. Funkcje aplikacji klienckiej

Po uruchomieniu aplikacji klienckiej użytkownik musi podać nazwę użytkownika i hasło (rysunek 2.18). Jeżeli użytkownik istnieje w bazie to przejdzie do ekranu głównego. W przeciwnym wypadku zostanie zwrócony błąd dostępu.



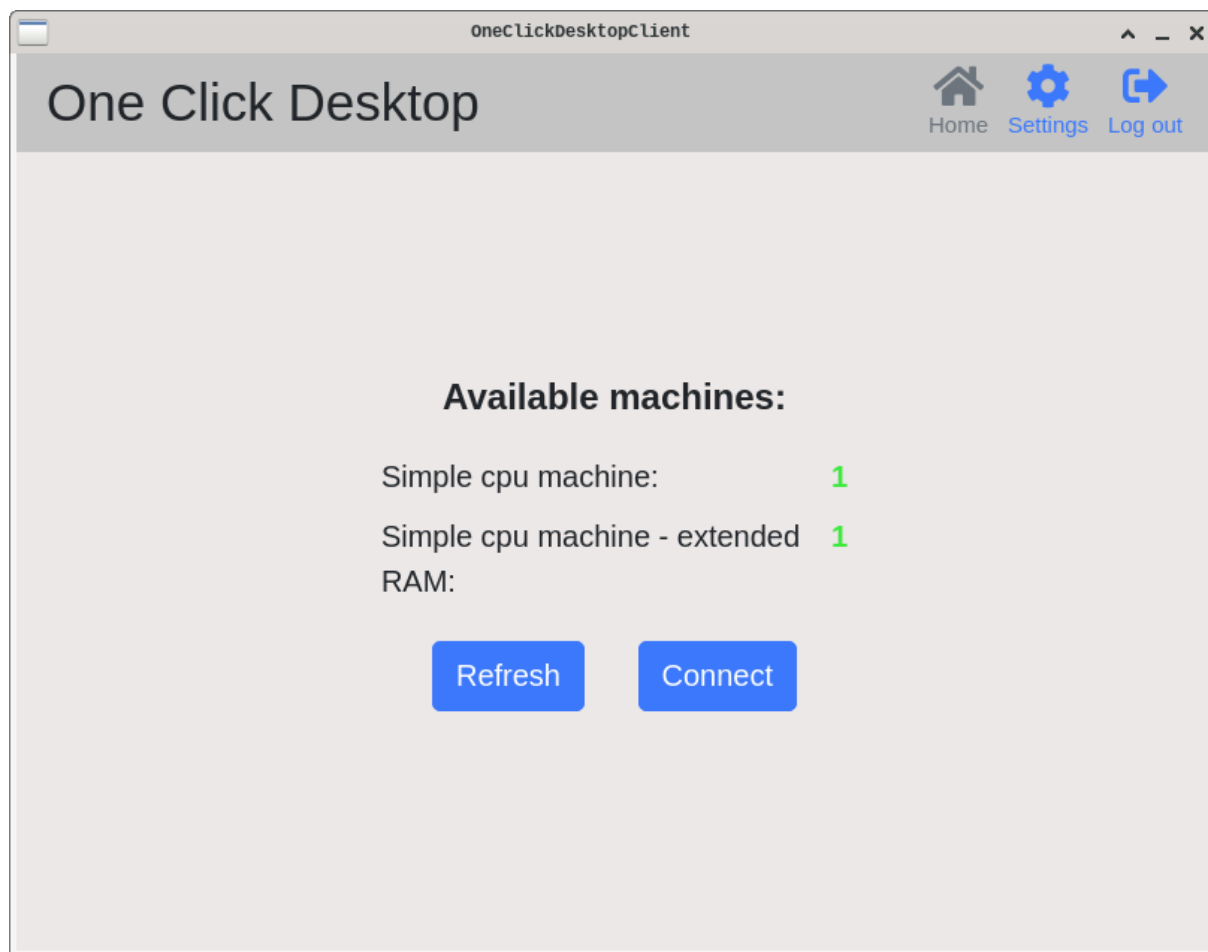
Rysunek 2.18: Ekran logowania aplikacji klienckiej

Po zalogowaniu użytkownik trafia do ekranu głównego (rysunek 2.19). Może tutaj zobaczyć ile jest dostępnych maszyn w systemie, poprosić o sesję oraz zmienić ustawienia aplikacji.

Przechodząc do ustawień (rysunek 2.20) użytkownik może edytować plik konfiguracyjny z poziomu aplikacji albo wrócić do ekranu głównego. Zmiana adresu nadzorca będzie wymagać ponownego uruchomienia. Inne ustawienia zostaną zastosowane od razu po zmianie.

Gdy użytkownik z ekranu głównego naciśnie przycisk *Connect* zostanie zapytany o typ sesji, jaki system ma mu przypisać (rysunek 2.21).

Po wybraniu jednego z typów sesji i naciśnięciu przycisku *Connect* aplikacja kliencka wyśle prośbę o znalezienie maszyny danego typu i utworzenie sesji dla pytającego użytkownika. Do czasu utworzenia sesji aplikacja będzie oczekiwać na dane do połączenia. W przypadku popraw-

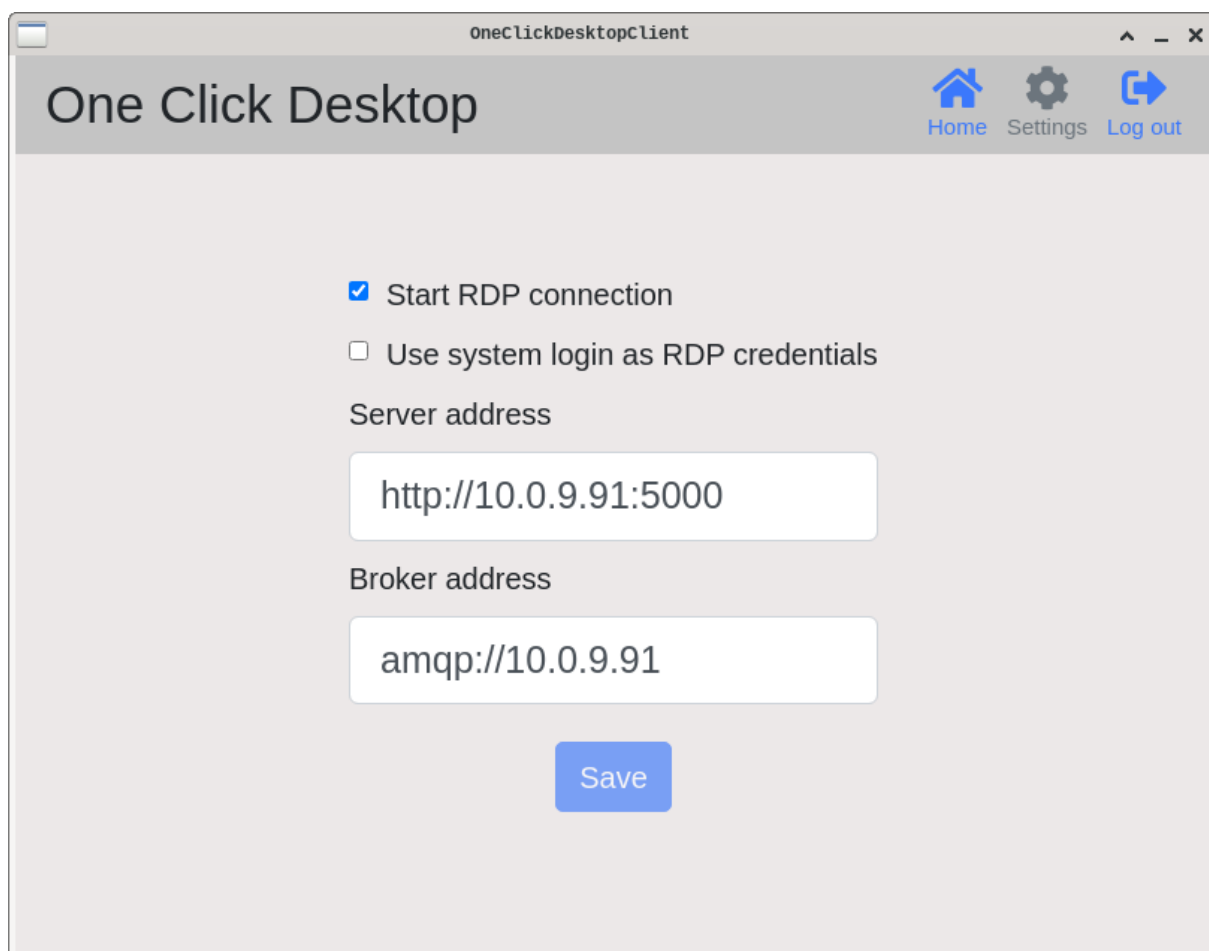


Rysunek 2.19: Ekran główny aplikacji klienckiej

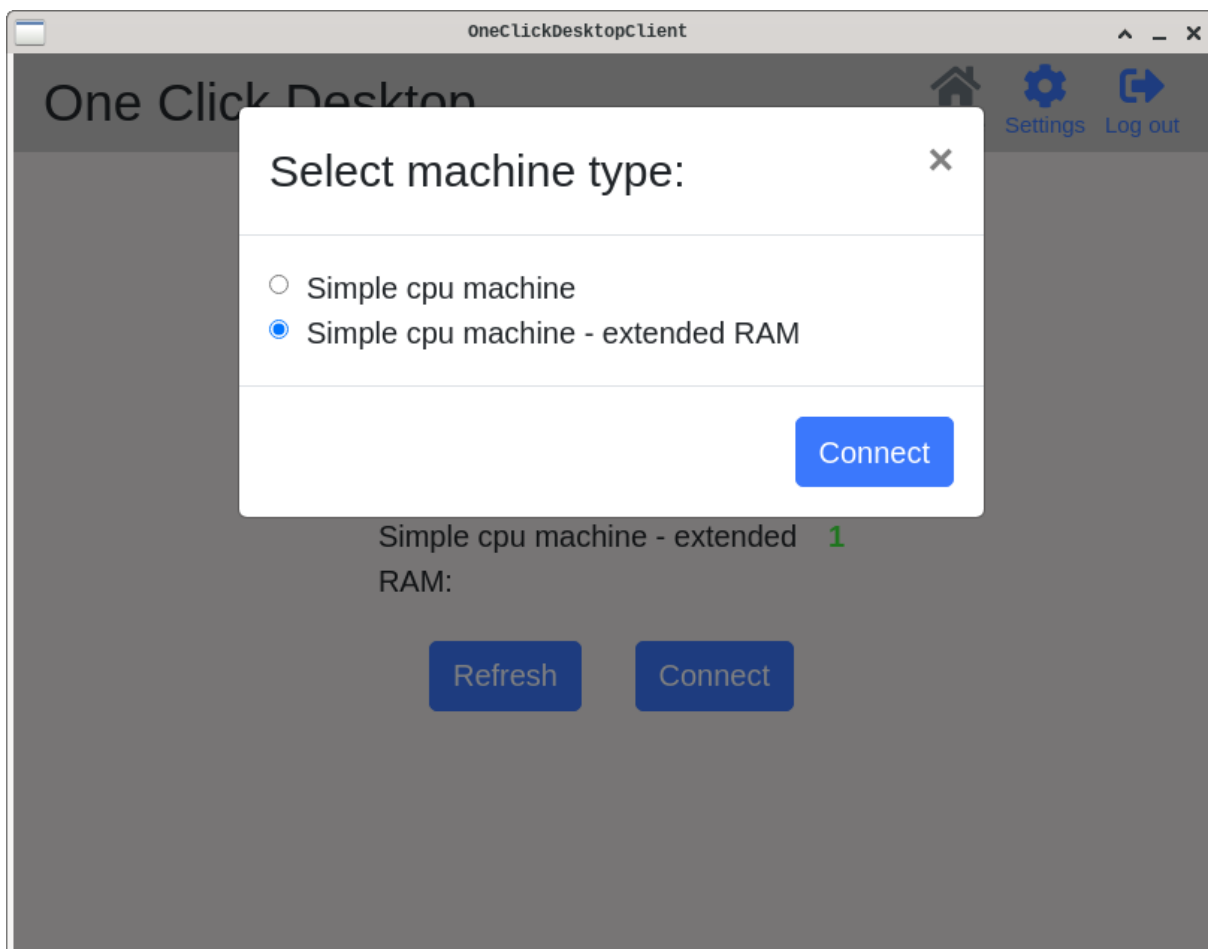
nego utworzenia sesji uruchomi się klient RDP i będzie można rozpocząć pracę. W przeciwnym wypadku zostanie zgłoszony błąd, a prośba o sesję zostanie umorzona.

Po prawidłowym utworzeniu sesji aplikacja rozpocznie zgłaszanie do zewnętrznego brokera wiadomości, że użytkownik pracuje na maszynie wirtualnej. Ten stan reprezentuje ekran pokazany na rysunku 2.22. Zamknięcie klienta RDP lub naciśnięcie przycisku *End session* spowoduje oznaczenie sesji jako *do usunięcia*. Od tego momentu użytkownik ma `DomainShutdownTimeout` minut na powrót do swojej sesji (szczegóły w rozdziale 2.8.6). Po upływie tego czasu sesja zostaje zamknięta, a maszyna z nią skojarzona wyłączona.

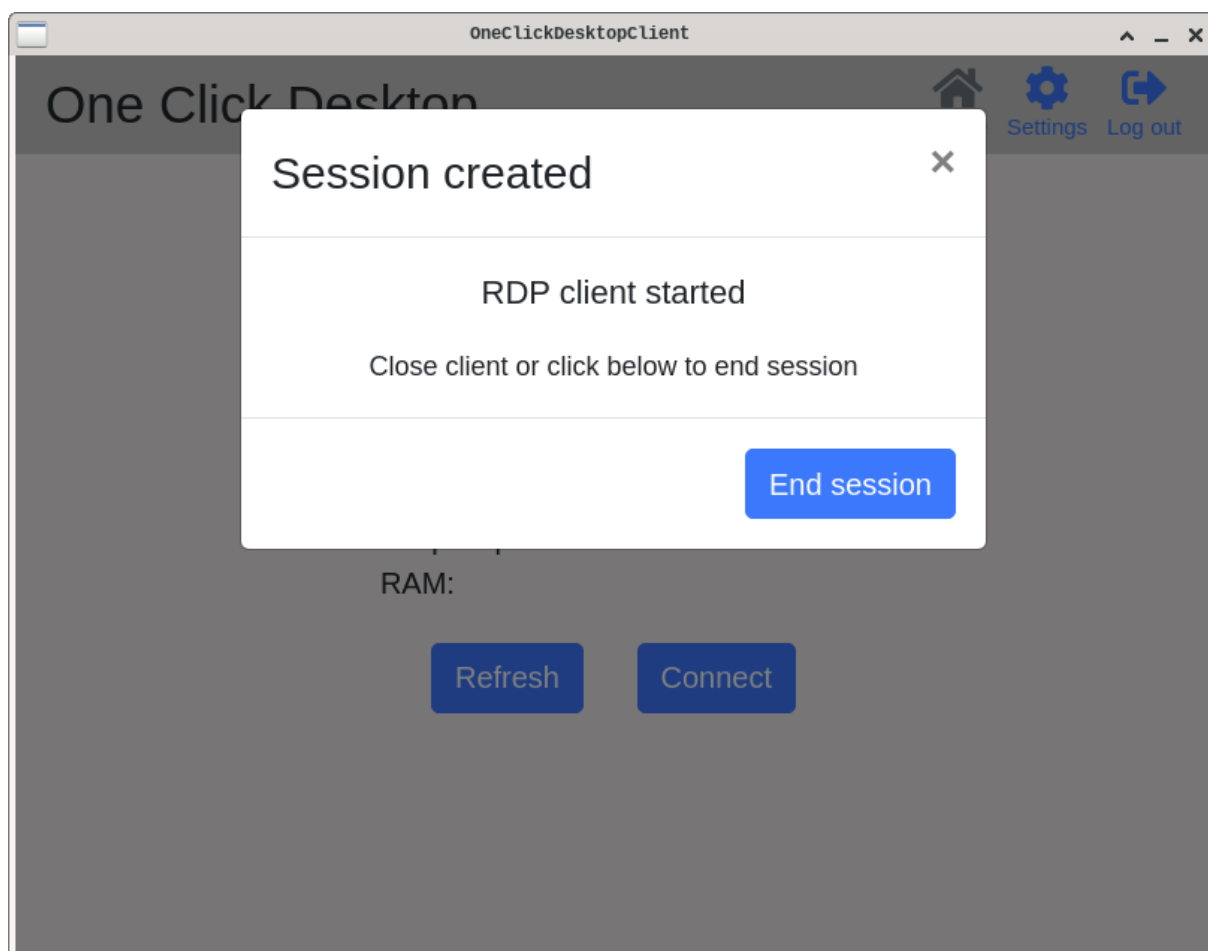
W przypadku gdy użytkownik poprosi aby nie uruchamiać dołączonego klienta RDP, albo uruchomienie klienta RDP zakończy się błędem, zostaną mu przekazane dane dostępowe do przypisanej maszyny wirtualnej (rysunek 2.23). Wtedy jedynym sposobem na zaznaczenie, że skończyło się pracę na maszynie, jest naciśnięcie przycisku *End session*.



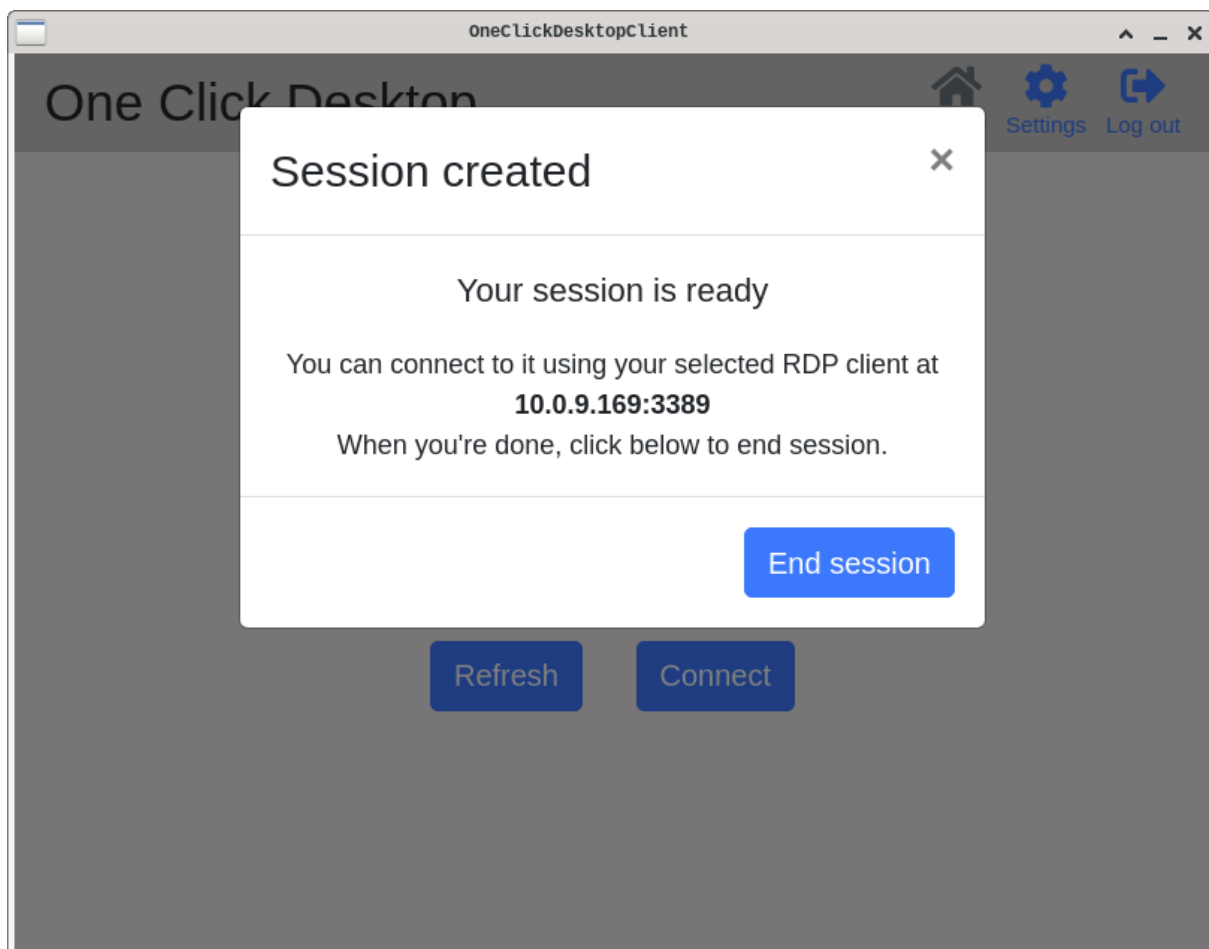
Rysunek 2.20: Ekran ustawień aplikacji klienckiej



Rysunek 2.21: Ekran wyboru typu maszyny do pracy



Rysunek 2.22: Ekran działającego połączenia RDP



Rysunek 2.23: Ekran trwania połączenia w zewnętrznym kliencie RDP

3. Analiza rozwiązania

3.1. Testowana funkcjonalność

Gotowy system poddany został testom akceptacyjnym mającym za zadanie sprawdzić poprawne działanie oraz spełnienie wymagań przedstawionych w rozdziałach 1.5 oraz 1.6. W opisie testów jako standardowe uruchomienie systemu rozumiemy procedurę opisaną w sekcji 2.8.8.

Wykonane zostały następujące scenariusze akceptacyjne:

3.1.1. Brak komunikacji z nadzorcą

Przy starcie samodzielnego serwera wirtualizacji i wykryciu braku komunikacji z jakimkolwiek nadzorcą (poprzez brokera wiadomości) serwer powinien poinformować o błędzie i zakończyć działanie.

Kroki:

1. Włącz wewnętrznego brokera wiadomości oraz serwer wirtualizacji.
2. Wyświetl błąd i zakończ działanie.

3.1.2. Utrata komunikacji z nadzorcą

Po poprawnym starcie systemu z pojedynczym nadzorcą oraz serwerem wirtualizacji, serwer powinien zakończyć pracę po wyłączeniu się ostatniego (jedyne) nadzorcy.

Kroki:

1. Uruchom system standardową procedurą.
2. Poczekać na prawidłowy start systemu.
3. Wyłącz nadzorcę lub brokery wiadomości.
4. Poczekać na wykrycie braku nadzorców (brak otrzymanych wiadomości).
5. Wyświetl błąd i zakończ działanie.

3.1.3. Standardowe użycie systemu przez użytkownika

Użytkownik podłącza się do systemu złożonego z pojedynczego nadzorcy oraz serwera wirtualizacji, gdzie działa przynajmniej jedna wolna maszyna. Użytkownik powinien prawidłowo otrzymać sesję, a po odłączeniu się od maszyny, powinna ona zostać wyłączona po 15 minutach (czas konfigurowalny).

Kroki:

1. Uruchom system standardową procedurą.
2. Poczekaj na uruchomienie przynajmniej jednej maszyny wirtualnej.
3. Poproś o sesję poprzez aplikację kliencką.
4. Podłącz się poprzez RDP do uzyskanej maszyny.
5. Zakończ sesję z poziomu aplikacji.
6. Po określonym czasie maszyna powinna się wyłączyć.

3.1.4. Standardowe użycie systemu przez użytkownika przy awarii nadzorcy

Użytkownik podłącza się do systemu złożonego z dwóch nadzorców oraz jednego serwera wirtualizacji, gdzie działa przynajmniej jedna wolna maszyna. Użytkownik uzyskuje sesję, a w trakcie jej użytkowania następuje awaria nadzorcy. Po odłączeniu się od sesji i ponownej prośbie o sesję, w czasie krótszym niż czas wyłączenia maszyny, użytkownik powinien otrzymać ponownie tę samą maszynę.

Kroki:

1. Uruchom system standardową procedurą z dwoma nadzorcami.
2. Poczekaj na uruchomienie przynajmniej jednej maszyny wirtualnej.
3. Poproś o sesję poprzez aplikację kliencką.
4. Podłącz się poprzez RDP do uzyskanej maszyny.
5. Wyłącz tego nadzorcę, z którym klient się komunikował.
6. Zakończ sesję z poziomu aplikacji.
7. Poproś ponownie o sesję poprzez aplikację kliencką (powinien uzyskać tę samą maszynę).
8. Podłącz się poprzez RDP do uzyskanej maszyny.

3.1. TESTOWANA FUNKCJONALNOŚĆ

3.1.5. Podłączenie nowego serwera wirtualizacji

W trakcie działania systemu nowy serwer wirtualizacji powinien zostać włączony do modelu nadzorców oraz wyświetlony w panelu administratora.

Kroki:

1. Uruchom system standardową procedurą.
2. Poczekaj na prawidłowy start systemu.
3. Włącz kolejną instancję serwera wirtualizacji.
4. Poczekaj na aktualizację modelu.
5. Sprawdź w panelu administratora, czy dwa serwery są w modelu.

3.1.6. Podłączenie nowego nadzorcy

W trakcie działania systemu nowy nadzorca powinien posiadać taki sam model, jak aktualnie działający

Kroki:

1. Uruchom system standardową procedurą.
2. Poczekaj na prawidłowy start systemu.
3. Włącz kolejną instancję nadzorcy.
4. Poczekaj na aktualizację modelu.
5. Sprawdź model na pierwszym nadzorcy poprzez panel administratora.
6. Sprawdź model na drugim nadzorcy poprzez panel administratora.

3.1.7. Odnotowanie utraty serwera wirtualizacji

W trakcie działania systemu, przy utracie serwera wirtualizacji, nadzorcy powinni usunąć go z modelu.

Kroki:

1. Uruchom system standardową procedurą.
2. Poczekaj na prawidłowy start systemu.
3. Wyłącz serwer wirtualizacji.

4. Poczekaj na odnotowanie straty.
5. Sprawdź model poprzez panel administratora.

3.1.8. Utrata komunikacji przy działającej sesji

W trakcie działania systemu, przy utracie ostatniego nadzorcy, serwer wirtualizacji powinien zakończyć działanie. Jeżeli serwer posiada działające sesje, przed zakończeniem pracy, musi poczekać na ich zakończenie.

Kroki:

1. Uruchom system standardową procedurą.
2. Poczekaj na prawidłowy start systemu.
3. Poproś o sesję poprzez aplikację kliencką.
4. Podłącz się poprzez RDP do uzyskanej maszyny.
5. Wyłącz nadzorcę lub brokery wiadomości.
6. Poczekaj na wykrycie braku nadzorców (brak otrzymanych wiadomości).
7. Wyświetl błąd, ale kontynuuj działanie.
8. Poczekaj na zakończenie sesji.
9. Zakończ działanie.

3.1.9. Odzyskanie komunikacji przy działającej sesji

W trakcie działania systemu, przy utracie ostatniego nadzorcy, serwer wirtualizacji powinien zakończyć działanie. Jeżeli serwer posiada działające sesje, przed zakończeniem pracy, musi poczekać na ich zakończenie. Jeżeli przed zakończeniem ostatniej sesji nastąpi przywrócenie komunikacji, serwer powinien kontynuować działanie po zakończeniu sesji.

Kroki:

1. Uruchom system standardową procedurą.
2. Poczekaj na prawidłowy start systemu.
3. Poproś o sesję poprzez aplikację kliencką.
4. Podłącz się poprzez RDP do uzyskanej maszyny.

3.2. ŚRODOWISKO TESTOWE

5. Wyłącz nadzorcę lub brokery wiadomości.
6. Poczekaj na wykrycie braku nadzorców (brak otrzymanych wiadomości).
7. Wyświetl błąd, ale kontynuuj działanie.
8. Włącz wyłączony wcześniej moduł.
9. Poczekaj na zakończenie sesji.
10. Kontynuuj działanie.

3.2. Środowisko testowe

Testy przeprowadzaliśmy na dwóch komputerach podłączonych do wspólnej sieci lokalnej. Usługa OpenLDAP była uruchomiona na innym komputerze oraz udostępniała testową bazę użytkowników. Każdy z nich pracował pod kontrolą systemu operacyjnego Arch Linux w wersji ze stycznia 2022 roku. Oba posiadały 8 GB pamięci operacyjnej oraz 4 rdzeniowy procesor o 8 wątkach (Intel i7-2600K oraz i7-4790). Do testów skonfigurowaliśmy serwery wirtualizacji, aby miały do dyspozycji 6 wątków oraz 4096 MB pamięci operacyjnej.

Niestety taka platforma uniemożliwiła przetestowanie funkcjonalności PCI Passthrough (patrz wymagania sprzętowe serwera wirtualizacji, rozdział 2.7.4). Budowa ich płyt głównych nie pozwalała na całkowitą izolację urządzeń podłączonych do złączy PCI Express. Skorzystaliśmy z innego komputera, który posiadał inną konstrukcję płyty głównej i każde z urządzeń zostało prawidłowo odizolowane dzięki przydzieleniu innych adresów pamięci. Pracował on pod kontrolą systemu operacyjnego Arch Linux w wersji ze stycznia 2022 roku. Posiadał on 48GB pamięci operacyjnej oraz 12 rdzeniowy procesor o 24 wątkach (AMD Threadripper 1920X). Uruchamiany na nim serwer wirtualizacji miał do dyspozycji 20 wątków oraz 43008 MB pamięci operacyjnej.

Na każdym z tych komputerów można było uruchomić aplikację serwera wirtualizacji, panelu administracyjnego oraz nadzorcy.

3.3. Wykonane testy

W trakcie rozwoju projektu zaprogramowaliśmy wiele testów automatycznych, które sprawdzały podstawową logikę wydzielonych części kodu. W przypadku modułów korzystających z zewnętrznych narzędzi (np. libvirt) stworzyliśmy proste testy integracyjne. Sprawdzają czy wszystkie wykorzystywane funkcjonalności tych narzędzi zostały prawidłowo zintegrowane.

Po zakończeniu rozwoju funkcjonalności systemu przystąpiliśmy do próby wdrożenia systemu w środowiskach testowych. Wykonaliśmy wtedy ręcznie podstawowe scenariusze uruchomienia systemu w postaci kontenerów dockerowych. Gdy już system pracował stabilnie przystąpiliśmy do ręcznego wykonywania scenariuszy akceptacyjnych

3.4. Wyniki testów

Testy automatyczne często kończyły się błędem po zmianie w kodzie. Był to dla nas znak aby sprawdzić co się dzieje i wprowadzić odpowiednie poprawki.

Próby wdrożenia systemu w środowisku testowym przyniosły odkrycie wielu błędów logicznych w naszych wstępnych pomysłach jak i samej implementacji. Dodatkowo pomogło nam to też lepiej zrozumieć wymagania sprzętowe jak i zależności naszego systemu.

Przy wykonywaniu testów akceptacyjnych mieliśmy szansę dopracować stabilność oraz ponownie skonfrontować wstępne pomysły z rzeczywistością. Po poprawkach błędów i upewnieniu się, że wszystkie scenariusze akceptacyjne są już spełnione system był już wystarczająco stabilny aby można było z niego korzystać. Zrozumieliśmy wtedy lepiej jakie rozwiązania nie są wygodne i wymagają poprawek albo nawet przeprojektowania. Nasze przemyślenia o nieudanych elementach systemu można znaleźć w podsumowaniu (rozdział 4.1).

4. Podsumowanie

4.1. Końcowy stan systemu

System, w swojej ostatecznej postaci, spełnia wszystkie postawione mu wymagania. Mimo trudności w implementacji niektórych funkcjonalności, w szczególności zarządzania maszynami wirtualnymi z poziomu kodu C#, ostateczny projekt działa zgodnie z oczekiwaniami. Projekt tworzony w ramach tej pracy uznajemy zatem za udany.

Elementem, który mógł zostać lepiej wykonany, jest zarządzanie maszynami wirtualnymi przy pomocy Vagranta. Z powodu braku interfejsu programistycznego udostępnianego przez ten program wykorzystywany jest on w naszym systemie za pośrednictwem wywołań poleceń powłoki. Nie jest to najlepsze rozwiązanie o czym świadczy liczba błędów, które wynikły podczas tworzenia systemu. Jest to w naszym przekonaniu najbardziej podatny na błędy element systemu. Taki sposób rozwiązania jest spowodowany chęcią skorzystania z Vagranta podczas projektowania systemu. Na tamtym etapie nie sprawdziliśmy czy dostępne są narzędzia umożliwiające korzystanie z niego z poziomu kodu. W momencie odkrycia braku takich narzędzi było już za późno na zastąpienie Vagranta innym rozwiązaniem.

Projektując komunikację wewnętrzną za pośrednictwem RabbitMQ postawiliśmy na wykorzystanie po jednej kolejce na odbieranie wiadomości bezpośrednich oraz zbiorczych. Z powodu różnych typów wysyłanych wiadomości musieliśmy deserializować wiadomości na różne typy obiektów, które zależą od typu otrzymanej wiadomości. Naszym zdaniem problem ten nie został przez nas rozwiązany zadowalająco i mógłby zostać wykonany lepiej.

Z pominięciem wyżej wymienionych aspektów uważamy projekt komunikacji i procesów biznesowych za dobrze wykonany. Dzięki głęboko przemyślanym rozwiązaniom w tych płaszczyznach ich implementacja była mało problematyczna. Podczas testów nie wystąpiły żadne problemy, które wymagałyby modyfikacji ustalonych procesów i sekwencji.

4.2. Możliwe ścieżki rozwoju

4.2.1. Panel administratora

Funkcjonalność panelu administratora jest obecnie ograniczona. Pozwala on jedynie na podejście do stanu zasobów w systemie. Informacja ta pozwala ocenić czy potrzebne jest uruchomienie nowych instancji serwera wirtualizacji. Jest to jednak jedyna dostępna informacja.

Możliwym rozwinięciem jest udostępnienie wglądu w cały model systemu przechowywany przez nadzorców. Umożliwi to ocenę stopnia zajętości maszyn każdego typu oraz ilości użytkowników systemu. Dodatkowo wgląd do stanu konkretnych maszyn i sesji może ułatwić rozwiązywanie problemów.

4.2.2. Użycie konkretnych kart graficznych

W konfiguracji typu maszyny wirtualnej możliwe jest jedynie wskazanie czy ma ona posiadać kartę graficzną na wyłączność. Oznacza to, że otrzymana karta graficzna nie jest sprecyzowana i może być dowolną ze skonfigurowanych w serwerze wirtualizacji.

W celu uniknięcia niespodzianek, co do modelu otrzymanej karty graficznej, preferowana byłaby możliwość sprecyzowania konkretnego modelu karty graficznej przekazywanej do maszyny danego typu. Możliwym sposobem osiągnięcia tej funkcjonalności jest rozszerzenie konfiguracji serwera wirtualizacji o możliwość nadania identyfikatora przekazywanym kartom graficznym.

4.2.3. Wsparcie innych protokołów zdalnego pulpitu

Protokół RDP został wybrany ze względu na swoją popularność oraz łatwość użycia. Jednak pożądanym byłoby wsparcie innych protokołów zdalnego pulpitu jak X2Go (<https://wiki.x2go.org/doku.php>) Dodanie wsparcia wymagałoby Vagrant Boxów posiadających skonfigurowany serwer wszystkich wspieranych protokołów. Dodatkowym protokołem, którego wsparcie może być trudniejsze, jest NVIDIA GameStream (<https://www.nvidia.com/en-us/shield/support/shield-tv/gamestream/>). Protokół ten umożliwia zdalne uruchamianie gier na komputerach wyposażonych w karty graficzne firmy NVIDIA.

4.3. Otwarte problemy

4.3.1. Konfiguracja połączenia RabbitMQ

Klient RabbitMQ używany przez moduły wewnętrzne do połączenia z instancją brokera udostępnia wiele opcji konfiguracji, które nie są udostępniane przez moduły. Konfiguracja ta definiuje opcje połączenia, takie jak hasło autoryzacji, czy niestandardowe ścieżki połączenia. W przypadku korzystania z niestandardowej instancji brokera konfiguracja ta może okazać się niezbędna. Z tego powodu powinno być możliwe przekazanie ustawień połączenia z brokerem wiadomości w plikach konfiguracyjnych modułów wewnętrznych.

4.3.2. Monitorowanie stanu połączenia klienta

Do monitorowania stanu połączenia z maszyną wirtualną wykorzystywany jest broker wiadomości RabbitMQ. Klient uznawany jest za połączony gdy istnieje kolejka z nazwą odpowiadającą identyfikatorowi sesji. Rozwiązanie to jest bardzo niestandardowe, kolejki używane są w nie zamierzony sposób. Dodatkowo klient RabbitMQ nie udostępnia możliwości powiadamiania o braku kolejki w sposób pasywny. Implementacja ta wymaga działania instancji brokera, która jest dostępna spoza wnętrza systemu. Może to powodować luki w bezpieczeństwie. Pożądane byłoby zastąpienie tej implementacji mechanizmem przeznaczonym do tego typu rozwiązań.

4.3.3. Znane problemy wyścigów

Pomimo przywiązania dużej uwagi do zaprojektowania komunikacji i procesów w sposób niwelujący ilość możliwych problemów wynikających z konkurencji wciąż istnieją miejsca, w których mogą one wystąpić. Znanym problemem jest możliwość uruchomienia dwa razy więcej maszyn niż zamierzono podczas startu systemu. Spowodowane jest to tym, że nadzorca otrzymując informację o dostępności serwera wirtualizacji poprosi go o utworzenie wymaganych maszyn wirtualnych. Jeżeli zanim serwer prześle informację o wykonaniu zadania, drugi serwer wirtualizacji również stanie się dostępny, nadzorca poprosi go o utworzenie tych samych maszyn. Taki ciąg wydarzeń doprowadzi do działania dwa razy większej liczby maszyn niż było zamierzone.

Możliwym rozwiązaniem problemu może być przeprojektowanie procesu uruchamiania wymaganych maszyn w celu uwzględnienia możliwości pojawienia się kolejnych instancji serwera wirtualizacji.

Bibliografia

Repozytoria kodu

- [6] P.Widomski K.Smogór. *Przykładowy VagrantBox smogork/archlinux-rdp*. URL: <https://app.vagrantup.com/smogork/boxes/archlinux-rdp> (term. wiz. 21.01.2022).
- [7] P.Widomski K.Smogór. *Repozytorium z demonstracyjnym systemem*. URL: <https://github.com/one-click-desktop/demonstration> (term. wiz. 21.01.2022).
- [8] P.Widomski K.Smogór. *Repozytorium z konfiguracją systemu*. 2022. URL: <https://github.com/one-click-desktop/configuration> (term. wiz. 21.01.2022).
- [9] P.Widomski K.Smogór. *Strona z wydaniem aplikacji klienckiej*. 2022. URL: <https://github.com/one-click-desktop/client/releases> (term. wiz. 21.01.2022).

Źródła zewnętrzne

- [1] *2020 State of Remote Work Report (United States)*. 2021. URL: <https://resources.owllabs.com/state-of-remote-work/2020> (term. wiz. 25.01.2022).
- [2] *2021 State of Remote Work Report (Europe)*. 2021. URL: <https://resources.owllabs.com/state-of-hybrid-work/2021/europe> (term. wiz. 25.01.2022).
- [3] *AMD I/O Virtualization Technology (IOMMU) Specification*. Kw. 2021. URL: https://www.amd.com/system/files/TechDocs/48882_IOMMU.pdf (term. wiz. 21.01.2022).
- [4] *Configuration in ASP.NET Core*. 7 sty. 2022. URL: <https://docs.microsoft.com/en-us/aspnet/core/fundamentals/configuration/?view=aspnetcore-5.0> (term. wiz. 21.01.2022).
- [5] *HTTP Over TLS - RFC 2818*. Maj 2000. URL: <https://datatracker.ietf.org/doc/html/rfc2818> (term. wiz. 21.01.2022).
- [10] *LDAP authentication*. URL: https://wiki.archlinux.org/title/LDAP_authentication#Client_Setup (term. wiz. 26.01.2022).

- [11] *Lista klientów RDP zgodnych z serwerem XRDP*. 2021. URL: <https://github.com/neutrino-labs/xrdp#overview> (term. wiz. 21.01.2022).
- [12] *Mechanizm odrzucania niedostarczonych wiadomości w RabbitMQ*. URL: <https://www.rabbitmq.com/publishers.html#unroutable> (term. wiz. 21.01.2022).
- [13] *Mechanizm potwierdzenia otrzymania wiadomości w RabbitMQ*. URL: <https://www.rabbitmq.com/confirms.html> (term. wiz. 21.01.2022).
- [14] *Oferta Citrix'a usług zdalnego pulpitu*. URL: <https://www.citrix.com/pl-pl/products/citrix-virtual-apps-and-desktops/> (term. wiz. 21.01.2022).
- [15] *Opis certyfikatu SSL/TLS*. URL: <https://protonmail.com/blog/tls-ssl-certificate/> (term. wiz. 21.01.2022).
- [16] *Rozdział dokumentacji Vagranta o tworzeniu boxów*. URL: <https://www.vagrantup.com/docs/boxes/base> (term. wiz. 21.01.2022).
- [17] *Rozdział dokumentacji Ansible'a o playbookach*. 31 mar. 2021. URL: https://docs.ansible.com/ansible/latest/user_guide/playbooks_intro.html (term. wiz. 21.01.2022).
- [18] *Rozdział dokumentacji libvirt o adresie dostępu*. URL: <https://libvirt.org/uri.html> (term. wiz. 21.01.2022).
- [19] *Rozdział dokumentacji Red Hat Enterprise Linux o PCI Passthrough*. URL: https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/5/html/virtualization/chap-virtualization-pci_passthrough (term. wiz. 21.01.2022).
- [20] *Rozdział dokumentacji vagrant-libvirt o tworzeniu boxów*. URL: <https://github.com/vagrant-libvirt/vagrant-libvirt#create-box> (term. wiz. 21.01.2022).
- [21] *Rozdział dokumentacji Vagranta o boxach*. 31 mar. 2021. URL: <https://www.vagrantup.com/docs/boxes> (term. wiz. 21.01.2022).
- [22] *Rozdział w dokumentacji Ansible o wymaganiach komunikacji*. URL: <https://docs.ansible.com/ansible/latest/plugins/connection.html#connection-plugins> (term. wiz. 21.01.2022).
- [23] *Understanding the Remote Desktop Protocol (RDP)*. 24 wrz. 2021. URL: <https://docs.microsoft.com/en-us/troubleshoot/windows-server/remote/understanding-remote-desktop-protocol> (term. wiz. 21.01.2022).
- [24] Alex Williamson. *[PATCH] pci: Enable overrides for missing ACS capabilities*. 30 maj. 2013. URL: <https://lkml.org/lkml/2013/5/30/513> (term. wiz. 21.01.2022).

Spis rysunków

1.1	Przypadki użycia aplikacji nadzorczej	15
1.2	Przypadki użycia serwera wirtualizacji	17
1.3	Przypadki użycia panelu administratora	19
2.1	Schematyczna architektura systemu	26
2.2	Schemat klas modelu systemu	27
2.3	Diagram stanów maszyny wirtualnej	30
2.4	Diagram stanów serwera wirtualizacji	31
2.5	Diagram stanów użytkownika	32
2.6	Proces uzyskania sesji	33
2.7	Proces zakończenia sesji	34
2.8	Proces rozpoczęcia pracy serwera wirtualizacji	35
2.9	Endpointy API	37
2.10	Sekwencja komunikacji utworzenia sesji	39
2.11	Sekwencja komunikacji zakończenia sesji	40
2.12	Sekwencja komunikacji aktualizacji stanu systemu	40
2.13	Sekwencja komunikacji włączenia maszyny	41
2.14	Sekwencja komunikacji wyłączenia maszyny	41
2.15	Okno logowania panelu administracyjnego	63
2.16	Widok wszystkich zasobów systemu	63
2.17	Widok szczegółowych zasobów serwera wirtualizacji	64
2.18	Ekran logowania aplikacji klienckiej	65
2.19	Ekran główny aplikacji klienckiej	66
2.20	Ekran ustawień aplikacji klienckiej	67
2.21	Ekran wyboru typu maszyny do pracy	68
2.22	Ekran działającego połączenia RDP	69
2.23	Ekran trwania połączenia w zewnętrznym kliencie RDP	70

Spis tabel

1.1	Przypadki użycia aplikacji nadzorczej	16
1.2	Przypadki użycia serwera wirtualizacji	18
1.3	Przypadki użycia panelu administratora	19
1.4	Wymagania нефункционалне	21
1.5	Analiza ryzyka	22

Spis załączników

1. `REST_API_Overseer.yaml` - dokładna specyfikacja API w standardzie OpenAPI 3.0.3