17.01.2022r.

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

Dokumentacja wdrożeniowa

Autorzy: Krzysztof Smogór, Piotr Widomski
Promotor: Dr inż. Marek Kozłowski

1 Wymagania systemu

System do poprawnej pracy wymaga konfiguracji środowiska na wielu płaszczyznach.

1.1 Komunikacja sieciowa między modułami

System do poprawnego działania musi składać się z:

1. Przynajmniej jednego nadzorcy.
2. Serwerów wirtualizacji (może być ich 0).
3. Maszyn wirtualnych uruchamnych na serwerach wirtualizacji.
4. Serwera HTTP udostępniającego panel administracyjny.
5. Brokera wiadomości do komunikacji wewnętrznej.
6. Brokera wiadomości do komunikacje zewnętrznej (może być tym samym brokerem co wewnętrzny).
7. Dowolnej liczby aplikacji klienckich.

1.1.1 Dostępność wewnętrznego brokera

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

1.1.2 Dostępność zewnętrznego brokera

Broker zewnętrzny powinien być dostępny dla każdego serwera wirtualizacji oraz dla każdego klienta łączącego się z systemem. Jest to wymagane do stwierdzenia czy użytkownik nadal jest podłączony do maszyny wirtualnej.

1.1.3 Dostępność nadzorców

Nadzorcy powinni być dostępni dla aplikacji klienckich, którzy poprzez nich komunikują się z resztą systemu. Administrator po pobraniu panelu administracyjnego z serwera HTTP powinien móc wysyłać zapytania do nadzorców.

1.1.4 Dostępność serwerów wirtualizacji

Serwery wirtualizacji nie muszą być dostępne dla żadnego z modułów.

1.1.5 Dostępność serwera http z panelem administracyjnym

Serwer HTTP powinien być dostępny dla każdego z administratorów.

1.1.6 Dostępność maszyn wirtualnych

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

1.2 Wymagania aplikacji klienckiej

Dla systemu Windows aplikacja kliencka jako plik wykonywalny pobrana z oficjalnych wydań 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. Pozostałe zależności są dostarczone wewnątrz pliku `.appimage`. Działanie aplikacji było testowane na dystrybucjach bazowanych na Debianie oraz ArchLinuxie. Dla innych dystrybucji aplikacja może nie działać prawidłowo.

1.3 Wymagania aplikacji nadzorcy i serwera panelu administracyjnego

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.

1.4 Wymagania serwera wirtualizacji

Serwer wirtualizacji oprócz działającej usługi `Dockera` potrzebuje dodatkowych usług. Do prawidłowego działania wymagana jest działająca usługa zarządcy wirtualnych maszyn `libvirt`. Aby prawidłowo uruchomić maszynę wirtualną potrzebna jest uruchomiona usługa zapory sieciowej oraz pakiet `dnsmasq`. Wymagana jest także inicjalizacja 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 potrzeba utworzyć interfejs sieciowy 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 uruchomiania maszyny wirtualnej przy użyciu `vagranta` oraz uruchomiania wybranych kontenerów w `dockerze` komunikacja sieciowa z maszyną wirtualną poprzez podłączony interfejs w trybie bridge może zostać ograniczona. Należy wtedy dopilnować by w trakcie działanie systemu zapora sieciowa posiadała zasadę na samej górze łańcucha `FORWARD`, która będzie bezwarunkowo zezwalać trasować pakiety z urządzeń w trybie bridge.

1.5 Automatyzacja konfiguracji

Przykładowa konfiguracja oraz narzędzia do automatycznej konfiguracji przed uruchomieniem dostępne sa w module `configuration`.

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.

1.5.1 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.

1.5.2 Zmienne dla serwera wirtualizacji

Playbook dla serwera wirtualizacji wykona wszystkie kroki opisane w 1.4. Aby tego dokonać należy zdefiniować dane dla tworzonego interfejsu typu bridge. Należy zdefiniować nazwę interfejsu sieciowego, który zostanie połączony do nowo tworzonego urządzenia bridge o nazwie zdefiniowanej w pliku. Playbook korzysta z NetworkManagera do zmiany konfiguracji, zatem trzeba podać nazwę *connection* (jednostka logiczna w NetworkManagerze) skojarzonego z początkowo wybranym interfejsem sieciowym.

1.5.3 Zmienne dla nadzorcy

Aby uruchomić nadzorcę wystarczą wspólne wymagania dla wszystkich grup.

1.6 Wymagania szablonu maszyny wirtualnej

Maszyna wirtualna uruchamiana w ramach systemu OneClickDesktop musi być w postaci Vagrant Boxa. Przykładowy box został utworzony w czasie rozwoju systemu i dostępny jest w chmurze Vagrant pod nazwą smogork/archlinux-rdp. 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 jest potrzebne jest utworzenie niestandardowego szablonu to musi on:

- Spełniać minimalne wymagania opisane w dokumentacji Vagranta.
- Mieć zainstalowany pewien menadżer okienek. Przykładowy szablon zawiera menadżera okienek XFCE.
- Przy uruchomieniu udostępniać usługę zdalnego pulpitu RDP. Przykładowy szablon korzysta z implementacji xrdp.

- Każdy nowy interfejs sieciowy powinien być tak skonfigurowany, aby uzyskać adres IP z usługi DHCP.
- Przy starcie systemu urchamiać usługę `qemu-guest-agent`.

Nie ma ograniczenia na system operacyjny uruchamiany wewnątrz systemu OneClickDesktop. Jedynie taki szablon musi spełniać powyższe wymagania.

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

2 Uruchomienie systemu

W tym podrozdziale zakładamy, że każdy system operacyjny został skonfigurowany poprawnie zgodnie z opisem w 1. Wtedy można przejść do uruchamiania systemu.

2.1 Pozyskanie aplikacji klienckiej

Zalecanym sposobem pozyskania aplikacji klienckiej jest udanie się do sekcji z wydaniami kodu 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.

2.2 Zbudowanie kontenerów

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

2.2.1 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`.

2.2.2 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 `one-click-desktop/virtualization-server-runtime`. Kontener zawierający aplikacje w czasie budowania będzie oczekiwał, ze taki obraz istnieje.

Po zbudowaniu kontenera z zależnościami można przystąpić do zbudowania 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`.

2.2.3 Budowanie kontenera nadzorcy i panelu administratora

W przypadku tych dwóch modułów nie trzeba wykonać żądnych 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`.

2.3 Budowanie modułów

Każdy z modułów posiada dokładną instrukcję budowania w pliku `README.md`. Uruchamianie zbudowanych modułów poleca się tylko w przypadku rozwoju aplikacji. Przy wdrażaniu systemu najszybciej i najbezpieczniej jest skorzystać z metod opisanych w 2.1 i 2.2.

2.3.1 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 1.4, zainstalowany vagrant wraz z wtyczka vagrant-libvirt w wersji przynajmniej 0.7.0.

2.3.2 Moduły napisane w technologii Node

W przypadku budowania modułów aplikacji klienckiej oraz panelu administracyjnego potrzebny będzie pakiet Node. Przed zbudowaniem aplikacji należy pobrać wszystkie użyte biblioteki przy użyciu menadżera pakietów npm. Są one zdefiniowane w projekcie aplikacji i zostaną pobrane automatycznie.

2.4 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(`basePath`) - adres musi być w formacie URI.
- Adres zewnętrznego brokera wiadomości(`rabbitPath`) - adres musi być w formacie 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 innego niż zalecany klient RDP. Aplikacja po uzyskaniu sesji wyświetli dane dostępowe do przypisanej maszyny wirtualnej.

2.5 Konfiguracja panelu administracyjnego

Przy uruchomieniu panelu administracyjnego trzeba przekazać adres jednego z nadzorców. Aby tego dokonać należy ustawić zmienną środowiskową `API_URL` na adres dostępowy jednego z nadzorców.

2.6 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 mu 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 uruchomienia `-c path/to/other/config/folder/`.

W pliku konfiguracyjnym nadzorca poszukuje 2 specjalnych sekcji:

- `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. Jednak warte uwagi są standardowe sekcje:

- `Logging` - parametry loggera dostarczonego przez Microsoft.
- `Kestrel` - parametry serwera HTTP udostępniającego aplikację. W wypadku aplikacji nadzorcy bez ustawienia certyfikatu nie zadziała ona w trybie HTTPS.

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.

2.6.1 Parametry sekcji `JwtSettings`

Sekcja ta zawiera jedynie parametr `Secret`. Należy go ustawić na dowolny ciąg znaków.

2.6.2 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 wewnętrznego 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 aktualizacje modelu. Domyślnie przyjmuje wartośc `60`. Zalecane jest aby wartość tego parametru opisywała czas od 30 do 60 sekund.

- `DomainShutdownTimeout` - ilość minut ile 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.

2.7 Konfiguracja serwera wirtualizacji

Serwer wirtualizacji posiada wiele parametrów związanych z zasobami przekazanymi do dyspozycji systemu. Wszystkie parametry należy przekazać mu poprzez pliki konfiguracyjne.

Pliki te muszą znaleźć 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 udostępnionych zasobach 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 on się znajdować w tym samym folderze co aplikacja oraz nazywać się `NLog.config`.

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

2.7.1 Główny plik konfiguracyjny

W głównym pliku konfiguracyjnym możemy wyróżnić 2 sekcje:

- `OneClickDesktop` - przechowuje parametry związane z działaniem systemu OneClickDesktop.
- `ServerResources` - przechowuje parametry określające zgrubnie 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`.
- `OversserCommunicationShutdownTimeout` - 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 2.6.
- `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 1.6. 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. Domyslnie przyjmuje wartośc `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. Domyslnie przyjmuje wartośc `0`.
- `MachineTypes` - lista typów maszyn wirtualnych. Każda nazwa typu jest oddzielona od sąsiednich przecinkiem. Nazwa typu musi składać się ze znaków opisanych nastepującym wyrażeniem regularnym `[a-zA-Z0-9\-]`. Dla każdej z nazw wyszukiwany jest plik konfiguracyjny o o nazwie zaczynającej się nazwą typu i kończącą się `_template.ini`.

Gdy $n =$ `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.

Każda z tych sekcji zawiera parametr `AddressCount`, który określa jak duża jest grupa IOMMU w której znajduje się przekazywane urządzenie PCI. W zależności $m =$ `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 sekcja zasobów z jednym GPU przekazanym do systemu:

```
[ServerResources]
Cpus=6
Memory=4096
```

```
Storage=200
GPUsCount=1
MachineTypes=cpu,gpu

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

2.7.2 Pliki konfiguracyjne opisujące typy maszyn wirtualnych

W pliku opisującym typy zawarte jest ile potrzeba zasobów aby uruchomić maszynę wytworzoną z tego typu.

Każdy typ ma przypisaną nazwę. Oznaczmy ją jako `${template_name}`. Plik opisujący typ musi mieć nazwę `${template_name}_template.ini`. Plik konfiguracyjny dla wybranego typu musi znajdować się w folderze z pozostałymi plikami konfiguracyjnymi. Jeżeli dla jakiejkolwiek 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`. Zwiera ona parametry:

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

2.8 Procedura uruchomienia

Prawidłowy start systemu powinien zachować następująca kolejność:

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 podłączenia się przez użytkowników. Ważne jest aby przed uruchomieniem jakiegokolwiek nadzorcy brokery wiadomości były już prawidłowo zainicjalizowane. W przeciwnym wypadku aplikacja nadzorcy zakończy się z błędem. Serwer wirtualizacji także zakończy się z błędem, jeżeli zabraknie brokerów wiadomości. Dodatkowo, jeżeli nie będzie żadnego nadzorcy nasłuchującego poprzez wewnętrznego brokera wiadomości, serwer wirtualizacji zgłosi błąd i zakończy pracę.

2.9 Parametry uruchomienia kontenera panelu administracyjnego

Kontener zawierający panel administratory udostępnia aplikacje poprzez serwer HTTP nginx. Do komunikacji wystawia on porty 80 (HTTP) oraz 443(HTTPS). Należy przekierować je na odpowiednie porty uruchamiającego systemu.

2.9.1 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 2.5.

2.9.2 Certyfikat SSL

W celu uruchomienia panelu w trybie HTTPS należy przekazać zmienioną konfigurację serwera nginx oraz parę klucza z certyfikatem pod postacią woluminu.

```
-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` praz `PATH_TO_KEY_IN_CONTAINER` są ścieżkami bezwzględnymi w kontenerze podanymi w przyłączonej konfiguracji.

2.10 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.

2.10.1 Plik konfiguracyjny

Aplikacja nadzorcy poszukuje plików konfiguracyjny w lokalizacji `/overseer/config/`. 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}:/overseer/config
```

2.10.2 Certyfikat SSL

W celu uruchomienia nadzorcy w trybie HTTPS należy przekazać certyfikat SSL w formacie `.pfx` przy pomocy woluminów oraz odpowiednio przygotowany plik konfiguracyjny tak jak opisano w 2.6. Gdy zabraknie certyfikatów a konfiguracja pokazuje, że ma się rozpocząć nasłuchiwanie w trybie HTTPS, nadzorca 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 podaną w konfiguracji.

2.11 Parametry uruchomienia kontenera serwera wirtualizacji

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

2.11.1 Zasoby libvirta

Aby komunikować się z usługa libvirta uruchomioną na systemie uruchamiającym potrzebujemy gniazda sieciowego przekazanego do wnętrza kontenera. Musi ono udawać, że jest uruchomiona usługa libvirta we wnętrzu kontenera. Dodatkowo nowo tworzone maszyny powinny zapisywać się pomiędzy uruchomieniami kontenera. W tym celu trzeba przekazać także cały folder z danymi libvirta. Tai 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/
```

2.11.2 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 przestrzec 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.

2.11.3 Plik konfiguracyjny

Aplikacja serwera wirtualizacji poszukuje plików konfiguracyjny 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 lokalizacje folderu z konfiguracja ustawiając zmienna środowiskową `CONFIG`. Może być ona względna do ścieżki `/app`.

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

2.12 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`. Znajduje się tam system składający się z jednego brokera wiadomości (zewnętrzna i wewnętrzna komunikacja w jednym), panelu administracyjnego, jednego nadzorcy i jednego serwera wirtualizacji. Można znaleźć tam przykłady zarówno konfiguracji modułów jak i parametrów startowych dla kontenerów.