Przedmiot: Technologie chmury obliczeniowej	Data: 11-01-2025
Przykład wykorzystania mikroserwisu dla realizacji usługi ftp	
Wersja doc ver 1.2	Wykonawcy: Kamil Kłak

Uwagi do projektu	
1 Wstęp	[data] Wypełnia prowadzący projekt
2 Cześć teoretyczna	[data] Wypełnia prowadzący projekt
3 Część praktyczna	[data] Wypełnia prowadzący projekt
4 Dokumentacja wykonania części praktycznej	[data] Wypełnia prowadzący projekt
5 Wnioski końcowe	[data] Wypełnia prowadzący projekt

1 Wstęp
Projekt przedstawia wykorzystanie mikroserwisów dla realizacji ftp poprzez plik docker-compose, oraz krótkiej konfiguracji/usprawnienia serwisu, dzięki vagrantowi i virtualbox. Celem projektu jest uzyskanie działającego serwera ftp z użytkownikami oraz folderami które będą miały skonfigurowane odpowiednie uprawnienia.

2 Cześć teoretyczna
Opis zagadnień poruszanych w projekcie:

Mikroserwisy
Mikroserwisy to architektura oprogramowania, w której aplikacje są budowane jako zbior zestaw niezależnych modułów, z każdym modułem realizującym pojedynczą funkcjonalność. Każdy mikroserwis jest autonomiczny, może być wdrażany niezależnie i komunikuje się z innymi mikroserwisami za pomocą lekkich protokołów, takich jak HTTP czy gRPC.
Pozwalają one na:

- Skalowalność: możliwość skalowania poszczególnych modułów w zależności od obciążenia.
- Niezależne wdrażanie i rozwój: każdy mikroserwis może być rozwijany i aktualizowany niezależnie od innych.
- Lepsze zarządzanie kodem: podział aplikacji na mniejsze, bardziej przejrzyste moduły.

Protokół FTP (File Transfer Protocol)
FTP to protokół służący do przesyłania plików w sieciach komputerowych opartych na TCP/IP. Używany jest głównie do:

- Przesyłania plików między serwerem a klientem.
- Zarządzania plikami na serwerze (tworzenie katalogów, usuwanie plików itp.).

FTP działa w modelu klient-serwer, gdzie klient inicjuje połączenie z serwerem, a komunikacja odbywa się za pomocą dwóch kanałów:

1. **Kanał sterujący** - służy do przesyłania poleceń i odpowiedzi.
2. **Kanał danych** - służy do przesyłania faktycznych plików.

Konteneryzacja za pomocą Dockera
Docker to narzędzie umożliwiające tworzenie, wdrażanie i uruchamianie aplikacji w odizolowanych środowiskach zwanych kontenerami. Kontenery zawierają wszystkie niezbędne zależności, co sprawia, że aplikacje są przenośne i działają w taki sam sposób na różnych systemach.

Docker Compose to narzędzie pozwalające na definiowanie i uruchamianie wielokontenerowych aplikacji za pomocą plików YAML. W projekcie został użyty Docker Compose do:

- Zdefiniowania konfiguracji serwera FTP.
- Automatycznego uruchamiania usługi FTP w odizolowanym kontenerze.
- Zarządzania siecią i zasobami dla kontenera.

2. Wyjaśnienie zasad teoretycznych poruszanych w projekcie

W projekcie dotyczącym realizacji usługi FTP za pomocą mikroserwisu w kontenerze Docker poruszane są następujące zasady teoretyczne:

2.1 Mikroserwisy

Mikroserwisy to architektura oprogramowania, w której aplikacja jest dzielona na niezależne, odseparowane komponenty (serwisy). Każdy mikroserwis:

- Realizuje określoną funkcjonalność (np. obsługa FTP).
- Komunikuje się z innymi mikroserwisami za pomocą protokołów sieciowych (np. HTTP, gRPC).
- Może być niezależnie rozwijany, wdrażany i skalowany.

Architektura mikroserwisów pozwala na lepsze zarządzanie złożonymi aplikacjami, ułatwia wdrażanie zmian oraz poprawia niezawodność całego systemu poprzez izolację błędów.

2.2 Protokół FTP (File Transfer Protocol)

FTP to protokół sieciowy służący do przesyłania plików między komputerami w sieci. Charakteryzuje się:

- Modelem klient-serwer, gdzie klient łączy się z serwerem w celu przesyłania plików.
- Użyciem dwóch kanałów komunikacyjnych:
  - Kanału sterującego (kontrolnego) do przesyłania poleceń i odpowiedzi.
  - Kanału danych do przesyłania plików.
- Wsparciem dla trybów aktywnego i pasywnego połączenia.

FTP jest szeroko stosowany w zarządzaniu plikami, choć jego brak szyfrowania stanowi wadę w kontekście bezpieczeństwa. Dlatego też w praktyce często stosuje się jego ulepszoną wersję FTPS lub alternatywę w postaci SFTP.

2.3 Konteneryzacja

Konteneryzacja to technologia pozwalająca na uruchamianie aplikacji w odizolowanych środowiskach zwanych kontenerami. Kontenery:

- Działają na wspólnym jądrze systemu operacyjnego, co zmniejsza ich zasobożerność w porównaniu z maszynami wirtualnymi.
- Zawierają wszystkie zależności aplikacji, co gwarantuje spójność środowiska uruchomieniowego.

Konteneryzacja ułatwia:

- Wdrażanie aplikacji w różnych środowiskach.
- Skalowanie poziome (dodawanie więcej instancji aplikacji).
- Testowanie i rozwój aplikacji w izolowanych środowiskach.

2.4 Docker i Docker Compose

Docker to platforma do konteneryzacji, która umożliwia:

- Tworzenie obrazów kontenerów.

- Zarządzanie cyklem życia kontenerów.

Docker Compose to narzędzie pozwalające na definiowanie i uruchamianie aplikacji składających się z wielu kontenerów za pomocą pliku YAML. Umożliwia:

- Określenie konfiguracji sieciowej i zależności między kontenerami.
- Automatyzowanie uruchamiania całej aplikacji.

2.5 Wirtualizacja sieci

Wirtualizacja sieci w kontekście Dockera polega na tworzeniu izolowanych sieci dla kontenerów, co:

- Pozwala na separację ruchu sieciowego między aplikacjami.
- Umożliwia konfigurację wirtualnych podsieci, mostów i translacji adresów (NAT).

Podsumowanie

Wyższej wymienione zasady teoretyczne stanowią fundament dla realizacji projektu mikroserwisu obsługującego FTP w kontenerze Docker. Ich zrozumienie jest kluczowe dla efektywnego wdrażania i zarządzania podobnymi systemami.

3.Opis zastosowanych narzędzi, technologii i modeli

W ramach projektu wykorzystano następujące narzędzia, technologie i modele:

Docker

Docker to platforma do konteneryzacji, która pozwala na tworzenie, wdrażanie i uruchamianie aplikacji w izolowanych środowiskach. Kontenery umożliwiają łatwe przenoszenie aplikacji między różnymi środowiskami oraz zapewniają spójność działania niezależnie od konfiguracji systemu operacyjnego. W projekcie Docker został użyty do uruchomienia serwera FTP w odizolowanym środowisku.

Docker Compose

Docker Compose to narzędzie umożliwiające definiowanie i uruchamianie aplikacji wielokontenerowych. Składnia plików YAML pozwala na łatwe określenie konfiguracji usług, sieci i wolumenów. W projekcie Docker Compose został wykorzystany do automatyzacji procesu uruchamiania serwera FTP wraz z niezbędnymi zależnościami.

FTP (File Transfer Protocol)

FTP to protokół sieciowy służący do przesyłania plików pomiędzy urządzeniami w sieci. W projekcie został wykorzystany do stworzenia usługi umożliwiającej użytkownikom przesyłanie i pobieranie plików. Protokół działa w modelu klient-serwer i opiera się na dwóch kanałach komunikacyjnych: kontrolnym i danych.

Modele komunikacji klient-serwer

Model klient-serwer to podstawowy wzorzec architektury sieciowej, w którym klient wysyła żądania do serwera, a serwer przetwarza te

żądania i zwraca odpowiedzi. W projekcie model ten został zastosowany w kontekście usługi FTP, gdzie serwer odpowiada za zarządzanie dostępem do plików, a klient inicjuje transfery.
Każde z tych narzędzi i technologii zostało wybrane ze względu na ich popularność, wsparcie społeczności oraz łatwość integracji w środowiskach wirtualizacyjnych, takich jak Docker.

4.Wyjaśnienie wyboru zastosowanych narzędzi, technologii, modeli, protokołów, urządzeń, środowiska wirtualizacji/emulatora oraz programów/aplikacji:

1. **Docker Compose**: Został wybrany ze względu na łatwość zarządzania wieloma kontenerami jednocześnie. W projekcie Docker Compose pozwala na skonfigurowanie serwera FTP oraz ewentualnych zależności w jednym pliku YAML, co upraszcza proces wdrażania i utrzymania aplikacji.
2. **Docker**: Użycie Dockera zapewnia izolację środowiskową, co jest kluczowe w przypadku mikroserwisów. Kontenery Dockera są lekkie i przenośne, co ułatwia ich wdrażanie na różnych platformach.
3. **alpine-ftp-server** Użycie właśnie tego wariantu serwera było wyborem ze względu na jego prostą budowę oraz instalacje dzięki czemu maszyna nie ma wysokich wymagań sprzętowych oraz jest prosta w obsłudze.
4. **Protokół FTP**: FTP został wybrany jako przykład prostego, ale skutecznego protokołu do przesyłania plików. Jego implementacja w projekcie pozwala na zrozumienie zasad działania protokołów sieciowych oraz ich integracji z mikroserwisami.
5. **Model klient-serwer**: Ten model komunikacji został zastosowany, ponieważ jest zgodny z architekturą protokołu FTP i dobrze ilustruje interakcje między klientem a serwerem.
6. **Środowisko wirtualizacji/emulator**: Docker pełni rolę środowiska wirtualizacji, umożliwiając uruchamianie kontenerów z serwerem FTP w sposób wydajny i bezpieczny.
7. **Programy/aplikacje**: W projekcie użyto narzędzi takich jak FileZilla (klient FTP) do testowania funkcjonalności serwera FTP oraz Docker CLI do zarządzania kontenerami. Wybór tych narzędzi wynika z ich popularności, łatwości użycia i zgodności z projektem.

3 Część praktyczna

1.Opis zastosowanego urządzeń, środowiska wirtualizacji/emulatora czy aplikacji w projekcie

VirtualBox

VirtualBox został użyty jako platforma wirtualizacyjna do uruchomienia środowiska laboratoryjnego. Dzięki temu możliwe było odizolowanie projektu od głównego systemu operacyjnego i zapewnienie pełnej kontroli nad konfiguracją środowiska.

Docker

Docker, działający w środowisku VirtualBox, umożliwił konteneryzację aplikacji, co zapewniło łatwość wdrażania i izolację usług.

Alpine-FTP-Server

Alpine-FTP-Server został wykorzystany jako obraz kontenera do uruchomienia serwera FTP. Jest to rozwiązanie lekkie, wydajne i łatwe w konfiguracji, co czyni je idealnym do demonstracji działania usługi FTP w środowisku kontenerowym.

2.Opis przeprowadzonych badań, eksperymentów, konfiguracji mającej na celu prawidłowe wykonane i działanie projektu:

Przygotowanie środowiska

1. Zainstalowano VirtualBox i uruchomiono na nim maszynę wirtualną z zainstalowanym Dockerem.
2. Pobranie obrazu Alpine-FTP-Server z repozytorium Dockera.

Konfiguracja serwera FTP

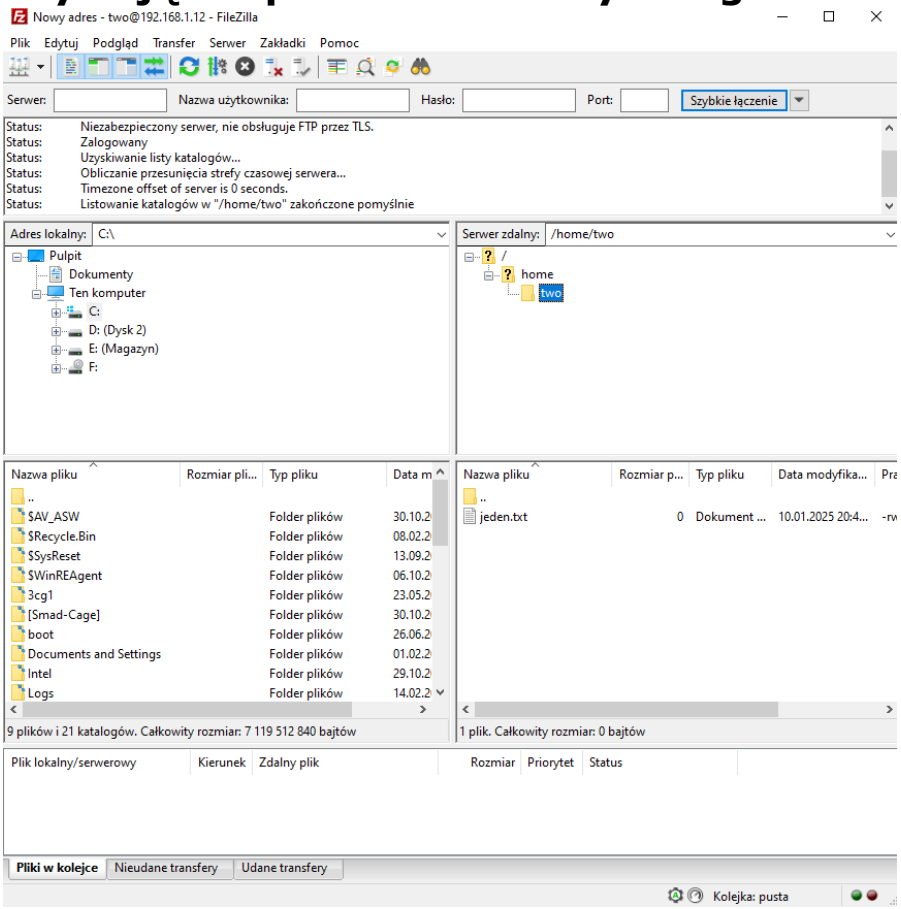
1. Utworzono plik docker-compose.yml zawierający konfigurację serwera FTP, w tym:
   - Definicję obrazu Alpine-FTP-Server.
   - Konfigurację portów i woluminów.
2. Utworzono foldery do przechowywania danych FTP.
3. Nadano odpowiednie uprawnienia do folderów.

Testowanie

1. **Uruchomiono kontener za pomocą polecenia docker-compose up -d.**

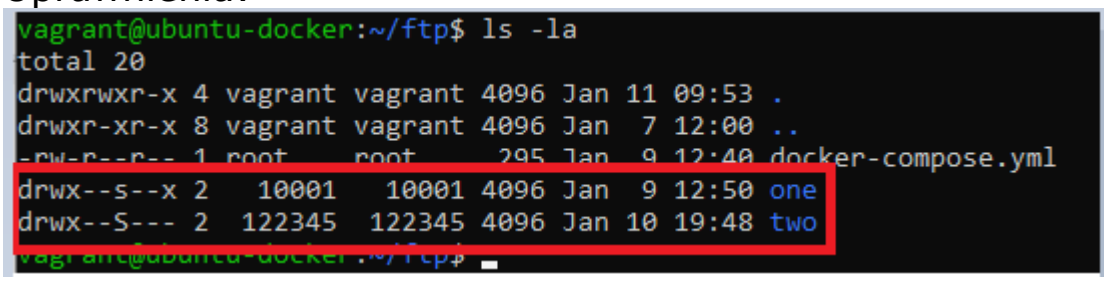
```
vagrant@ubuntu-docker:~/ftp$ vagrant@ubuntu-docker:~/ftp$ ls -l
total 12
-rw-r--r-- 1 root   root    295 Jan  9 12:40 docker-compose.yml
drwx--s--x 2  10001  10001 4096 Jan  9 12:50 one
drwx--S--- 2 122345 122345 4096 Jan 10 19:48 two
vagrant@ubuntu-docker:~/ftp$ docker-compose up -d
ftp_ftp_1 is up-to-date
vagrant@ubuntu-docker:~/ftp$
```

2. **Połączono się z serwerem FTP za pomocą FileZilla, używając odpowiednich danych logowania.**



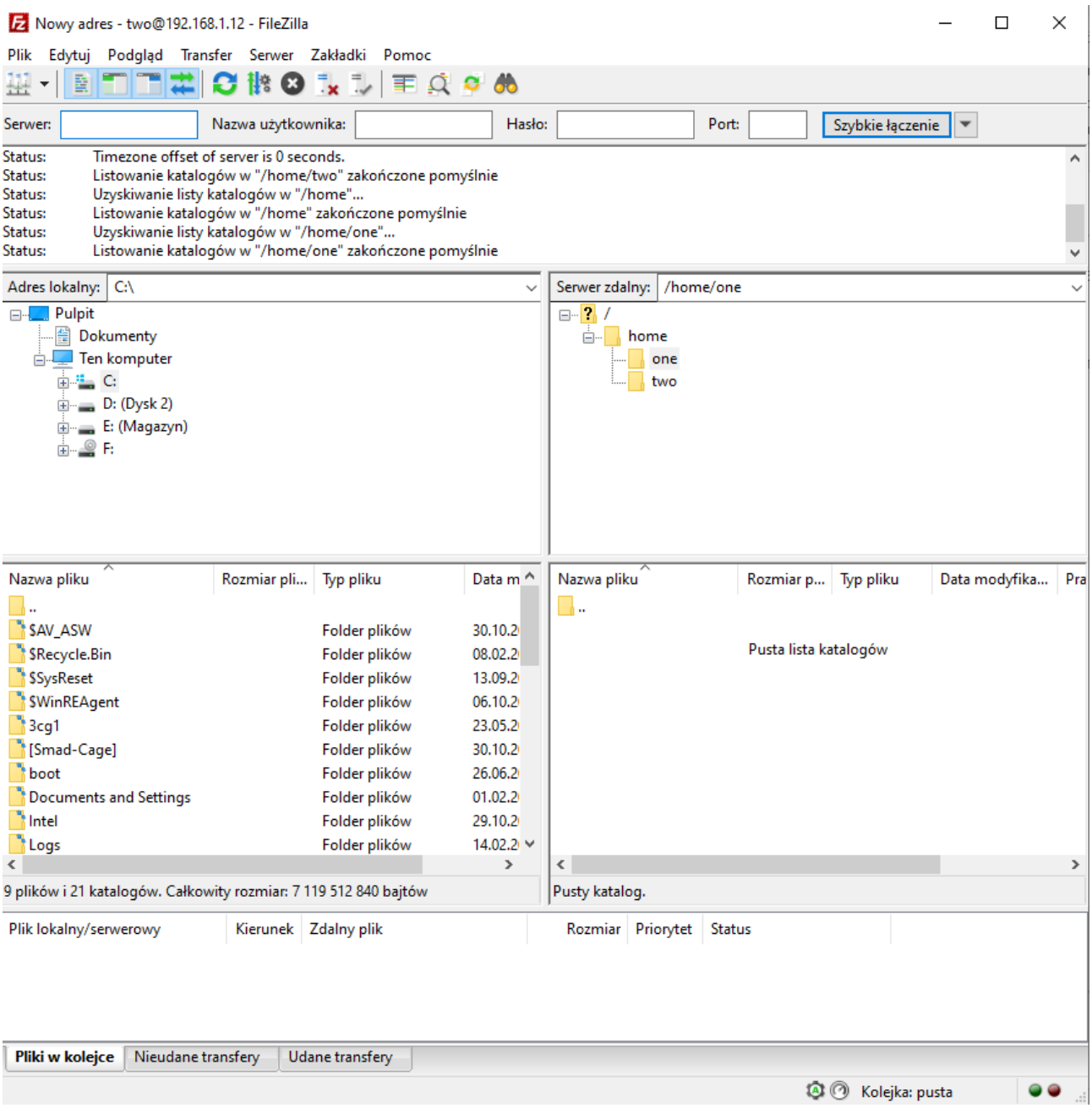
3. **Przeprowadzono testy przesyłania plików, weryfikując poprawność działania serwera. Sprawdzono również zastosowane zabezpieczenia folderów użytkowników. (użytkownik one nie ma dostępu do foldera użytkownika two; A użytkownik two nie widzi plików w katalogu użytkownika one).**

Uprawnienia:

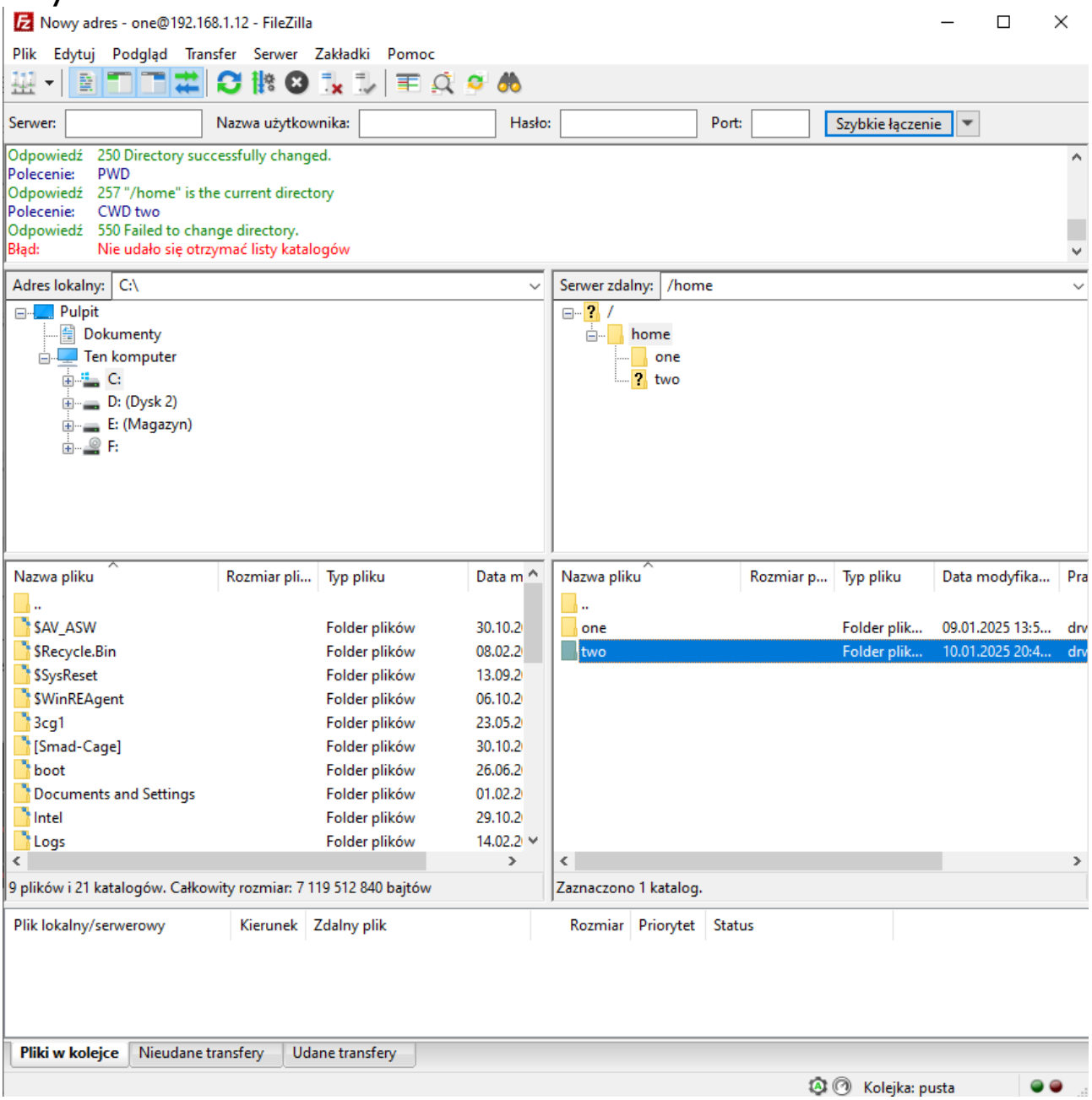


Widok w filezilla:

Użytkownik two:


Nowy adres - two@192.168.1.12 - FileZilla
Plik Edytuj Podgląd Transfer Serwer Zakładki Pomoc
Serwer: Nazwa użytkownika: Hasło: Port: Szybkie łączenie
Status: Timezone offset of server is 0 seconds.
Status: Listowanie katalogów w "/home/two" zakończone pomyślnie
Status: Uzyskiwanie listy katalogów w "/home"...
Status: Listowanie katalogów w "/home" zakończone pomyślnie
Status: Uzyskiwanie listy katalogów w "/home/one"...
Status: Listowanie katalogów w "/home/one" zakończone pomyślnie
Adres lokalny: C:\
Pulpit
Dokumenty
Ten komputer
C:
D: (Dysk 2)
E: (Magazyn)
F:
Serwer zdalny: /home/one
/
home
one
two
Nazwa pliku Rozmiar pli... Typ pliku Data m
$AV_ASW Folder plików 30.10.2
$Recycle.Bin Folder plików 08.02.2
$SysReset Folder plików 13.09.2
$WinREAgent Folder plików 06.10.2
3cg1 Folder plików 23.05.2
[Smad-Cage] Folder plików 30.10.2
boot Folder plików 26.06.2
Documents and Settings Folder plików 01.02.2
Intel Folder plików 29.10.2
Logs Folder plików 14.02.2
9 plików i 21 katalogów. Całkowity rozmiar: 7 119 512 840 bajtów
Nazwa pliku Rozmiar p... Typ pliku Data modyfika...
Pusta lista katalogów
Pusty katalog.
Plik lokalny/serwerowy Kierunek Zdalny plik Rozmiar Priorytet Status
Pliki w kolejce Nieudane transfery Udane transfery
Kolejka: pusta

Użytkownik one:



Dodatkowe modyfikacje

1. Dodano uprawnienia do folderów w celu poprawy bezpieczeństwa i funkcjonalności.
2. Wprowadzono zmiany w konfiguracji docker-compose.yml, aby dostosować środowisko do wymagań projektu.

3. Dokumentacja wykonania części praktycznej:

Lista zawierających nazwy plikow z konfiguracja oraz krótkim opisem z przeznaczenia:
ftp/Docker-compose.yml – Główny plik konfiguracyjny, dzięki niemu serwer ftp jest postawiony w kontenerze gdzie znajduja się foldery użytkowników, prawie całą konfiguracja jest w nim zapisana.

ftp/one – folder użytkownika one z skonfigurowanymi uprawnieniami dostępu drwx—s--x
ftp/two - folder użytkownika two z skonfigurowanymi uprawnieniami dostępu drwx—S---

Poniżej są screeny z większości zabiegów przeprowadzonych na maszynie wirtualnej:

```
vagrant@ubuntu-docker:~$ docker pull delfer/alpine-ftp-server
Using default tag: latest
latest: Pulling from delfer/alpine-ftp-server
4abcf2066143: Pull complete
70fa23ad9f28: Pull complete
dbda407a11d5: Pull complete
5fabc0a3b2f5: Pull complete
c5005f1f1199: Pull complete
Digest: sha256:8f63d017e25ac17fb1b5c109fff82b8bb88e42f33ea81c0a4849f955b521e87d
Status: Downloaded newer image for delfer/alpine-ftp-server:latest
docker.io/delfer/alpine-ftp-server:latest
```

```
vagrant@ubuntu-docker:/$ cd vagrant
vagrant@ubuntu-docker:/vagrant$ ls -l
total 28
-rwxrwxrwx 1 vagrant vagrant  1001 Sep 25  2023 Vagrantfile
-rwxrwxrwx 1 vagrant vagrant 43048 Jan  7 11:47 ubuntu-bionic-18.04-cloudimg-console.log
vagrant@ubuntu-docker:/vagrant$ cd ~
vagrant@ubuntu-docker:~$ ls -l
total 20
-rw-rw-r-- 1 vagrant vagrant  405 Nov  3 14:54 docker-compose-drupal_with_postesql.yml
-rw-r--r-- 1 root    root     246 Jan  7 11:58 docker-compose.yml
-rw-rw-r-- 1 vagrant vagrant   69 Nov  3 14:27 dockerfile
drwxrwxr-x 2 vagrant vagrant 4096 Jan  7 12:00 ftp
-rw-rw-r-- 1 vagrant vagrant 1264 Nov  3 14:24 index.html
vagrant@ubuntu-docker:~$ cd ftp
vagrant@ubuntu-docker:~/ftp$ sudo nano docker-compose.yml
vagrant@ubuntu-docker:~/ftp$ docker-compose up -d
Creating ftp_ftp_1 ... done
vagrant@ubuntu-docker:~/ftp$ docker-compose ps
  Name                Command              State   Ports
-------------------------------------------------------
ftp_ftp_1   /sbin/tini -- /bin/start_v ...   Up
vagrant@ubuntu-docker:~/ftp$ docker-compose logs -f --tail 100
Attaching to ftp_ftp_1
ftp_1  | Changing password for one
ftp_1  | New password:
ftp_1  | Bad password: too short
ftp_1  | Retype password:
ftp_1  | passwd: password for one changed by root
ftp_1  | Changing password for two
ftp_1  | New password:
ftp_1  | Bad password: too short
ftp_1  | Retype password:
ftp_1  | passwd: password for two changed by root
ftp_1  | pidfd_open syscall is not supported, falling back to polling
ftp_1  | failed to watch for direct child exit (pidfd_open error): Function not implemented
```

```
vagrant@ubuntu-docker:~$ ls -l
total 20
-rw-rw-r-- 1 vagrant vagrant  405 Nov  3 14:54 docker-compose-drupal_with_postesql.yml
-rw-r--r-- 1 root    root     246 Jan  7 11:58 docker-compose.yml
-rw-rw-r-- 1 vagrant vagrant   69 Nov  3 14:27 dockerfile
drwxrwxr-x 4 vagrant vagrant 4096 Jan  7 12:07 ftp
-rw-rw-r-- 1 vagrant vagrant 1264 Nov  3 14:24 index.html
vagrant@ubuntu-docker:~$ cd ftp
vagrant@ubuntu-docker:~/ftp$ ls -l
total 12
-rw-r--r-- 1 root    root     259 Jan  7 12:06 docker-compose.yml
drwxr-sr-x 2 vagrant vagrant 4096 Jan  7 12:14 one
drwxr-sr-x 2 ubuntu  ubuntu  4096 Jan  7 12:07 two
vagrant@ubuntu-docker:~/ftp$ sudo nano docker-compose.yml
vagrant@ubuntu-docker:~/ftp$ vagrant@ubuntu-docker:~/ftp$ cd /home
vagrant@ubuntu-docker:/home$ ls -l
total 8
drwxr-xr-x 3 ubuntu  ubuntu  4096 Oct 26 12:54 ubuntu
drwxr-xr-x 8 vagrant vagrant 4096 Jan  7 12:00 vagrant
vagrant@ubuntu-docker:/home$ cd ~
vagrant@ubuntu-docker:~$ cd ftp
vagrant@ubuntu-docker:~/ftp$ sudo nano docker-compose.yml
vagrant@ubuntu-docker:~/ftp$ vagrant@ubuntu-docker:~/ftp$ docker-compose down && docker-compose up -d
Stopping ftp_ftp_1 ... done
Removing ftp_ftp_1 ... done
Creating ftp_ftp_1 ... done
vagrant@ubuntu-docker:~/ftp$ ls -la
total 20
drwxrwxr-x 4 vagrant vagrant 4096 Jan  9 12:40 .
drwxr-xr-x 8 vagrant vagrant 4096 Jan  7 12:00 ..
-rw-r--r-- 1 root    root     295 Jan  9 12:40 docker-compose.yml
drwxr-sr-x 2   10001   10001 4096 Jan  7 12:14 one
drwxr-sr-x 2  122345  122345 4096 Jan  7 12:07 two
vagrant@ubuntu-docker:~/ftp$ sudo chmod -R go-rw-r--r--  one/
vagrant@ubuntu-docker:~/ftp$ sudo chmod -R go-rx  two/
vagrant@ubuntu-docker:~/ftp$ ls -la
total 20
drwxrwxr-x 4 vagrant vagrant 4096 Jan  9 12:40 .
drwxr-xr-x 8 vagrant vagrant 4096 Jan  7 12:00 ..
-rw-r--r-- 1 root    root     295 Jan  9 12:40 docker-compose.yml
drwx--s--x 2   10001   10001 4096 Jan  7 12:14 one
drwx--S--- 2  122345  122345 4096 Jan  7 12:07 two
```

```
  GNU nano 2.9.3

version: '2.1'
services:
  ftp:
    image: delfer/alpine-ftp-server
    network_mode: host
#    ports:
#      - 21:21
#      - 21000-21010:21000-21010
    environment:
      - USERS=one|1234|/home/one|10001 two|12345|/home/two|122345
    volumes:
      - ./one:/home/one
      - ./two:/home/two
```

5 Wnioski końcowe

Projekt dotyczący realizacji usługi FTP przy użyciu mikroserwisów oraz konteneryzacji w środowisku Docker pozwolił na osiągnięcie kilku kluczowych celów:

1. **Zastosowanie mikroserwisów** – Projekt pokazał, jak mikroserwisy mogą być wykorzystane do realizacji konkretnej

funkcjonalności, jaką jest serwer FTP. Podział na niezależne komponenty ułatwił zarządzanie i rozwój aplikacji, a także pozwolił na łatwe skalowanie poszczególnych usług w przyszłości.

2. **Docker** – Docker okazał się idealnym narzędziem do konteneryzacji serwera FTP. Dzięki temu projekt stał się bardziej przenośny i łatwiejszy w utrzymaniu. Konteneryzacja pozwoliła na szybkie wdrożenie usługi w różnych środowiskach, a także zapewniła łatwą izolację poszczególnych komponentów.
3. **Automatyzacja konfiguracji za pomocą Docker Compose** – Użycie Docker Compose do konfiguracji środowiska umożliwiło szybkie uruchomienie całego systemu. Dzięki plikowi YAML możliwe było łatwe zdefiniowanie wszystkich zależności, portów i woluminów, co zminimalizowało czas konfiguracji i błędy ludzkie.
4. **Bezpieczeństwo i zarządzanie użytkownikami** – Projekt pozwolił na wdrożenie podstawowych zasad zarządzania użytkownikami FTP, takich jak tworzenie folderów z odpowiednimi uprawnieniami dostępu. Dzięki temu serwer FTP stał się bardziej bezpieczny i funkcjonalny.
5. **Testowanie i weryfikacja** – Przeprowadzone testy z użyciem klienta FTP (FileZilla) pozwoliły na weryfikację poprawności działania serwera FTP. Proces przesyłania plików przebiegał bezproblemowo, co potwierdziło, że serwer jest w pełni funkcjonalny.

Podsumowanie projektu

Projekt stanowił kompleksową demonstrację wykorzystania mikroserwisów oraz konteneryzacji w kontekście realizacji usługi FTP. Zastosowanie Dockera oraz Docker Compose umożliwiło szybkie uruchomienie i zarządzanie środowiskiem serwera FTP, zapewniając jednocześnie pełną izolację i przenośność aplikacji. Przeprowadzenie projektu pozwoliło na lepsze zrozumienie zasad działania mikroserwisów, konteneryzacji oraz protokołu FTP. Dzięki testom przeprowadzonym za pomocą FileZilli, udało się zweryfikować poprawność działania serwera i zapewnić jego niezawodność.

Wnioski z projektu wskazują na wysoką efektywność i elastyczność podejścia opartego na mikroserwisach i konteneryzacji, które mogą być z powodzeniem stosowane w bardziej złożonych systemach chmurowych. Projekt stanowi dobry punkt wyjścia do dalszych badań nad skalowalnością i bezpieczeństwem usług FTP w

kontenerach. Ostatecznie postanowiłem użyć alpine-ftp-server zamiast vsftpd.

6 Literatura:

Alpine - https://hub.docker.com/r/delfer/alpine-ftp-server
Uprawnienia folderów - https://pl.wikipedia.org/wiki/Chmod
Ogólne informacje dotyczące ftp i docker:
https://andradefarrael.medium.com/how-to-configure-a-ftp-server-on-your-ubuntu-local-environment-using-docker-e36e52a33698
https://stackoverflow.com/questions/47231797/how-to-install-and-configure-ftp-in-docker
https://tinztwinshub.com/software-engineering/set-up-an-ftp-server-with-docker/