Zarządzanie Systemami Rozproszonymi
Laboratoria z Docker - Kontenery aplikacyjne

mgr inż. Jakub Woźniak

Wprowadzenie

Cel laboratoriów: Celem tych laboratoriów jest zapoznanie studentów z podstawami pracy z kontenerami aplikacyjnymi przy użyciu Dockera. Studenci nauczą się uruchamiać kontenery w trybie interaktywnym, budować własne obrazy Docker, zarządzać wolumenami oraz tworzyć sieci między kontenerami. Laboratoria zakończą się zadaniem polegającym na uruchomieniu Redmine i Postgres w oddzielnych kontenerach, z zapewnieniem trwałości danych.
Wymagana wiedza:

- Podstawy pracy w terminalu (Linux).
- Znajomość programowania (np. w Pythonie).
- Podstawy pracy z bazami danych (Postgres).

Narzędzia:

- Docker zainstalowany na komputerze.
- Przykładowe pliki: `requirements.txt`, prosty kod aplikacji (np. Flask).

1 Uruchamianie kontenerów w trybie interaktywnym

Teoria: Kontenery to lekkie środowiska wirtualne, które pozwalają na izolowanie aplikacji wraz z ich zależnościami. Docker pozwala na uruchamianie takich kontenerów w trybie interaktywnym, co umożliwia bezpośrednią interakcję z systemem w kontenerze, np. za pomocą terminala.
Uruchomienie kontenera z bash:

```
docker run -it ubuntu bash
```

- `-it`: Tryb interaktywny z terminalem, umożliwia pracę z kontenerem jak z lokalnym terminalem.

- `ubuntu`: Wskazuje na obraz bazowy Ubuntu.
- `bash`: Polecenie uruchomienia interpretera bash wewnątrz kontenera.

Uruchomienie kontenera z konkretną wersją Pythona: Docker to również przydatne narzędzie gdy potrzebna jest konkretna wersja oprogramowania, która nie jest dostępna na systemie operacyjnym, a jego instalacja mogłaby spowodować konflikt zależności. Oprogramowanie uruchomione w kontenerze Docker jest w izolowanym środowisku i nie ma wpływu na inne procesy w systemie operacyjnym.

```
docker run -it python:3.8 bash
```

- `-it`: Tryb interaktywny z terminalem.
- `python:3.8`: Wskazuje na obraz Python w wersji 3.8.
- `bash`: Uruchamia interpreter bash w kontenerze.

Studenci mogą sprawdzić wersję Pythona za pomocą komendy `python --version`.

2 Budowanie własnego obrazu kontenera

Teoria: Docker umożliwia tworzenie własnych obrazów kontenerów, które zawierają wszystkie zależności potrzebne do uruchomienia aplikacji. Obraz kontenera jest budowany na podstawie pliku `Dockerfile`, który definiuje kroki budowy obrazu. Kolejność tych kroków ma kluczowe znaczenie w optymalizacji procesu budowy, ponieważ Docker używa mechanizmu cache do ponownego wykorzystania wcześniej wykonanych kroków.
Studenci powinni napisać prostą aplikację webową w Pythonie, wykorzystując framework Flask. Następnie zostanie zbudowany obraz Docker, który będzie uruchamiał tę aplikację.
Przygotowanie Dockerfile:
Studenci muszą stworzyć plik `Dockerfile` z poniższą zawartością:

```
FROM python:3.8

WORKDIR /app

COPY requirements.txt .

RUN pip install -r requirements.txt

COPY . .

ENV FLASK_APP=app.py

CMD ["flask", "run", "--host=0.0.0.0"]
```

Wyjaśnienie kolejności dyrektyw w Dockerfile: Kolejność poleceń w Dockerfile ma kluczowe znaczenie ze względu na sposób, w jaki Docker buduje obraz. Każde polecenie w Dockerfile tworzy nową warstwę obrazu. Jeśli wprowadzimy zmiany w jednej z warstw, Docker musi przebudować wszystkie kolejne warstwy. Dlatego warto najpierw skopiować plik `requirements.txt` i zainstalować zależności, a dopiero później skopiować kod aplikacyjny. Dzięki temu Docker będzie mógł skorzystać z mechanizmu cache, jeśli zależności nie zmienią się, co znacznie przyspieszy budowę obrazu.
Budowanie obrazu:

```
docker build -t my-flask-app .
```

- `-t my-flask-app`: Nazwa i tag dla obrazu, który zostanie zbudowany.
- `.`: Wskazuje na bieżący katalog jako lokalizację pliku `Dockerfile`.

Uruchomienie kontenera:

```
docker run -p 5000:5000 my-flask-app
```

- `-p 5000:5000`: Mapowanie portu 5000 na porcie hosta do portu 5000 w kontenerze, na którym działa aplikacja.
- `my-flask-app`: Nazwa obrazu, który ma być uruchomiony.

Wymiana kontenerów między komputerami:
Eksportowanie kontenera do pliku `.tar.gz`:

```
docker save my-flask-app | gzip > my-flask-app.tar.gz
```

- `docker save`: Eksportuje obraz kontenera do pliku tar.
- `gzip`: Kompresuje plik `.tar` do formatu `.tar.gz`.

Importowanie kontenera na innym komputerze:

```
gunzip -c my-flask-app.tar.gz | docker load
```

- `gunzip -c`: Dekompresuje plik `.tar.gz` i przekazuje jego zawartość do polecenia `docker load`.
- `docker load`: Wczytuje obraz kontenera z pliku tar.

3 Docker Volumes - Trwałość danych z Postgres

Teoria: Kontenery są z natury efemeryczne, co oznacza, że po usunięciu kontenera dane mogą zostać utracone. Aby tego uniknąć, Docker pozwala na używanie wolumenów, które przechowują dane poza kontenerem. Dzięki wolumenom możemy zapewnić trwałość danych nawet po restarcie lub usunięciu kontenera.

Uruchomienie bazy danych Postgres z wolumenem:

```
docker run -d --name postgres-db \
  -e POSTGRES_PASSWORD=mysecretpassword \
  -v postgres-data:/var/lib/postgresql/data \
  postgres
```

- `-d`: Uruchamia kontener w trybie "daemon" (w tle).
- `--name postgres-db`: Nazwa dla kontenera.
- `-e POSTGRES_PASSWORD=mysecretpassword`: Ustawia zmienną środowiskową dla hasła bazy danych.
- `-v postgres-data:/var/lib/postgresql/data`: Tworzy wolumen o nazwie `postgres-data`, który przechowuje dane Postgres.
- `postgres`: Obraz bazy danych Postgres.

4 Sieć między kontenerami

Teoria: Kontenery działające w Dockerze mogą komunikować się ze sobą, jeśli znajdują się w tej samej sieci. Docker pozwala na tworzenie sieci wirtualnych, które łączą kontenery i umożliwiają ich komunikację. Jest to szczególnie przydatne, gdy różne aplikacje (np. baza danych i aplikacja) muszą wymieniać dane.

Tworzenie sieci Dockera:

```
docker network create mynetwork
```

- `docker network create mynetwork`: Tworzy nową sieć o nazwie `mynetwork`.

Uruchomienie Postgres w tle:

```
docker run -d --name postgres-db \
  --network mynetwork \
  -e POSTGRES_PASSWORD=mysecretpassword \
  postgres
```

- `--network mynetwork`: Dołącza kontener do sieci o nazwie `mynetwork`.

Uruchomienie klienta PostgreSQL w trybie interaktywnym:

```
docker run -it --network mynetwork \
  --rm postgres psql -h postgres-db -U postgres
```

- `--rm`: Usuwa kontener po zakończeniu pracy.
- `psql`: Uruchamia klienta PostgreSQL wewnątrz kontenera.
- `-h postgres-db`: Wskazuje hosta bazy danych Postgres.
- `-U postgres`: Używa użytkownika o nazwie `postgres`.

5 Zadanie główne: Redmine z bazą danych Postgres

Opis zadania: Redmine to popularne narzędzie do zarządzania projektami, które wykorzystuje bazę danych Postgres do przechowywania danych. Celem zadania jest skonfigurowanie dwóch kontenerów: jednego z Redmine oraz drugiego z bazą danych Postgres. Kontenery te muszą działać w jednej sieci i być skonfigurowane w taki sposób, aby dane były przechowywane w sposób trwały, co oznacza, że reset żadnego z kontenerów nie może spowodować utraty danych.
Wymagania:

- Uruchom kontener z aplikacją Redmine.
- Uruchom oddzielny kontener z bazą danych Postgres.
- Zapewnij trwałość danych, korzystając z odpowiednich mechanizmów (wolumeny Docker).
- Oba kontenery muszą działać w tej samej sieci i Redmine musi korzystać z bazy Postgres.

Dokumentacja:

- Obraz Redmine: `https://hub.docker.com/_/redmine`
- Obraz Postgres: `https://hub.docker.com/_/postgres`
- Dokumentacja Redmine: `https://www.redmine.org/projects/redmine/wiki/Guide`

Wskazówki: Obraz postgres bierze pod uwagę zmienne środowiskowe wyłącznie do inicjalizacji bazy danych przy pierwszym uruchomieniu. Następnie większość tych zmiennych jest ignorowanych.
Do listowania wszystkich kontenerów służy polecenie `docker ps`. Aby zobaczyć logi kontenera, można użyć polecenia `docker logs <container-name>`. Uwaga! Jeżeli kontener nie jest widoczny na liście to znaczy, że prawdopodobnie zatrzymał się zaraz po uruchomieniu. Jego logi będą dostępne, a sam kontener będzie wciąż widoczny przy pomocy polecenia `docker ps -a`.
Kryteria zaliczenia:

- Redmine działa prawidłowo i jest dostępny na porcie 3000.
- Restart kontenera Redmine lub Postgres nie powoduje utraty danych.
- Po restarcie kontenerów, aplikacja Redmine nadal łączy się z bazą danych Postgres.
- Studenci muszą samodzielnie wyjaśnić, dlaczego dane są trwałe (wolumeny) oraz jak ich konfiguracja zapewnia tę trwałość.

Weryfikacja polega na wykonaniu następujących kroków:

- Zalogowanie na http://localhost:3000 przy pomocy użytkownika `admin` i hasła `admin`.
- Zmiana hasła użytkownika `admin` na inne.
- Dodanie nowego projektu w Redmine.
- Zatrzymanie kontenerów redmine i postgres przy pomocy polecenia `docker stop`.
- Usunięcie kontenerów redmine i postgres przy pomocy polecenia `docker rm`.
- Ponowne stworzenie kontenerów redmine i postgres.
- Sprawdzenie, czy dane są nadal dostępne po restarcie kontenerów.