Technologie i oprogramowanie chmurowe
Projekt
Bazan Michał – 163881

Temat:

Konteneryzacja – Docker. Różnica między Docker Compose i Docker Swarm (konfigurowania wielu kontenerów na tym samym hoście, narzędzie do orkiestracji kontenerów). Docker + WSL2 - windows subsystem for linux.

Charakterystyka+porownanie+przyklad implementacji.

Spis treści

Konteneryzacja

Czym jest konteneryzacja?

Konteneryzacja [1] to metoda wytwarzania, wdrażania i uruchamiania aplikacji w izolowanych środowiskach zwanymi kontenerami. Kontenery są swoistą formą wirtualizacji, która umożliwia pakowanie aplikacji wraz z jej zależnościami i środowiskiem uruchomieniowym, co pozwala na jednolite i niezawodne działanie aplikacji na różnych platformach.

Izolacja, która charakteryzuje kontenery, sprawia, że aplikacja w kontenerze ma ograniczony dostęp do zasobów systemowych hosta, takich jak pamięć RAM, procesor czy dysk twardy. Dzięki temu kontenery mogą współdzielić ten sam system operacyjny hosta, ale działają w izolowanych środowiskach, co zapewnia lepszą wydajność i bezpieczeństwo w porównaniu z tradycyjnymi maszynami wirtualnymi.

Kontenery są zwykle oparte na obrazach kontenerowych, które zawierają wszystko, co potrzebne do uruchomienia aplikacji, takie jak kod aplikacji, środowisko uruchomieniowe, biblioteki i inne zależności. Dzięki temu kontenery są przenośne i można je łatwo wdrażać w różnych środowiskach, co pozwala deweloperom na konsekwentne i niezawodne dostarczanie aplikacji.

Konteneryzacja staje się coraz bardziej popularna w dziedzinie rozwoju oprogramowania ze względu na swoją elastyczność, niezawodność i możliwość automatyzacji procesów wdrażania i skalowania aplikacji.

Konteneryzacja i Docker

Docker [2] [3] to potężne narzędzie, które rewolucjonizuje sposób, w jaki deweloperzy, inżynierowie DevOps i administratorzy zarządzają aplikacjami i infrastrukturą IT. Dzięki Dockerowi, aplikacje są pakowane w kontenery, które zawierają wszystkie potrzebne zależności, biblioteki i środowisko uruchomieniowe, co sprawia, że są przenośne i działać tak samo w każdym środowisku, niezależnie od tego, czy jest to środowisko deweloperskie, testowe czy produkcyjne.

Jego elastyczność pozwala na szybkie tworzenie, uruchamianie, skalowanie i zarządzanie aplikacjami w kontenerach. Deweloperzy mogą łatwo tworzyć lokalne środowiska deweloperskie, wdrażać aplikacje na różnych platformach chmurowych i zarządzać skalowaniem aplikacji w zależności od zmieniających się potrzeb biznesowych.

Jednym z kluczowych atutów Docker jest także możliwość integracji z innymi narzędziami i technologiami, takimi jak Continuous Integration/Continuous Deployment (CI/CD) pipeline'y, narzędzia monitorowania i logowania, czy też narzędzia do zarządzania infrastrukturą jako kodem.

Dodatkowo, Docker cieszy się dużą popularnością dzięki swojej otwartej społeczności, która stale rozwija ekosystem narzędzi, obrazów i rozwiązań, co ułatwia pracę z Dockerem i zapewnia wsparcie dla szerokiego zakresu zastosowań, od prostych aplikacji internetowych po złożone systemy mikroserwisów.

Docker Compose i Docker Swarm

Docker Compose

Docker Compose [4] jest narzędziem służącym do definiowania i uruchamiania wielokontenerowych aplikacji. Pozwala ono deweloperom opisać konfigurację wielu kontenerów w plikach YAML, co ułatwia zarządzanie zależnościami między kontenerami oraz uruchamianie aplikacji w środowiskach deweloperskich i testowych. Docker Compose jest szczególnie użyteczny w prostych aplikacjach, które składają się z kilku kontenerów i nie wymagają zaawansowanego skalowania.

Docker Swarm

Docker Swarm [5] jest narzędziem do orkiestracji kontenerów, które umożliwia zarządzanie klastrami kontenerów w środowiskach produkcyjnych. Pozwala na tworzenie i zarządzanie wieloma kontenerami na wielu maszynach fizycznych lub wirtualnych, co zapewnia skalowalność i wysoką dostępność aplikacji. Docker Swarm oferuje funkcje takie jak równoważenie obciążenia, odtwarzanie po awarii, aktualizacje bez przestoju oraz monitorowanie stanu klastra. Jest to idealne narzędzie do budowania i zarządzania złożonymi systemami mikroserwisów w środowiskach produkcyjnych.

Porównanie narzędzi

Docker Compose i Docker Swarm to dwa różne narzędzia stworzone przez Docker, Inc., które pomagają w zarządzaniu kontenerami, ale posiadają różne funkcje i zastosowania.

Zastosowanie: Docker Compose jest skoncentrowany na uruchamianiu wielokontenerowych aplikacji w środowiskach deweloperskich i testowych, podczas gdy Docker Swarm jest przeznaczony do zarządzania klastrami kontenerów w środowiskach produkcyjnych.

Skalowanie: Docker Swarm oferuje zaawansowane funkcje skalowania, takie jak automatyczne równoważenie obciążenia i skalowanie horyzontalne, które są nieobecne w Docker Compose.

Złożoność: Docker Compose jest prostszy w użyciu i szybszy do nauki, podczas gdy Docker Swarm może być bardziej złożony ze względu na potrzebę zarządzania klastrami i konfiguracją wielu węzłów.

Elastyczność: Docker Compose jest bardziej elastyczny i bardziej odpowiedni dla prostych aplikacji, podczas gdy Docker Swarm jest bardziej skomplikowany, ale oferuje większą kontrolę nad zasobami i skalowaniem aplikacji

Implementacja Docker Compose

Implementacja Docker Compose polega na tworzeniu pliku konfiguracyjnego w formacie YAML, który definiuje wszystkie kontenery, sieci, woluminy i inne zasoby potrzebne do uruchomienia aplikacji wielokontenerowej. Ten plik zawiera listę wszystkich usług (kontenerów) i ich konfiguracji.

Niżej ukazana implementacja została oparta o przykładową aplikację wielokontenerową [7] udostępnioną przez Dockera.

Zdefiniowanie i konfiguracja usług (kontenerów)

```
services:
  redis:
    image: redislabs/redismod
    ports:
      - '6379:6379'
  web:
    build:
      context: .
      target: builder
    # flask requires SIGINT to stop gracefully
    # (default stop signal from Compose is SIGTERM)
    stop_signal: SIGINT
    ports:
      - '8000:8000'
    volumes:
      - .:/code
    depends_on:
      - redis
```

Listing 1: Plik docker-compose.yml

Na wyżej umieszczonym listingu ukazany został plik *docker-compose.yml*, który definiuje dwie usługi:

a) usługa ***redis***, która wykorzystywana jest jako baza danych aplikacji

- wykorzystuje obraz ***redislabs/redismod***

- mapuje port hosta ***6379*** na port kontenera ***6379***, co umożliwia komunikację z serwerem Redis [8].

b) usługa ***web***

- buduje kontener na podstawie pliku Dockerfile w bieżącym katalogu,

- mapuje port hosta ***8000*** na port kontenera ***8000***, aby aplikacja była dostępna na porcie ***8000*** hosta,

- tworzy wolumin co umożliwia dynamiczne aktualizacje kodu źródłowego aplikacji bez ponownego uruchamiania kontenera,

- określa zależność od usługi ***redis***, co oznacza, że kontener usługi ***web*** nie będzie uruchamiany, dopóki serwer bazodanowy nie będzie uruchomiony.

```
# syntax=docker/dockerfile:1.4
FROM --platform=$BUILDPLATFORM python:3.10-alpine AS builder

WORKDIR /code

COPY requirements.txt /code
RUN --mount=type=cache,target=/root/.cache/pip \
    pip3 install -r requirements.txt

COPY . /code

ENTRYPOINT ["python3"]
CMD ["app.py"]

FROM builder as dev-envs

RUN <<EOF
apk update
apk add git bash
EOF

RUN <<EOF
addgroup -S docker
adduser -S --shell /bin/bash --ingroup docker vscode
EOF
# install Docker tools (cli, buildx, compose)
COPY --from=gloursdocker/docker / /
```

Listing 2: Plik Dockerfile

Wyżej umieszczony listing przedstawia plik *Dockerfile*, który definiuje dwa etapy budowy obrazu kontenera:

a) sekcja **builder** wykorzystuje obraz Pythona 3.10 Alpine jako bazowy obraz (FROM python:3.10-alpine AS builder). Tworzy katalog roboczy /code oraz kopiuje plik requirements.txt do tego katalogu. Następnie instaluje zależności wymienione w pliku requirements.txt za pomocą polecenia pip3 install -r requirements.txt. Ostatecznie kopiowane są pozostałe pliki źródłowe aplikacji do katalogu /code. Na koniec określa punkt wejścia (entrypoint) aplikacji jako python3 app.py.

b) sekcja **dev-envs** oparta jest na wcześniej zdefiniowanym etapie builder (FROM builder as dev-envs). W tej sekcji dodawane są narzędzia deweloperskie i konfiguracja środowiska deweloperskiego. Są one dodawane do kontenera za pomocą poleceń apk update, apk add git bash, addgroup, adduser. Następnie instalowane są narzędzia Docker (cli, buildx, compose) poprzez kopiowanie ich z innego obrazu (COPY --from=gloursdocker/docker / /).

Uruchomienie aplikacji

Aby uruchomić aplikację wielokontenerową wystarczy wykorzystać polecenie *$ docker-compose up -d*, które uruchamia kontenery w trybie tła (demon/detached mode).

```
$ docker-compose up -d

Building web
[+] Building 0.6s (12/12) FINISHED
docker:default
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 526B
 => [internal] load .dockerignore
 => => transferring context: 2B
 => resolve image config for docker.io/docker/dockerfile:1.4
 => CACHED
docker-image://docker.io/docker/dockerfile:1.4@sha256:9ba7531bd80fb0a858632727cf7a112fb
fd19b17e94c4e84ced81e24ef1a0dbc
 => [internal] load metadata for docker.io/library/python:3.10-alpine
 => [builder 1/5] FROM docker.io/library/python:3.10-alpine
 => [internal] load build context
 => => transferring context: 228B
 => CACHED [builder 2/5] WORKDIR /code
 => CACHED [builder 3/5] COPY requirements.txt /code
 => CACHED [builder 4/5] RUN --mount=type=cache,target=/root/.cache/pip     pip3
install -r requirements.txt
 => CACHED [builder 5/5] COPY . /code
 => exporting to image
 => => exporting layers
 => => writing image
sha256:3ed010016a47e8538f2c11b89973f7881e4ad54041f96268ac4e89927f67d883
 => => naming to docker.io/library/flask-redis_web
WARNING: Image for service web was built because it did not already exist. To rebuild
this image you must use `docker-compose build` or `docker-compose up --build`.
Creating flask-redis_redis_1 ... done
Creating flask-redis_web_1   ... done
```

Listing 3: Uruchomienie aplikacji

```
$ docker ps
CONTAINER ID   IMAGE                COMMAND                  CREATED         STATUS
PORTS                                       NAMES

5c2a6c632251   flask-redis_web      "python3 app.py"         4 minutes ago   Up 4
minutes   0.0.0.0:8000->8000/tcp, :::8000->8000/tcp   flask-redis_web_1

8360da071cf2   redislabs/redismod   "redis-server --load…"   4 minutes ago   Up 4
minutes   0.0.0.0:6379->6379/tcp, :::6379->6379/tcp   flask-redis_redis_1
```

Listing 4: Sprawdzenie poprawności uruchomienia aplikacji

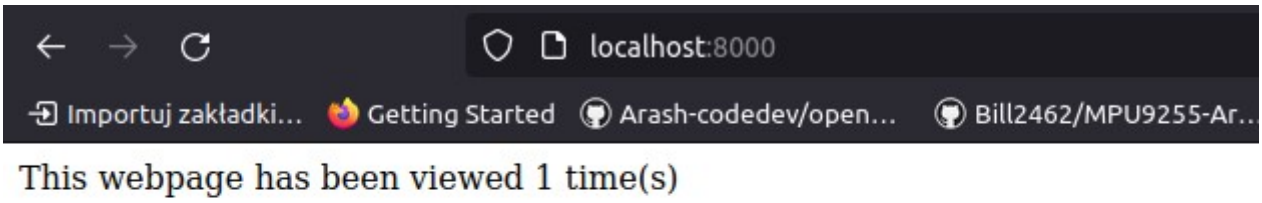

Figura 1: Aplikacja

Wyżej przedstawione listingi ukazują pomyślne zbudowanie aplikacji za pomocą narzędzia *docker-compose*.

Zatrzymanie i usunięcie aplikacji

Aby zatrzymać i usunąć aplikację należy wykorzystać następujące polecenie:

- *$ docker-compose down* – zatrzymuje i usuwa aplikację,

- *$ docker-compose down -v* – zatrzymuje i usuwa aplikację oraz zwalnia wykorzystywany wolumin.

```
$ docker-compose down

Stopping flask-redis_web_1   ... done
Stopping flask-redis_redis_1 ... done
Removing flask-redis_web_1   ... done
Removing flask-redis_redis_1 ... done
Removing network flask-redis_default
```

Listing 5: Zatrzymanie aplikacji

Implementacja Docker Swarm

Wnioski

Integracja z WSL2

Czym jest WSL2?

Test działania Dockera wewnątrz WSL2

Zalety i wady rozwiązania

Źródła

[1]. https://pl.wikipedia.org/wiki/Konteneryzacja

[2]. https://pl.wikipedia.org/wiki/Docker_(oprogramowanie)

[3]. https://docs.docker.com/

[4]. https://docs.docker.com/compose/

[5]. https://docs.docker.com/engine/swarm/

[6]. https://www.vim.org/

[7]. https://github.com/docker/awesome-compose/tree/master/flask-redis

[8]. https://redis.io/