Docker 101

Maciej Gowin @ CoderBrother

Linki

Opis

https://maciejgowin.github.io/coderbrother/#docker101

Kod źródłowy przykładów oraz zadań

https://github.com/MaciejGowin/docker101

Wirtualizacja

Wirtualizacja to technologia, która pozwala na stworzenie "wirtualnej" wersji narzędzia, która z perspektywy użytkownika działa w sposób zgodny z jego natywną wersją.

Istnieje szereg technik wirtualizacji. Jedną z najpopularniejszych jest wirtualizacja sprzętowa. Umożliwienie jednoczesnego uruchamianie wielu systemów operacyjnych na tej samej fizycznej maszynie.

Wirtualizacja sprzętowa

Maszyna hosta (host) - maszyna używana przez mechanizm wirtualizacji
Maszyna gościa (guest) - faktyczna maszyna wirtualna

Tworzenie wirtualnych maszyn gościa w obrębie maszyn hosta odbywa się poprzez narzędzia (software'owe lub hardware'owe) zwane hypervisor'ami (lub też monitorami maszyn wirtualnych).

Wirtualizacja sprzętowa: typy

Pełna wirtualizacja

Pełna lub prawie pełna symulacja faktycznego sprzętu umożliwiająca uruchamianie oprogramowania na maszynach wirtualnych bez jakiejkolwiek modyfikacji uruchamianego oprogramowania

Parawirtualizacja

Oprogramowanie uruchamiane na wirtualnej maszynie działa w środowisku izolowanym, sprawia wrażenie działania na osobnym systemie, podczas gdy środowisko sprzętowe nie jest symulowane.
Wymagana jest modyfikacja oprogramowania uruchamianego.

Wirtualizacja sprzętowa: hyperwizor

Hypervisor typu 1 (natywny)

Zarządca natywny, który jest uruchamiany bezpośrednio na zasobach sprzętowych bez udziału
systemu operacyjnego hosta. Sam pełni rolę systemu operacyjnego mając bezpośredni dostęp do sprzętu.

Hypervisor typu 2 (osadzony)

Zarządca osadzony działający na systemie operacyjnym maszyny hosta.

Wirtualizacja a emulacja

W emulacji jedno rozwiązanie sprzętowe imituje drugie, podczas gdy w wirtualizacji "hypervisor" imituje część lub całość sprzętu.

Wirtualizacja sprzętowa: przykładowe rozwiązania

Najpopularniejsze rozwiązania wirtualizacji sprzętowej:

- VMware
- Microsoft Hyper-V
- Xen
- KVM

Konteneryzacja

Wirtualizacja na poziomie systemu operacyjengo nazywana jest konteneryzacją
Jest to rozwiazanie działające na poziomie ssytemu operacyjnego wykorzystujące jego jądro, które pozwala na istnienie wielu wyizolowancyh przestrzeni użytkownika.

Każda z wyizolowanych instncji nazywana jst kontenerem.

Konteneryzacja

Kontener imituje działanie osobnego bytu z punktu widzenia programu, który jest w nim osadzony.

Programy uruchamiane w obrębie kontenerów widzą jedynie zasoby, które zostały przypisane do danego kontenera.

Główną zaletą konteneryzacji nad wirtualizacją jest użycie mniejszej ilości zasobów przez współdzielenie jądra.

Narzędziem do konteneryzacji jest Docker.

Docker

Głównym narzędziem pozwalającym na prace z kontenerami jest Docker Engine, którego częścią jest klient CLI `docker`.

```
https://docs.docker.com/engine/
```

Narzędziem wspierającym jest rozszerzenie silnika o narzędzia programistyczne dostarczanego jako Docker Desktop.

```
https://docs.docker.com/desktop/
```

Docker: instalacja

Instalacja Docker Desktop poprzez instalator dla konkretnej platformy.

```
https://docs.docker.com/get-docker/
```

Docker: działanie

Aby uruchomić kontener (container), musimy wybrać obraz (image), na podstawie którego zostanie on stworzony.

Obraz zawiera wszystkie biblioteki, zależności oraz pliki konieczne do uruchomienia kontenera.

Docker: Docker Hub

Jeżeli obraz nie istnieje na lokalnej maszynie, Docker postara się go pobrać z `Docker Registry`. W naszym przypadku z publicznego rejestru `Docker Hub`.

Repozytorium `Docker Hub` posiada wiele obrazów gotowych do użycia. Dodatkowo możemy tworzyć nowe obrazy na podstawie już istniejących.

Komenda: *docker container*

Zarządzanie kontenerami.

```
docker container ...
```

Szczegóły:

- https://docs.docker.com/reference/cli/docker/container/

Komenda: *docker container run*

Stworzenie oraz uruchomienie kontenera na podstawie obrazu.

```
docker container run ...
docker run ...
```

Szczegóły:

- https://docs.docker.com/reference/cli/docker/container/run/

Komenda: *docker container run*

- Stworzenie oraz uruchomienie kontenera o nazwie `--name test` na podstawie obrazu `ubuntu` w trybie interaktywnym `-i` oraz alokacją pseudo-TTY `-t`.

  ```
  docker run -it --name test ubuntu
  ```

  W przypadku braku nazwy zostanie wygenerowana nazwa losowa.

Komenda: *docker container start/stop*

Uruchomienie/zatrzymanie istniejącego kontenera.

```
docker container start ...
docker start ...

docker container stop ...
docker stop ...
```

Szczegóły:

- https://docs.docker.com/reference/cli/docker/container/start/
- https://docs.docker.com/reference/cli/docker/container/stop/

Komenda: *docker container start/stop*

- Uruchomienie kontenera o nazwie `test`

```
docker start test
```

- Zatrzymanie kontenera o nazwie `test`

```
docker stop test
```

Komenda: *docker container exec*

Wykonanie komendy na uruchomionym kontenerze.

```
docker container exec ...
docker exec ...
```

Szczegóły:

- https://docs.docker.com/reference/cli/docker/container/exec/

Komenda: *docker container exec*

- Uruchomienie w trybie interaktywnym powłoki na kontenerze `test`

```
docker exec -it test bash
```

- Wylistowanie plików na poziomie katalogu roboczego na kontenerze `test`

```
docker exec test ls
```

Komenda: *docker container ls*

Wylistowanie dostępnych kontenerów.

```
docker container ls ...
docker ps ...
```

Szczegóły:

- https://docs.docker.com/reference/cli/docker/container/ls/

Komenda: *docker container ls*

- Wylistowanie uruchomionych kontenerów

```
docker ps
```

- Wylistowanie wszystkich kontenerów

```
docker ps -a
```

- Wylistowanie wszystkich kontenerów, których nazwa zawiera `te`

```
docker ps -a --filter "name=te"
```

- Wylistowanie wszystkich kontenerów oraz wyświetlenie tylko identyfikatorów

```
docker ps -aq
```

Komenda: *docker image*

Zarządzanie obrazami.

```
docker image ...
```

Szczegóły:

- https://docs.docker.com/reference/cli/docker/image/

Komenda: *docker image*

- Wylistowanie dostępnych lokalnie obrazów

```
docker image ls
```

- Wylistowanie dostępnych lokalnie obrazów, które nie zostały otagowane (dangling) przy użyciu filtra

```
docker image ls --filter "dangling=true"
```

- Usunięcie lokalnego obrazu na podstawie identyfikatora

```
docker image rm 717f054228ad
```

Docker: tworzenie obrazów

Docker pozwala na tworzenie własnych obrazów, które powastają na bazie istniejących. Definicja obrazu kótry ma zosta stworzony opiera się na pliku `Dockerfile`.

Szczegóły:

- https://docs.docker.com/reference/dockerfile/

Docker: tworzenie obrazów

```
FROM openjdk:17-oracle
WORKDIR app

COPY target/docker-spring-boot-1.0-SNAPSHOT.jar app.jar

COPY entrypoint.sh entrypoint.sh

ENTRYPOINT ["./entrypoint.sh"]
```

Komenda: *docker buildx build*

Zarządzanie obrazami.

```
docker buildx build ...
docker build ...
```

Domyślnie budowanie będzie odbywa się na podstawie pliku `Dockerfile` znajdującego się w katalogu, w którym komenda została uruchomiona.

Szczegóły:

- https://docs.docker.com/reference/cli/docker/image/

Zadanie: docker-spring-boot

- Stworzenie prostego RESTful service opartego o Spring Boot.
- Stworzenie obrazu z wbudowaną aplikacją.
- Uruchomienie nowopowstałego kontenera.

Zadanie: docker-spring-boot-jpa

- Stworzenie prostego RESTful service opartego o Spring Boot wymagającego połączenia do bazy danych.
- Uruchomienie serwisu oraz odwołanie się do uruchomionego kontenera z bazą danych.

Zadanie: docker-spring-boot-jpa-compose

- Stworzenie prostego RESTful service opartego o Spring Boot wymagającego połączenia do bazy danych.
- Stworzenie obrazu ze wbudowaną aplikacją.
- Uruchomienie kompozytu kontenerów.