INTERNET PAMETINIH UREĐAJA

prof. dr Dejan S. Aleksić

Prirodno-matematički fakultet, Niš

06. DOCKER: REVOLUCIJA U ISPORUCI SOFTVERA

ŠTA JE DOCKER? (KONCEPT)

Glavna ideja (Analuzija sa transportnim kontejnerom**):** Pre standardnih metalnih kontejnera, utovar robe na brodove je bio haos (različite veličine buradi, džakova, kutija). Docker je uradio istu stvar za softver.

Ključne teze:

- **Standardizovana jedinica:** Docker pakuje aplikaciju i *sve* njene zavisnosti (biblioteke, konfiguracije, runtime) u jedan paket koji se zove **Kontejner**.
- **"Build once, run anywhere":** Ako radi na mom laptopu, garantovano radi i na serveru, u Cloudu, ili kod kolege na mašini.
- **Izolacija:** Svaki kontejner misli da je jedini na sistemu, ima svoj fajl sistem i procese, iako deli resurse mašine.

KRAJ SINDROMA "RADI NA MOJOJ MAŠINI"

Vizuelni prikaz:

- **Development:** Programer kaže: "Kod mene radi!" (Ima Python 3.8, specifične biblioteke).
- **Production:** Server pada. (Ima Python 3.6, nedostaju biblioteke, drugačiji OS patch).

Rešenje:

- Docker eliminiše **"Dependency Hell"**.
- Ne instalirate softver direktno na operativni sistem servera.
- Server treba da ima instaliran *samo* Docker, a sve ostalo (baza, aplikacija, cache) dolazi zapakovano u kontejnerima.

DOCKER VS. VIRTUELNE MAŠINE

- Ovo je pitanje koje svi postavljaju. Razlika je u arhitekturi.

Osobina	Virtuelna Mašina (VM)	Docker Kontejner
Arhitektura	Hardverska virtualizacija (Hypervisor)	OS virtualizacija (Container Engine)
Operativni sistem	Svaka VM ima svoj **pun Guest OS** (memorija).	Svi dele **isti Linux Kernel** domaćina (memorija).
Veličina	Gigabajti (GB)	Megabajti (MB)
Vreme pokretanja	Minuti (Boot time)	Milisekunde (Start time)
Performanse	Manje efikasno (overhead OS-a)	Skoro kao nativna aplikacija

OSNOVNI POJMOVI -DOCKERFILE, IMAGE, CONTAINER

1. Dockerfile:

   Tekstualni fajl sa instrukcijama. Primer: "Uzmi Linux, instaliraj Java 17, kopiraj moj kod, otvori port 8080.“

2. Image:

   Kada se Dockerfile "izgradi" (build), dobija se Image. To je "read-only" šablon. Ne može se menjati.

3. Container:

   Kada pokrenemo Image, dobijamo Kontejner. To je živa, "run-time" instanca. Možemo pokrenuti 100 kontejnera iz jednog Image-a.

Vizuelna analogija:

Dockerfile = Recept za pitu.

Image = Zamrznuta pita u prodavnici.

Container = Ispečena pita vruća na stolu (koju jedemo/koristimo).

DOCKER U PRAKSI (HELLO WORLD)

Cilj: Pokrenuti Nginx web server bez ikakve instalacije i konfiguracije, u jednoj sekundi.

```
docker run -d -p 8080:80 nginx
```

Šta ova "magična" linija radi?

- docker run: Naređuje Dockeru da kreira i pokrene kontejner.
- -d (Detached): Pokreni u pozadini (da ne blokira terminal).
- -p 8080:80 (Port Mapping): Ključni deo!
  - Poveži port 8080 na mom laptopu (Host) sa portom 80 unutar kontejnera.
- nginx: Ime Image-a. Ako ga nemate, Docker ga automatski skida (pull) sa interneta (Docker Hub).

Rezultat:

- Otvorite web pretraživač, ukucate localhost:8080 i vidite "`Welcome to Nginx!`".
- Kada vam više ne treba: docker stop [id]. Server nestaje kao da nikad nije postojao.

DOCKER COMPOSE: ORKESTRACIJA KONTEJNERA

Problem:

Retko kad nam treba samo jedan kontejner. Obično nam treba Aplikacija + Baza podataka + Cache (npr. Python + PostgreSQL + Redis). Pokretati ih ručno ("docker run..." tri puta) je naporno i podložno greškama.

Rešenje:

docker-compose.yml

Jedan fajl koji opisuje celu arhitekturu vašeg sistema.

DOCKER COMPOSE: ORKESTRACIJA KONTEJNERA

docker-compose.yml - Primer strukture fajla:

```
version: '3'
services:
  mosquitto:
    image: eclipse-mosquitto:latest
    ports: ["1883:1883"]
    restart: unless-stopped

 baza-podataka:
    image: postgres:13
    environment:
      POSTGRES_PASSWORD: tajna

 web-aplikacija:
    image: moja-aplikacija:latest
    ports: ["5000:5000"]
```

DOCKER COMPOSE: ORKESTRACIJA KONTEJNERA

```
docker-compose up
```

Docker čita fajl, skida sve Image-e, kreira mrežu između njih, i pokreće celu aplikaciju odjednom.

Sledeći logičan korak bi bilo predavanje Kubernetes, jer on dolazi nakon što Docker Compose više nije dovoljan (kada prelazimo sa jednog servera na klaster od 100 servera).

OPTIMIZACIJA DOCKER IMAGE-A

- Optimizacija Docker Image-a je ključna tema za produkciju.
- Niko ne želi da čeka 10 minuta da se Image od 2GB prebaci na server, niti da plaća skladištenje tolikih podataka.
- Pored toga, manji Image-i su i bezbedniji (manje biblioteka = manje potencijalnih rupa).

VELIČINA JE BITNA: ALPINE VS. STANDARD

Problem:

- Mnogi početnici počinju sa FROM ubuntu ili FROM python.
- Ovi image-i sadrže stotine alata koji vašoj aplikaciji verovatno ne trebaju (curl, git, vim, man-pages...).
- To rezultira ogromnim Image-ima (500MB+).

Rešenje:

- Koristite Alpine Linux ili Slim verzije.
- Alpine: Minimalistička Linux distribucija (svega ~5MB).
- Slim: Verzije jezika očišćene od nepotrebnih fajlova.

VELIČINA JE BITNA: ALPINE VS. STANDARD

Poređenje (Primer za Python):

Base Image	**Veličina (cca)**	**Sadržaj**
python:3.9	~900 MB	Pun OS, kompajleri, biblioteke
python:3.9-slim	~120 MB	Samo neophodno za Python
python:3.9-alpine	**~45 MB**	Apsolutni minimum

PAMETNO KORIŠĆENJE KEŠA (LAYER CACHING)

Koncept:

Docker gradi Image u slojevima (Layers). Ako promenite jednu liniju koda, Docker mora da ponovo izgradi taj sloj i sve slojeve ispod njega.

Loša praksa (Sve se ponovo gradi pri svakoj promeni koda):

```
COPY . .                  # <-- Kopiramo kod (često se menja)
RUN pip install -r requirements.txt # <-- Instaliramo biblioteke (dugo traje)
```

Ako promenite jedno slovo u kodu, Docker ruši keš i ponovo skida sve biblioteke!

Dobra praksa (Optimizovan keš):

```
COPY requirements.txt .  # 1. Kopiramo samo spisak zavisnosti
RUN pip install -r requirements.txt # 2. Instaliramo ih (Kešira se!)
COPY . .                 # 3. Tek na kraju kopiramo kod
```

Sada, kad menjate kod, Docker koristi keširane biblioteke iz koraka 2 i build traje 1 sekundu umesto 5 minuta.

STAGE BUILDS (ZLATNI STANDARD)

Problem:

Za jezike koji se kompajliraju (Go, Java, C++, Rust), potreban vam je kompajler (npr. JDK, GCC) da napravite aplikaciju, ali vam on ne treba da biste je pokrenuli. Ako ostavite kompajler u Image-u, on nepotrebno zauzima prostor.

Rešenje: Koristite više faza u jednom Dockerfile-u.

```
# FAZA 1: Build (Težak Image)
FROM golang:1.19 AS builder
COPY . .
RUN go build -o moja-aplikacija

# FAZA 2: Produkcija (Lagan Image)
FROM alpine:latest
# Kopiramo SAMO gotov fajl iz Faze 1
COPY --from=builder /app/moja-aplikacija .
CMD ["./moja-aplikacija"]
```

Rezultat: Dobijate Image koji sadrži **samo** vaš binarni fajl. Veličina se često smanjuje sa **800MB** na **20MB**.

.dockerignore