Docker

Praktični uvod u kontejnerizaciju softvera

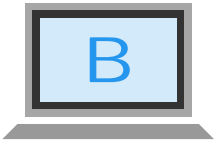
Marin Meštrović

Softversko inženjerstvo

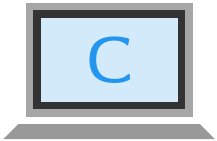
22. siječnja 2026.

Uvod: Tri programera (i tri OS-a)

MacOS (Dev)



Windows (Dev)



Linux (Produkcija)

Konflikt Verzija

Lokalno je instalirana novija verzija baze (v3) za drugi projekt, a ova aplikacija zahtijeva v2.

Rezultat:
Rušenje aplikacije.

Nedostajuće Biblioteke

Setup skripte su pisane u Bashu i koriste Linux alate koji ne postoje na Windowsu.

Rezultat:
Neuspješan build proces.

Konfiguracija Okoline

Hardkodirane putanje datoteka i prava pristupa s Dev računala ne vrijede na Serveru.

Rezultat:
Runtime greške u produkciji.

Izazovi: Kompatibilnosti i okruženja

Problem eksponencijalnog rasta konfiguracija

10 Mikroservisa $\times$ 3 Okruženja (Dev, Test, Prod) = 30 Scenarija

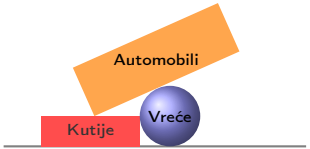
Svaka kombinacija servisa i OS-a zahtijeva specifičan skup biblioteka i konfiguracija.

- **Dependency conflict:** Dvije aplikacije na istom serveru zahtijevaju različite verzije iste biblioteke (npr. Python 3.9 vs Python 3.13), što dovodi do konflikta.
- **Narušavanje integriteta OS-a:** Instalacija stotina različitih alata direktno na Host OS s vremenom stvara zaostale datoteke koje uzrokuju nestabilnost sustava.
- **Dugotrajna priprema okoline:** Postavljanje razvojnog okruženja za novog člana tima je ručni proces koji traje danima, umjesto da bude automatiziran i brz.

Inspiracija: Revolucija u transportu

Prije 1960-ih, utovar brodova je bio spor i kompliciran jer je svaki teret bio drugačijeg oblika.

Prije standardizacije



- Teško za slaganje (različite dimenzije).
- Spor utovar i istovar.
- Ovisnost o vrsti prijevoza.

Intermodalni kontejner

- Standardne dimenzije.
- Proizvoljan sadržaj unutar kontejnera.
- Proizvoljan brod ili kamion.

Docker: Standardizirana jedinica softvera

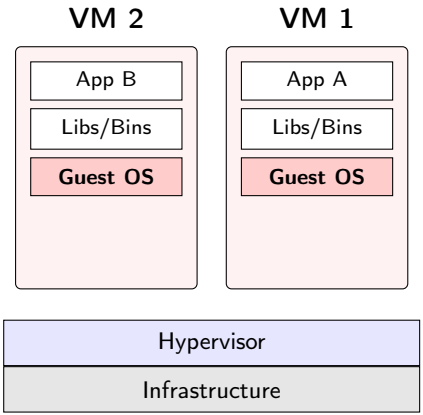
Docker primjenjuje istu filozofiju na softver. Umjesto fizičkog kontejnera, imamo digitalni.

Rješenje

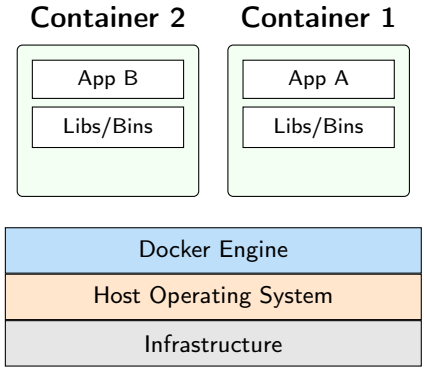
Docker omogućuje pakiranje aplikacije i **svih njenih ovisnosti** (biblioteke, konfiguracijske datoteke, alati sustava) u jedan standardizirani paket spreman za pokretanje.

1. **Što je "unutra"? (Sadržaj)**
   - *Aplikacijski kod* (Node.js, Java, Python).
   - *Runtime okruženje* (npr. JRE).
   - *Sistemske biblioteke* (.so, .dll).
   - *Postavke* (Environment variables).
2. **Što je "izvana"? (Infrastruktura)**
   - *Nezavisnost:* Kontejneru je svejedno vrti li se na Linux serveru, Windows laptopu ili u Cloud clusteru.
   - *Garancija:* Ako radi u Build fazi, radit će identično i u Produkciji.

Tehnička pozadina: VM vs. Docker



Virtual Machine



Docker Container

Zašto je Docker brži? (Usporedba)

Značajka	Virtual Machine	Docker Container
Kernel	Svaki VM ima svoj	Dijele Host Kernel
Veličina	Gigabajti (GB)	Megabajti (MB)
Start-up	Minute (boot OS-a)	Milisekunde (start procesa)
Performanse	Slabije (Hypervisor overhead)	Nativne (direktan pristup CPU)

Koncept: Image vs. Container

Analogija s **OOP**:

Docker Image ≈ Klasa

- **Nacrt** aplikacije.
- **Read-only**.
- Sadrži izvorni kod i biblioteke.
- *Primjer:* `node:14`

RUN →

Docker Container ≈ Objekt

- **Instanca** pokrenuta iz slike.
- **Read-Write**.
- Moguće više instanci iz jedne slike.
- *Primjer:* `backend-server`

Docker Registry (Docker Hub)

Gdje se nalaze slike?

- **Docker Hub:** "App Store" za Docker slike.
- **Official Images:** Službene slike koje održavaju proizvođači (Mongo, Redis, Node, Nginx).
- Sigurnost: Uvijek preferirati *Official Images* jer su skenirane na ranjivosti.

```
docker pull nginx:latest
```

Instalacija Dockera

1. Linux

```
sudo apt-get update && sudo apt-get install -y docker.io
sudo systemctl start docker
```

2. macOS

```
brew install --cask docker
# Alternativno: Preuzeti Docker Desktop .dmg
```

3. Windows

- Preuzeti instalaciju:
  `https://www.docker.com/products/docker-desktop`
- Pokrenuti instalaciju i **restartati** računalo.

Provjera instalacije

Terminal Output

```
$ docker --version
Docker version 28.2.2, build 28.2.2-0ubuntu1~24.04.1

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

Što ovo znači?

- Prva naredba potvrđuje da je binarna datoteka instalirana.
- Druga naredba (`docker ps`) potvrđuje da **Docker Daemon** radi u pozadini (prazna lista znači da nema aktivnih kontejnera, ali servis radi).

Osnovne naredbe (Pull & Run)

Zamislimo da želimo pokrenuti web server (**Nginx**).

```
# 1. Preuzmi sliku (ako ne postoji lokalno)
docker pull nginx:1.23

# 2. Pokreni container
docker run nginx:1.23
```

Detached Mode i Naming

Kako pokrenuti container u pozadini (kao servis)?

```
# -d = detached (u pozadini)
# --name = dajemo mu ime (lakse upravljanje)
docker run -d --name web nginx:1.23

# Provjera aktivnih kontejnera
docker ps

# Pregled logova (sto se dogadja unutra?)
docker logs web
```

Anatomija docker run naredbe

Struktura naredbe

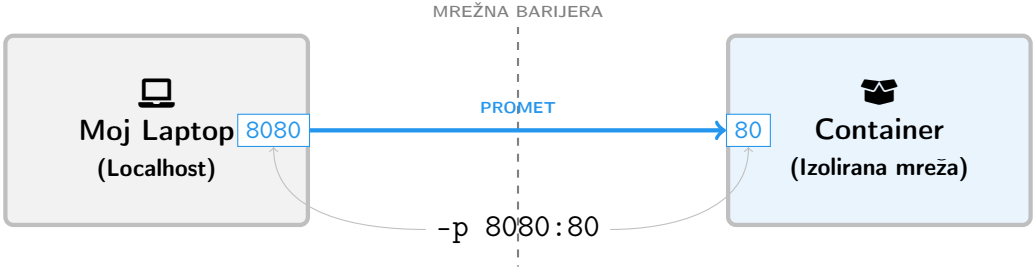
Općenita sintaksa uvijek prati ovaj redoslijed:

```
docker run [OPTIONS] IMAGE [COMMAND]
```

Port Binding (Umrežavanje)

Problem: Container je po defaultu izoliran.

- Nginx vrti aplikaciju na portu **80**.
- Mi želimo pristupiti preko porta **8080**.
- Rješenje: `-p <VANJSKI>:<UNUTARNJI>`



Pokretanje s mapiranjem porta

```
# Mapiraj port 8080 na mom laptopu na port 80 u kontejneru
docker run -d -p 8080:80 --name web-app nginx:1.23
```

Sada možemo otvoriti preglednik i otići na:
`http://localhost:8080` -> Prikazuje se Nginx "Welcome" stranica!

1. Što pakiramo?

Cilj: Zapakirati jednostavnu Python web aplikaciju.

app.py (naša aplikacija):

- Prikazuje broj posjeta stranice.
- Ispisuje *hostname* (ID containera).

```
from flask import Flask, render_template_string
import os

app = Flask(__name__)

counter = 0
...
```

Bez Dockera: Treba nam Python, pip, venv...
S Dockerom: Samo nam treba ***Dockerfile****.*

2. Dockerfile: Recept za sliku

Kako bi Docker znao što napraviti, pišemo upute korak po korak.

```
# 1. Baza: Sluzbeni Python image
FROM python:3.11-slim

# 2. Radni direktorij u containeru
WORKDIR /app

# 3. Instalacija ovisnosti
COPY requirements.txt .
RUN pip install -r requirements.txt

# 4. Kopiranje koda aplikacije
COPY app.py .

# 5. Komanda za pokretanje
CMD ["python", "app.py"]
```

Što se događa?

- `FROM`: Temelj (OS + Python).
- `WORKDIR`: Gdje ćemo raditi.
- `COPY`: Prebacivanje fajlova s laptopa u sliku.
- `RUN`: Instalacija biblioteka.

3. Izgradnja (Build) Slike

Pretvaramo naš `Dockerfile` i kod u izvršnu sliku.

```
docker build -t docker-demo:latest .
```

Rezultat:

- Docker prolazi kroz instrukcije.
- Kreira slojeve.
- Rezultat je **image** koji sadrži sve: Python, Flask i naš kod.

4. Pokretanje Containera

Imamo sliku, sada želimo pokrenuti instancu.

```
docker run -d -p 5000:5000 --name my-app docker-demo:latest
```

Ključne zastavice:

- `-d` (detached): Pokreni u pozadini (ne blokiraj terminal).
- `-p 5000:5000`: Mapiraj port laptopa na port containera.
- `-name`: Dajemo mu ime za lakše upravljanje.

5. Aplikacija u akciji

Provjera u terminalu:

```
$ docker ps
CONTAINER ID   IMAGE          PORTS
a1b2c3d4e5     docker-demo    0.0.0.0:5000->5000/tcp
```

Provjera u pregledniku:

- URL: `http://localhost:5000`

Slika: Aplikacija prikazuje Broj posjeta stranice

Naredbe: rad s Image

Priprema i izgradnja:

```
# Preuzimanje gotove slike s Docker Huba
docker pull nginx:1.23

# Izgradnja vlastite slike iz Dockerfile-a
# -t definira ime i tag (verziju)
docker build -t moj-app:1.0 .
```

Pregled i brisanje:

```
# Izlistaj sve slike dostupne lokalno
docker images

# Obri i sliku (oslobadanje prostora)
docker rmi <image_name_or_id>
```

Naredbe: rad s Container

Pokretanje (Run):

```
# -d (pozadina), -p (port host:container), --name (naziv)
docker run -d -p 5000:3000 --name web-app moja-app:1.0
```

Dijagnostika i nadzor:

```
docker ps              # Samo aktivni kontejneri
docker ps -a           # Svi (ukljucujuci zaustavljene)
docker logs <id>       # Ispis logova aplikacije
```

Životni ciklus:

```
docker stop <id>       # Zaustavi container
docker start <id>      # Ponovno pokreni postojeceg
docker rm <id>         # Trajno obrisi kontejner
```

Inspekcija, Skaliranje i Čišćenje

1. Dubinska analiza i testiranje:

```
# Prikaz svih detalja (IP, env vars, paths)
docker inspect my-app

# Testiranje odgovora iz terminala
curl http://localhost:5000
```

2. Skaliranje (Više instanci):

```
docker run -d -p 5001:5000 --name my-app-2 docker-demo
```

3. Potpuno čišćenje:

```
# Zaustavi i obrisi oba kontejnera odjednom
docker stop my-app my-app-2
docker rm my-app my-app-2

# Obrisi image (nakon brisanja containera)
docker rmi docker-demo:latest
```

Docker Compose: Automatizacija

Umjesto ručnog kucanja `docker run` naredbi, koristimo definiciju u datoteci (`docker-compose.yml`).

```
# Podigni cijelu aplikaciju (u pozadini)
docker-compose up -d

# Provjeri status servisa definiranih u Composeu
docker-compose ps

# Prati logove svih servisa
docker-compose logs -f

# Zaustavi i ukloni sve (mreze, kontejnere)
docker-compose down
```

Napomena: Docker Compose je idealan za lokalni razvoj (Dev environment).

Sažetak Ključnih Koncepata

1. **Reproducibilnost**
   - Okolina je identična u razvoju, testiranju i produkciji.

2. **Image vs. Container**
   - **Image:** Statični, *read-only* predložak. Sadrži kod, biblioteke i ovisnosti.
   - **Container:** Izvršna instanca slike (Runtime). To je izolirani proces s vlastitim datotečnim sustavom.

3. **Izolacija**
   - Containeri dijele kernel OS-a, ali su međusobno izolirani na razini procesa i mreže.

Hvala na pažnji!

Pitanja?