UNIVERZITET U NOVOM SADU
FAKULTET TEHNIČKIH NAUKA
KATEDRA ZA PRIMENJENE RAČUNARSKE NAUKE

Računarstvo u oblaku

ms Helena Anišić

Zimski semester 2022/2023.

Studijski program: Računarstvo i automatika

Modul: Računarstvo visokih performansi

Zadatak 1

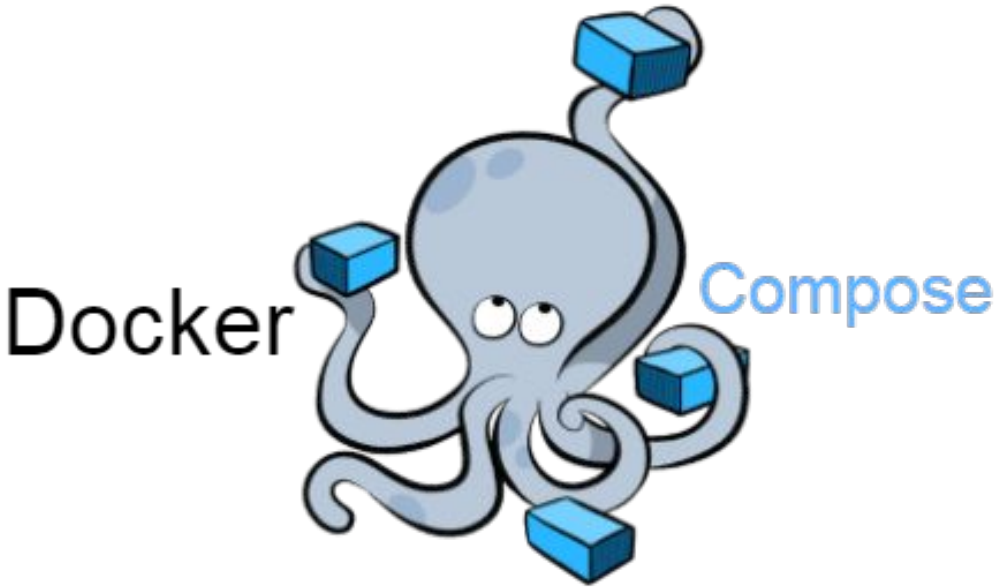
- Potrebno je kontejnerizovati jedan tipičan primer veb aplikacije.
- Delovi veb aplikacije su:
  - mongo baza podataka (naredba za pokretanje baze: ***docker run mongo)***
    - trajno perzistiranje podataka: **/data/db** putanja u mongo kontejneru gde se čuvaju podaci
  - node.js backend aplikacija
    - trajno perzistiranje logova: **/logs** putanja u aplikaciji
    - live reload koda (bez potrebe da se builduje slika docker kontejnera)
    - node_modules i Dockerfile ne treba da se kopiraju u kontejner
  - react frontend aplikacija
    - **react aplikacija ne može da se nalazi unutar docker mreže jer se izvršava unutar pretraživača**
    - **react aplikacija mora da se pokrene sa -it opcijom**
    - live reload koda (bez potrebe da se builduje slika docker kontejnera)
    - node_modules i Dockerfile ne treba da se kopiraju u kontejner

Docker komande za Zadatak 1

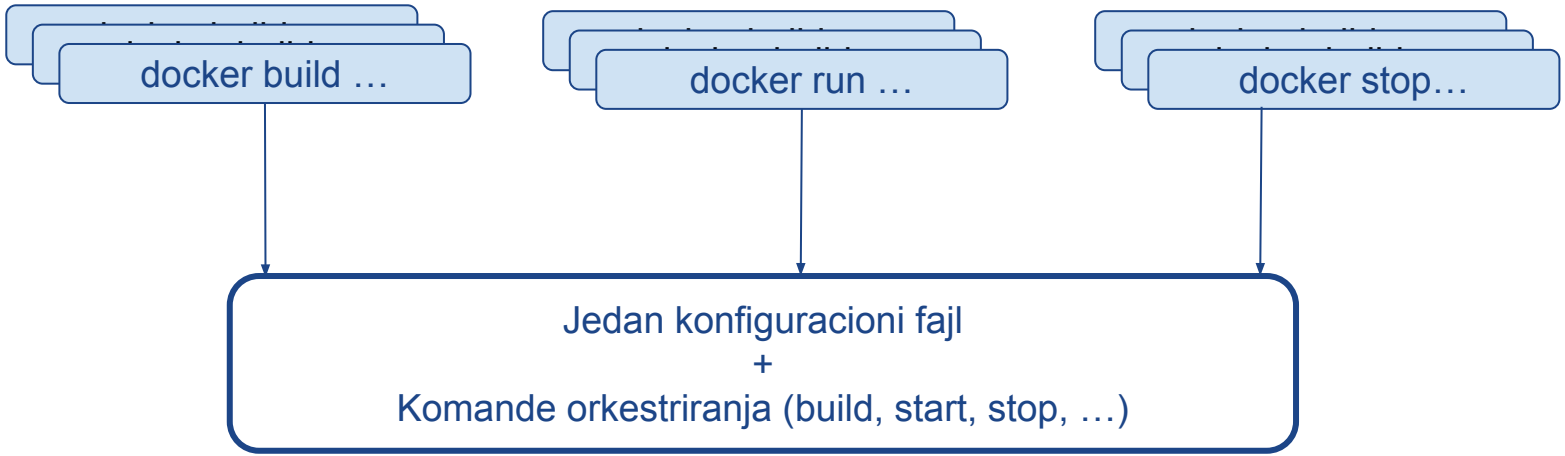
1. Kreirati mrežu
   - **docker network create goals-net**
2. Pokrenuti mongoDB kontejner
   - **docker run --name mongodb -v data:/data/db --rm -d --network goals-net mongo**
3. Bildovati nodejs bekend sliku kontejner
   - **docker build -t goals-node .**
4. Pokrenuti nodejs bekend kontejner
   - **docker run --name goals-backend --network goals-net -v logs:/app/logs -v /home/helena/Desktop/veb/backend:/app -v /app/node_modules --rm -d -p 8082:8081 goals-node**
5. Bildovati react frontend sliku kontejnera
   - **docker build -t goals-react .**
6. Pokrenuti react frontend kontejner
   - **docker run --name goals-frontend -v /home/helena/Desktop/veb/frontend/src:/app/src --rm -d -p 3000:3000 -it goals-react**
7. Stopiranje kontejnera
   - **docker stop mongodb goals-backend goals-frontend**

Docker komande za Zadatak 1

- Kako olakšati, ubrzati i učiniti preglednijim bildovanje, pokretanje, zaustavljanje i brisanje više-kontejnerskih aplikacija?


Docker
Compose

Šta je Docker Compose?



Šta je Docker Compose?

- Docker Compose je alat koji omogućava definisanje i izvršavanje više-kontejnerskih Docker aplikacija.
  - Moguće je koristiti Docker Compose i za jedno-kontejnerske aplikacije, ali više smisla ima za više-kontejnerske
- Prednosti Docker Compose-a:
  - ubrzava rad sa dockerom
  - obezbeđuje preglednost
  - olakšava deljenje docker aplikacije
- Koristi se YAML fajl za definisanje servisa, mreža i volume-a Docker aplikacije

Šta nije Docker Compose?

- Docker Compose ne može da zameni pisanje pojedinačnih Dockerfile-ova za kreiranje slika kontejnera
- Docker Compose ne zamenjuje slike kontejnera niti kontejnere već samo olakšava rad sa njima
- Docker Compose nije pogodan za upravljanje više-kontejnerskim aplikacijama koje se nalaze na nekoliko host-ova (mašina)
    - za to se koriste alati poput Kubernetes-a

Primer compose.yaml

```yaml
version: "3.9"  # optional since v1.27.0
services:
 web:
   build: .
   ports:
     - "8000:5000"
   volumes:
     - .:/code
     - logvolume01:/var/log
   depends_on:
     - redis
 redis:
   image: redis
volumes:
 logvolume01: {}
```

YAML

- *human-readable* jezik za serijalizaciju podataka
- Često se koristi za konfiguracione fajlove, kao i u aplikacijama koje skladište ili šalju podatke
- Originalni značenje YAML jezika - *Yet another markup language*
    - Zbog promene namene ovog fajl značenje naziva je takođe promenjeno u *Yaml ain't markup language*
        - Kako bi se označila razlika YAML jezika koji je orijentisan na podatke i XML, HTML, … jezika koji su namenjeni za markup dokumenata
- Zvanična ekstenzija za fajlove napisane u YAML jeziku je .yaml
- Zvanična veb stranice YAML jezika: https://yaml.org/
- YAML je kombinacija:
    - indentacije kao u Python jeziku
    - i kompaktnog formata koji koristi [...] za liste i {...} za mape
        - Zbog toga su JSON fajlovi ujedno i validni YAML fajlovi

YAML sintaksa

- Jednolinijske komentare YAML procesor preskače
  - # ispred komentara
  - Komentar mora biti odvojen od ostatka makar jednim whitespace-om
- Whitespace je deo YAML formatiranja
  - Za ugnježdavanje rečnika
  - Može biti jedan ili više space-ova (**PREPORUČENO 2**)
  - Tabovi su zabranjeni jer ih različiti alati drugačije interpretiraju
- Novi red označava kraj datog polja u YAML fajlu

YAML sintaksa

- Osnovna gradivna jedinica za YAML je ***par ključ-vrednost***
  - Svaki podatak u YAML dokumentu je član barem jednog rečnika
  - Ključ je uvek string
  - Vrednost je skalar tako da može biti bilo koji tip podatka (string, broj ili rečnik)
- Stringovi najčešće ne treba da se definišu pod navodnicima
  - Osim ako postoji neki escape sequence koga želimo da izbegnemo - dupli navodnici

YAML sintaksa - Lista

- Liste mogu da se definišu u jednoj liniji ili u više linija
- Više-linijsko definisanje liste je zgodno kada su elementi liste kompleksni objekti a ne skalari
  - Svaki element u listi započinje sa -
  - Svaki element u listi mora biti uvučen jednakim brojem whitespace-ova u odnosu na naziv te liste
- Primer:

```
items: [ 1, 2, 3 ]
numbers:
  - 1
  - 2
  - 3
```

YAML sintaksa - Rečnik

- Rečnici mogu da se definišu u jednoj liniji ili u više linija
- Više-linijsko definisanje rečnika je zgodno kada su elementi rečnika kompleksni objekti a ne skalari
  - Svaki element u rečniku mora biti uvučen jednakim brojem whitespace-ova u odnosu na naziv tog rečnika
- Primer:

```
items: [ one: 1, two: 2, three: 3 ]
four: 4
numbers:
 one: 1
 two: 2
 three: 3
```

Primeri razlike između rečnika i liste u YAML-u

https://gist.github.com/carlessanagustin/50dab6d642e34f8f617d

Compose model

- Compose specifikacija omogućava definisanje aplikacije zasnovane na kontejnerima koja je platformski agnostična
  - Jedna takva aplikacija je dizajnirana kao skup kontejnera koji su pokrenuti zajedno, dele resurse i komuniciraju putem dodeljenih kanala
- Compose fajl se sastoji iz:
  - **Servisa (services)**
    - Computing komponente aplikacije
    - Apstraktni koncept koji se implementira na platformi pokretanjem iste slike kontejnera jednom ili više puta
  - **Mreže (networks)**
    - Način komunikacije između servisa
  - **Docker skladišta (volumes)**
    - Način skladištenja i deljenja perzistentnih podataka između servisa
  - **Konfiguracija (configs)**
  - **Tajne (secrets)**

Compose fajl

- Predefinisana putanja za compose fajl je:
  - ***compose .yaml***
    - Može i compose.yml
  - docker-compose.yaml / docker-compose.yml
    - Trebalo bi da radi zbog unazadne kompatibilnosti ali nije zagarantovano

Docker compose - top level elementi

- Docker compose fajl ima nekoliko top-level elemenata:
  - **Version (deprecated)**
  - **Name**
  - **Services**
  - **Networks**
  - **Volumes**
  - **Configs**
  - **Secrets**
- Ovi su rezervisane reči koje definišu model compose fajla
  - docker-compose mora da sadrži date reči za definisanje određenih elemenata u compose fajlu
    - U suprotnom konfiguracija neće biti uvažena

Docker compose klijent [name]

- top-level element koji određuje naziv projekta
- Postoji predefinisan naziv ukoliko korisnik izostavi ***name*** element u compose fajlu
- Naziv projektna je dostupan putem environment varijable **COMPOSE_PROJECT_NAME**

```
name: "Project"
services:
 foo:
   image: busybox
   environment:
     - COMPOSE_PROJECT_NAME
   command: echo "I'm running ${COMPOSE_PROJECT_NAME}"
```

Docker compose klijent [services]

- Servis je apstraktna definicija računarskog resursa u aplikaciji
  - Može da se skalira/zameni sa drugom komponentom
- Docker compose fajl **MORA** da deklariše **services** korenski element
  - Predstavlja mapu čiji ključevi su stringovi koji reprezentuju nazive servisa, a vrednosti su definicije tih servisa
- Definicija servisa je konfiguracija koja se primenjuje na svaki kontejner tog servisa
- Svaki servis može da uključi Build sekciju koja definiše kako se kreira Docker slika za taj servis
  - Build sekcija nije obavezna
- Svi servisi moraju biti jednako indentovani da bi se fajl smatrao validnim

Docker compose klijent [services - build]

- **build** specificira konfiguraciju za kreiranje slika kontejnera na osnovu Dockerfile-a
- ako postoji build sekcija, image sekcija može da izostane i obrnuto
  - Ako postoje obe sekcije image sekcija ima prednost
- build ima kratku i dužu sintaksu
  - Kratka: build: ./dir
    - putanja za build context (tu mora da se nalazi i Dockerfile)
  - Duža:
    - context polje (OBAVEZNO)
      - sadrži ili putanju do build contexta u kome se nalazi Dockerfile-a ili url do repozitorijuma
    - dockerfile
      - dozvoljava podešavanje alternativnog Dockerfile-a (koji se nalazi van build contexta)

Docker compose klijent [services - build]

```
services:
 frontend:
   image: awesome/webapp
   build: ./webapp

 backend:
   image: awesome/database
   build:
     context: backend
     dockerfile: ../backend.Dockerfile

 custom:
   build: ~/custom
```

Docker compose klijent [services - command]

- **command** override-uje predefinisane komande deklarisane od strane slike kontejnera (u okviru Dockerfile CMD komande)

```
command: bundle exec thin -p 3000
command: [ "bundle", "exec", "thin", "-p", "3000" ]
```

Docker compose klijent [services - container_name]

- container_name je string koji specificira naziv kontejnera
  - Svaki kontejner ima i predefinisani naziv
- Compose implementacija ne sme da skalira servis za više od jednog kontejnera ukoliko compose fajl sadrži specifikaciju naziva kontejnera
  - Mora rezultovati greškom
- Naziv kontejnera mora da poštuje sledeći regex format: [a-zA-Z0-9][a-zA-Z0-9_.-]+

```
container_name: my-web-container
```

Docker compose klijent [services - depends_on]

- depends_on izražava startup i shutdown zavisnosti između servisa
- Compose implementacija garantuje da će prvo pokrenuti servis od koga zavisi neki drugi servis, pa tek onda onaj servis koji se oslanja na prethodno podignuti servis
- Duža sintaksa dozvoljava i definisanje stanja kontejnera: service_started (predefinisan za kraću sintaksu), service_healthy i service_completed_successfully

```
frontend:
  depends_on:
    - backend
backend:
  depends_on:
    database:
      condition: service_healthy
```

Docker compose klijent [services - environment]

- environment definiše environment varijable koje važe u okviru kontejnera
  - Može da se definiše preko liste ili rečnika

```
environment:
 RACK_ENV: development
 SHOW: "true"

environment:
 - RACK_ENV=development
 - SHOW=true
 - USER_INPUT
```

Docker compose klijent [services - healthcheck]

- healthcheck deklariše proveru dostupnosti kontejnera
  - Da li je kontejner zdrav?
- Ova konfigracija override-uje HEALTHCHECK naredbu u Dockerfile-u

```
healthcheck:
 test: ["CMD", "curl", "-f", "http://localhost"]
 interval: 1m30s
 timeout: 10s
 retries: 3
 start_period: 40s
```

Docker compose klijent [services - image]

- image definiše sliku na osnovu koje će se kreirati kontejner
- image može da bude izostavljen iz compose fajla ukoliko je build sekcija deklarisana
- ukoliko definisana slika ne postoji na host-u, compose će pokušati da je povuče sa predefinisanog registra

```
image: redis
   image: redis:5
   image: redis@sha256:0ed5d5928d4737458944eb604cc8509e245c3e19d02ad83935398bc4b991aac7
   image: library/redis
   image: docker.io/library/redis
   image: my_private.registry:5000/redis
```

Docker compose klijent [services - networks]

- **networks** definiše mrežu na koju su zakačeni kontejneri tog servisa
  - Mreža koja se navodi mora da bude definisana u okviru ***networks*** top-level elementa

```
services:
 some-service:
   networks:
     - some-network
     - other-network

networks:
  some-network:
  other-network:
```

Docker compose klijent [services - volumes]

- ***volumes*** definiše mount putanju hosta ili kreira imenovano Docker skladište koje mora biti dostupno kontejnerima tog servisa
- Jedino imenovani docker volume-i moraju biti definisani i u okviru volumes top-level elementa
- Postoje dve vrste sintakse za definisanje volumes elementa
    - kratka
    - duga

Docker compose klijent [services - volumes]

- Kratka sintaksa:
  - VOLUME:CONTAINER_PATH:ACCESS_MODE
    - VOLUME - može biti host putanja (bind mount) ili naziv imenovanog volume-a
    - CONTAINER_PATH - putanja u kontejneru gde je volume mountovan
    - ACCESS_MODE
      - rw - read and write (default)
      - ro - read-only

Docker compose klijent [services - volumes]

- Duga sintaksa:
  - Omogućava definisanju dodatnih polja koja ne mogu biti izražena u kratkoj formi
    - ***type***: tip mount-a volume, bind, tmpfs ili npipe
    - ***source***: izvor mount-a, putanja na host mašini (bind mount) ili naziv volume-a definition u top-level volumes elementu
    - ***target***: putanja u kontejneru gde je volume mount-ovan
    - ***read-only***: flag za podešavanje volume-a kao read-only
    - + dodatna polja

Docker compose klijent [services - volumes]

```
services:
 backend:
   image: awesome/backend
   volumes:
     - type: volume
       source: db-data
       target: /data
       volume:
         nocopy: true
     - type: bind
       source: /var/run/postgres/postgres.sock
       target: /var/run/postgres/postgres.sock

volumes:
 db-data:
```

Docker compose klijent [services - ports]

- ports definiše portove na kome će se nalaziti dati kontejner
- Kratka sintaksa: [HOST:]CONTAINER[/PROTOCOL]
  - HOST - port na host mašini
  - CONTAINER - port u Docker mreži
  - PROTOCOL - definisani protokol TCP/UDP

```
ports:
      - "3000"
      - "3000-3005"
      - "8000:8000"
      - "9090-9091:8080-8081"
      - "127.0.0.1:8001:8001"
      - "127.0.0.1:5000-5010:5000-5010"
      - "6060:6060/udp"
```

Docker compose klijent [services - stdin_open & tty]

- opcija -it prilikom pokretanja kontejnera (bez docker-compose) omogućava pristup terminalu u okviru pokrenutog kontejnera
- da bi se ta opcija omogućila i iz docker-compose fajla:
  - stdin_open: true
  - tty: true

Docker compose klijent [networks]

- **networks** je top-level element koji definiše mrežu na koju se kontejneri mogu prikačiti
- **networks** ne mora da se definiše u docker-compose fajlu
    - docker-compose predefinisano kreira mrežu na koju prikači sve kontejnere čiji servisi su definisani u fajlu
- mreža se kreira definisanje naziva mreža
- servisi se mogu konektovati na datu mrežu definisanjem naziva mreže ispod networks podsekcije u oviru definicije servisa

Docker compose klijent [networks]

- primer definisanje nove mreže i konektovanja servisa na datu mrežu

```
services:
  frontend:
    image: awesome/webapp
    networks:
      - front-tier
      - back-tier

networks:
  front-tier:
  back-tier:
```

Docker compose klijent [volumes]

- volumes je top-level element koji definiše perzistentna skladišta podataka
- Omogućava konfiguraciju imenovanih volume-a koji mogu da se koriste za više razlčitih servisa

```
services:
 backend:
   image: awesome/database
   volumes:
     - db-data:/etc/data
 backup:
   image: backup-service
   volumes:
     - db-data:/var/lib/backup/data
volumes:
 db-data:
```

Docker compose CLI

- **docker-compose up**
  - kreira i pokrene kontejnere
  - svakim pozivom se prvo proveri da li treba da se ponovo izbilda slika - da li ima izmena
  - ako već postoje kreirani kontejneri za servis definisan u compose fajlu, ali se konfiguracija ili slika kontejnera promenila nakon kreiranja kontejnera, pozivom docker-compose up kontejneri se zaustavljaju i rekreiraju
- **docker-compose up -d**
  - pokretanje u detached režimu
- **docker-compose down**
  - zaustavi kontejnere i ukloni:
    - kontejnere i
    - mreže
- **docker-compose down -v**
  - opcija za brisanje volume-a nakon zaustavljanja kontejnera
- **docker-compose down --rmi <opcija>**
  - opcija za brisanje slika kontejnera koje su servisi koristili
  - opcija (“local”|”all”)
    - local briše samo slike koje nemaju tag

Zadatak 2

- Napisati compose fajl za aplikaciju iz zadatka 1

Zadatak 3

- Napisati compose fajl za Postgres + Django aplikaciju

Materijali:

- https://docs.docker.com/compose/
- https://www.cloudbees.com/blog/yaml-tutorial-everything-you-need-get-started
- https://www.igordejanovic.net/courses/tech/docker/