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

Računarstvo u oblaku

ms Helena Anišić

Zimski semester 2024/2025.

Studijski program: Računarstvo i automatika

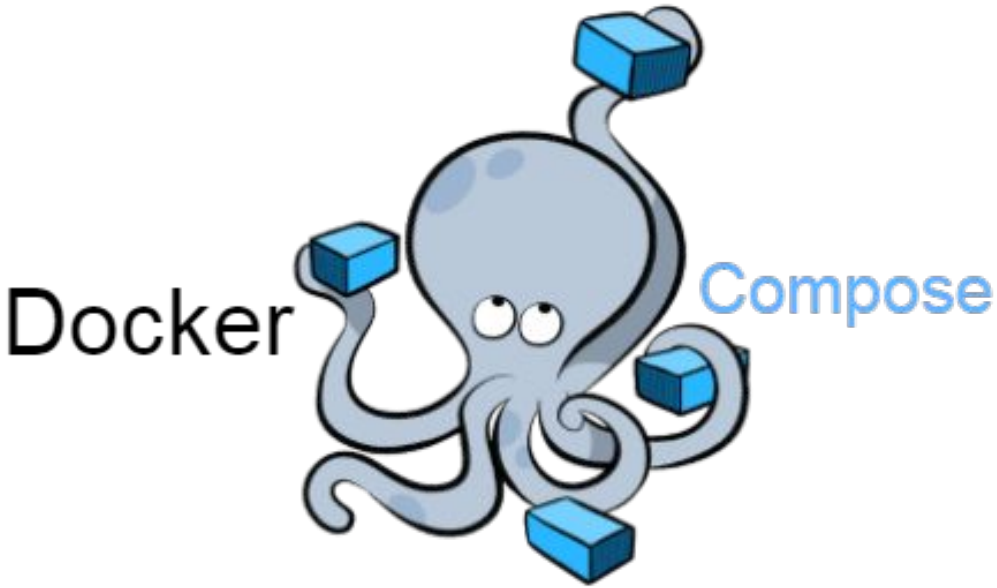
Modul: Računarstvo visokih performansi

Zadatak 1

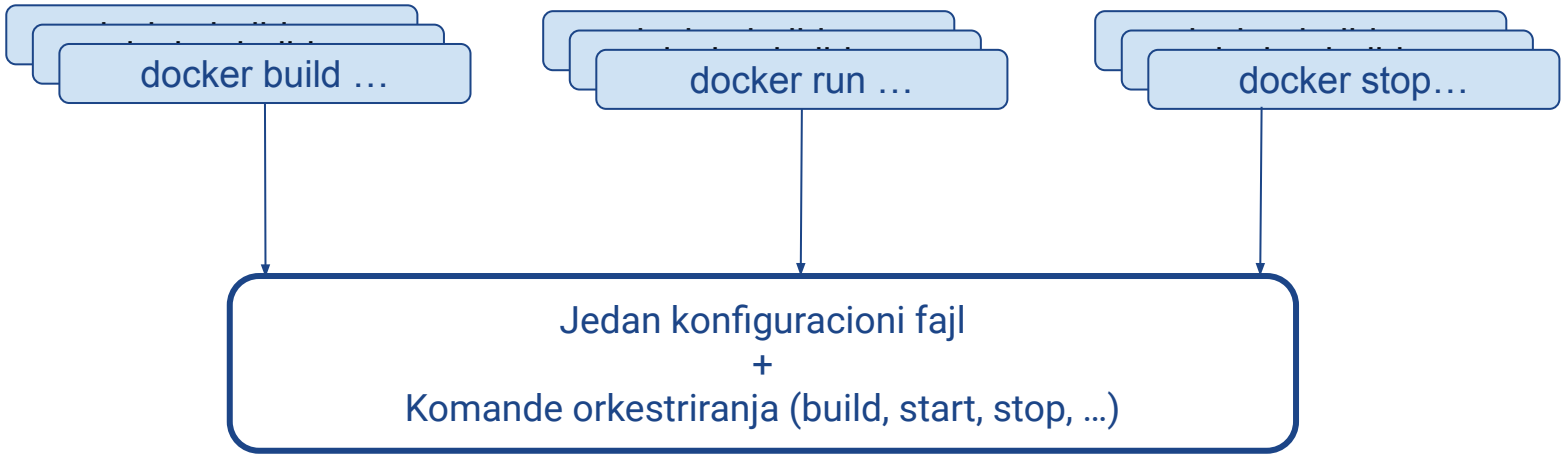
- Potrebno je kontejnerizovati jedan tipičan primer veb aplikacije.
- Delovi veb aplikacije su:
  - mongo baza podataka (naziv zvanične slike: **mongo)**
    - trajno perzistiranje podataka: **/data/db** putanja u mongo kontejneru gde se čuvaju podaci
  - node.js backend aplikacija
    - trajno perzistiranje logova: **/logs** putanja u aplikaciji
    - live reload koda (bez potrebe da se build-uje slika docker kontejnera nakon izmene koda)
    - node_modules i Dockerfile ne treba da se kopiraju u kontejner
  - react frontend aplikacija
    - **react aplikacija mora da se pokrene sa -it opcijom (u novijim verzijama može i bez ove opcije)**
    - live reload koda (bez potrebe da se builduje slika docker kontejnera nakon izmene koda)
    - node_modules i Dockerfile ne treba da se kopiraju u kontejner

Zadatak 1

- Kako olakšati, ubrzati i učiniti preglednijim build-ovanje, 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 YAML konfiguracioni fajl za definisanje servisa, mreža i skladišta podataka 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

Instalacija Docker Compose

- Informacije o instalaciji Compose alata: https://docs.docker.com/compose/install/

Primer [*compose.yaml / docker-compose.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

- Ljudski čitljiv 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 fajla 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 specifikacija - *top level* elementi

- Compose fajl specifikacija može da se sastoji iz sledećih *top-level* elemenata:
  - Version (optional)
  - 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

Compose specifikacija [*version*]

- *top-level* element koji je samo informativan
- compose ne koristi `version` element kako bi odredio tačnu šemu sa kojom treba da validira fajl
    - svakako koristi najnoviju šemu

Compose specifikacija [*name*]

- *top-level* element koji određuje naziv projekta
- postoji predefinisan naziv ukoliko korisnik izostavi `name` element u compose fajlu
- naziv projekta 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}"
```

Compose specifikacija [*services*]

- `services` je *top-level* element koji objedinjuje pojedinačne servise definisane u compose fajlu
- Compose fajl mora da sadrži `services` *top-level* element
  - predstavlja mapu čiji ključevi su stringovi koji reprezentuju nazive servisa, a vrednosti su definicije tih servisa
- Servis je apstraktna definicija računarskog resursa u aplikaciji
  - može da se skalira/zameni sa drugom komponentom
- 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 uvučeni u odnosu na `services` element da bi se compose fajl smatrao validnim

Compose specifikacija [*services - image*]

- `image` definiše na osnovu koje slike kontejnera da se pokrene kontejner
- `image` mora da prati Open Container Spcefication (OCI) format
- `[<registry>/][<project>/]<image>[:<tag>|@<digest>]`
- Ako navedena slika ne postoji na platformi, preuzima se slika predefinisano sa *DockerHub* registra slika
- ako postoji `build` sekcija, `image` sekcija može da izostane i obrnuto
  - ako postoje obe sekcije `image` sekcija ima prednost

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

Compose specifikacija [*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` polje
      - dozvoljava podešavanje alternativnog Dockerfile-a (koji se nalazi van build contexta)

Compose specifikacija [*services - build*]

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

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

 custom:
   build: ~/custom
```

Compose specifikacija [*services - command*]

- `command` *override*-uje predefinisane komande deklarisane od strane slike kontejnera
  - u okviru Dockerfile to je `CMD` komanda
- `command` može da se napiše i u obliku liste slično kao u Dockerfile-u
- Ako je vrednost `command` naredbe null, koristi se predefinisana naredbe iz slike
- Ako je vrednost `command` naredbe `[]` (prazna lista) ili `''` (prazan string), predefinisana naredba iz slike se ignoriše i postaje prazna

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

Compose specifikacija [*services - container_name*]

- `container_name` je string koji definiše naziv kontejnera
  - svaki kontejner ima i predefinisani naziv koji se na ovaj način *override*-uje
- Compose implementacija ne sme da skalira servis za više od jednog kontejnera ukoliko compose fajl sadrži specifikaciju naziva kontejnera
  - problem nastaje pošto u toj situaciji compose implementacija pokušava da kreira više kontejnera sa istim nazivima
- Naziv kontejnera mora da poštuje sledeći RegEx format: [a-zA-Z0-9][a-zA-Z0-9_.-]+

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

Compose specifikacija [*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
```

Compose specifikacija [*services - environment*]

- `environment` definiše *environment* varijable koje važe u okviru kontejnera
  - može da se definiše preko liste ili rečnika

```
#map syntax
environment:
 RACK_ENV: development
 SHOW: "true"

#array sytax
environment:
 - RACK_ENV=development
 - SHOW=true
 - USER_INPUT
```

Compose specifikacija [*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
```

Compose specifikacija [*services - networks*]

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

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

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

Compose specifikacija [*services - volumes*]

- `volumes` definiše *mount* putanju *host*-a ili kreira Docker skladište (*volume*) 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

Compose specifikacija [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 mount-ovan
    - ACCESS_MODE
      - rw - read and write (default)
      - ro - read-only

Compose specifikacija [*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

Compose specifikacija [*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:
```

Compose specifikacija [*services - volumes*]

- Primer kraće sintakse:

```
services:
 backend:
   image: awesome/backend
   volumes:
     - db-data:/data #named volume
     - /some_data #anonymous volume
     - ./host:/app #bind mount

volumes:
 db-data:
```

Compose specifikacija [*services - ports*]

- `ports` definiše portove na kojima 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"
```

Compose specifikacija [*services - stdin_open & tty*]

- opcija `-it` prilikom pokretanja kontejnera omogućava pristup terminalu u okviru pokrenutog kontejnera
- da bi se ta opcija omogućila i prilikom upotrebe compose fajla potrebno je navesti sledeće:
  - `stdin_open: true`
  - `tty: true`

Compose specifikacija [*networks*]

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

Compose specifikacija [*networks*]

- primer definisanje nove mreže i povezivanja servisa na datu mrežu

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

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

Compose specifikacija [*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:
```

Compose specifikacija [*configs*]

- `configs` je *top-level* element koji omogućava da se servisi prilagode novim podešavanjima bez potrebe da se slika kontejnera ponovo kreira
- Servisi mogu da koriste `configs` samo kada im je to eksplicitno dozvoljeno putem `configs` atributa

```
services:
  redis:
    image: redis:latest
    configs:
      - my_config
configs:
  my_config:
    file: ./my_config.txt
```

Compose specifikacija [*secrets*]

- `secrets` je vrsta configs elementa fokusirana na osetljive podatke
- Servisi mogu da koriste `secrets` samo kada im je to eksplicitno dozvoljeno putem `secrets` atributa

```
services:
  frontend:
      image: example/webapp
      secrets:
      - server-certificate
secrets:
  server-certificate:
      file: ./server.cert
```

Docker Compose CLI

- `docker compose up`
  - kreira sliku kontejnera, (re)kreira kontejner i *attach*-uje se na pokrenuti kontejner
  - svakim pozivom se prvo proveri da li treba da se ponovo kreira slika - da li ima izmena u odnosu na
  - 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 CLI

- `docker compose down`
  - zaustavi kontejnere i ukloni:
    - kontejnere i
    - mreže
- `docker compose down -v`
  - opcija za brisanje imenovanih i anonimnih skladišta podataka (engl. *volume*) nakon zaustavljanja i brisanja 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 1. zadatka

Zadatak 3

- Napisati compose fajl za Postgres + Django aplikaciju sa prethodnih vežbi

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/