Ime:
Priimek:
Vpisna št:
Datum:

Osnove virtualizacije in kontejnerizacije
Praktični primeri dela s tehnologijo Docker

Namen

- Spoznavanje osnovnih konceptov virtualizacije in kontejnerizacije
- Spoznavanje problematike lansiranja aplikacij v raznolika okolja
- Spoznavanje arhitekture in delovanja tehnolgije Docker
- Prodobivanje praktičnih izkušenj uporabe aplikacijskih zabojnikov
- Modeliranje aplikacijskih arhitektur na osnovi mikrostoritev

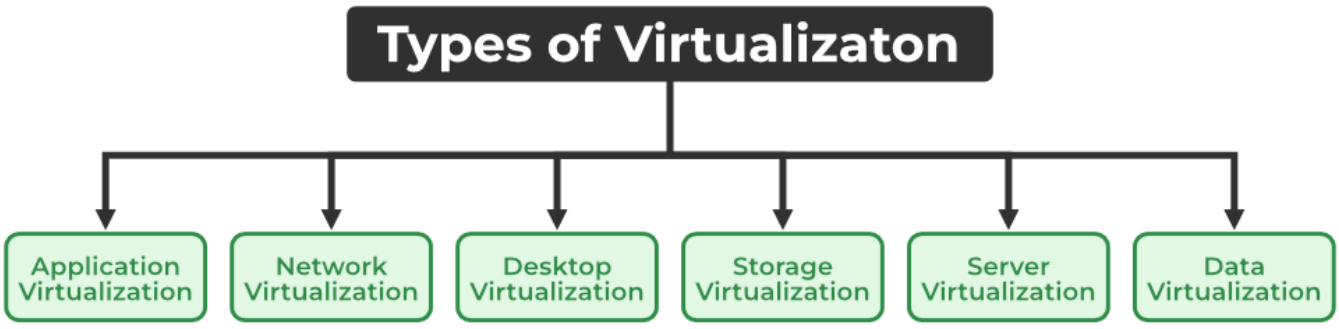
Naloga

Spoznajte se z osnovnimi koncepti virtualizacije in kontejnerizacije ter odgovorite na vprašanja za utrjevanje znanja. Spoznajte se z arhitekturo in delovanjem tehnologije Docker ter uporabite nekaj predpripravljenih aplikacijskih zabojnikov. Naučite se upravljanja s sistemskimi slikami, zabojniki in omrežnimi ter podatkovnimi viri. Izdelajte preprosto spletno stran ter jo zapakirajte in lansirajte v lasten aplikacijski zabojnik. Spletno stran lansirajte (angl. deploy) z uporabo okolja Docker, najprej v razvojni različici, nato še v produkcijski obliki (t.i. kontejnerizacija, Dockerfile, dobre razvojne in varnostne prakse). Nazadnje izdelajte model kompleksnejšega projekta, ki sestoji iz povezanega sistema mikrostoritev ter omogoča preprosto lansiranje v raznolikih okoljih. Za izvedbo vaje si lahko pomagate z vsemi razpoložljivimi spletnimi viri (npr.: Google, StackOverflow, ChatGPT, ipd.).

1 Teoretični uvod v vajo

1.1 Predstavitev problematike

V okviru skupinske razprave se spoznajte s ključnimi sodobnimi tehnologijami, ki omogočajo računalništvo v oblaku (virtualizacija, kontejnerizacija, ipd.). Ugotovite kakšne so njihove prednosti in slabosti. Identificirajte razlike med načini virtualizacije in kontejnerizacije ter ugotovite kakšne so tipične težave pri lansiranju aplikacij v različna produkcijska okolja.



V kontekstu računalništva v oblaku in strežniške virtualizacije odgovorite na naslednja vprašanja:

Naštejte nekaj ključnih prednosti računalnišva v oblaku, ki jih omogoča virtualizaija:

_____

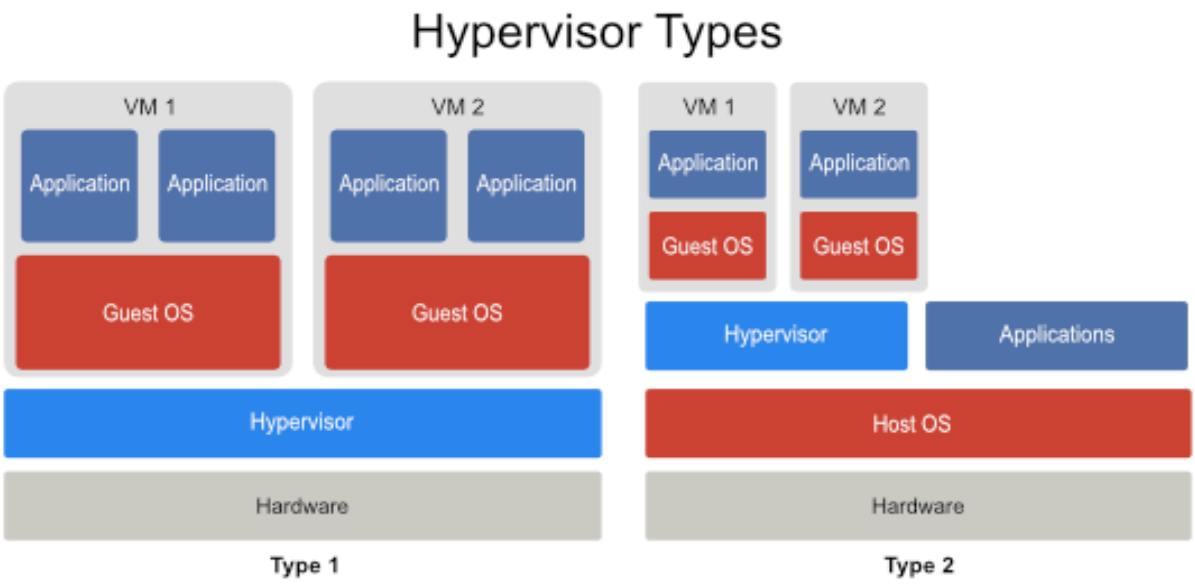
_____

1.2 Virtualizacija

Virtualizacija je tehnološki pristop, ki omogoča hkratno izvajanje več virtualnih sistemov na eni fizični napravi. Gre za postopek abstrakcije strojne opreme, kjer je fizična naprava razdeljena na več **virtualnih strojev (angl. virtual machine – VM)**, pri čemer vsak virtualni stroj deluje kot samostojen računalnik s svojim operacijskim sistemom in aplikacijami. V kontekstu virtualizacije torej govorimo o gostitelju (angl. **host**) na katerem teče več gostujočih VM-jev (angl. **guests**).

Virtualizacijo na gostitelju omogoča programska oprema, imenovana **hipervizor**, ki skrbi za deljenje strojne opreme med več virtualnimi okolji. Hipervizor nadzoruje in upravlja razporeditev pravih strojnih virov, kot so CPU, RAM, omrežje in diskovna zmogljivost z ustvarjanjem **virtualnih strojnih virov**, ki jih uporabljajo VM-ji.

Poznamo dva tipa hipervizorjev, tip 1 oz. »bare-metal« in tip 2 oz. »hosted«:



Strojni viri gostitelja so lahko neposredno virtualizirani (ob ustrezni strojni podpiri npr. VT-x/AMD-V ter z uporabo učinkovitih vmesnikov npr. VirtIO) ali pa so **emulirani,** če govorimo o virih, ki simulirajo delovanje strojne opreme ali druge arhitekture. Prav tako lahko nekatere I/O strojne vire tudi direktno pripnemo gostujočim VM-jem (angl. passthrough) ob ustrezni strojni podpori platforme (npr. VT-d).

Odgovorite na naslednja vprašanja na temo virtualizacije:

Kakšne so razlike med hipervizorjem tipa 1 in 2? Naštejte njune prednosti in slabosti:

_____

Navedite vsaj dva »bare-metal« in dva »hosted« hipervizorja:

_____

Navedite primer emulatorja ali primer emulacije med dvema procesorskima ahitekturama:

_____

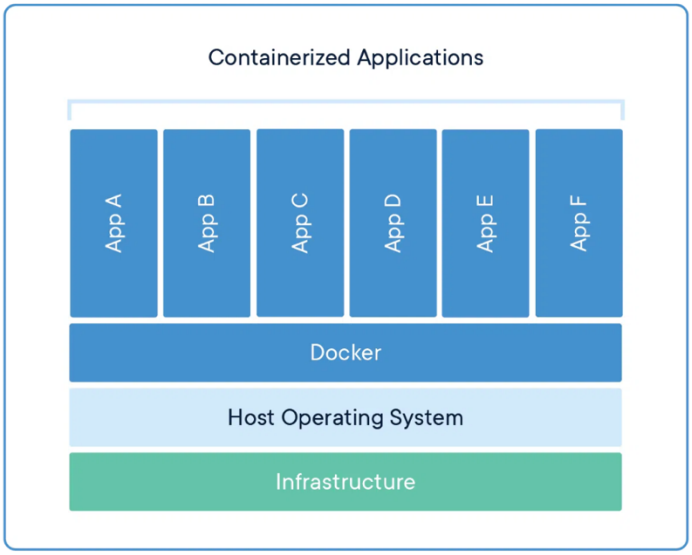
Ali so hitrejši oz. bolj učnkoviti emulirani ali virtualizirani viri? Kakšna je tipična uporaba »passthrough« načina?

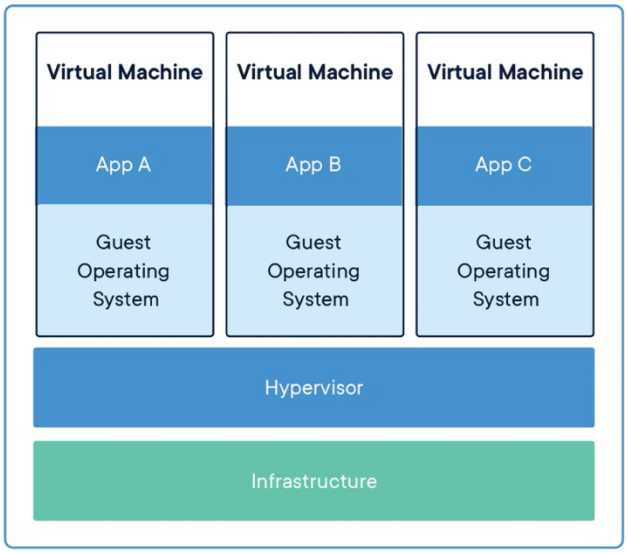
_____

1.3 Kontejnerizacija

Kontejnerizacija je tehnološki pristop, ki omogoča izvajanje programske opreme oz. procesov v **izoliranih virtualnih okoljih**, imenovanih kontejnerji. Gre za metodo virtualizacije, ki deluje **na nivoju operacijskega sistema gostitelja** in omogoča **izolacijo procesov in virov brez potrebe po ločenih virtulanih strojih in gostujočih operacijskih sistemih** za vsako okolje.

Ključna razlika med virtualizacjo in kontejnerizacijo je v **deljenju jedra operacijskega sistema**. Orodja za kontejnerizacijo delujejo na principu **izolacije virov na operacijskem sistemu gostitelja**, zato si kontejnerizirane aplikacije z gostiteljem delijo jedro. Pri virtualizaciji pa hipervizor ustvari popolnoma ločeno virtualno strojno opremo na katero namesti celoten operacijski sistem vključno z njegovim jedrom.





Kontejnerizacija pod pokrovom temelji na mehanizmih Linux jedra kot sta cgroups in namespaces, ki lahko procese na sistemu razporedita v poljubne kontrolne skupine in jim dodelita različen dostop do virov (npr. omejitev CPU časa, omejitev količine RAM pomnilnika) ter različno vidljivost oz. izolacijo virov (npr. izolirano omrežje, omejitev vidljivosti in komunikacje z drugimi procesi, izoliran datotečni sistem).

Sprva se je kontejnerizacija osredotočala predvsem na **izolacijo posamičnih aplikacij** (enega oz. manjšega števila procesov, ki tvorijo aplikacijo in okolje za njen zagon oz. delovanje), vendar danes vključuje tudi **kontejnerizacijo variant operacijskih sistemov** (npr. Alpine, Ubuntu, Debian, Fedora, ob deljenju jedra z gostiteljem) kar lahko v nekaterih primerih nadomesti potrebo po virtualizaciji.

Opišite razlike med klasično virtualizacijo ter kontejnerizacijo:

_____

Naštejte prednosti kontjnerjev pred virtualnimi stroji in obratno (učinkovitost, varnost, izolacija, ipd.):

_____

Katere tehnologije kontejnerizacije poznate? Naštejte vsaj eno iz vsake vrste kontejnerizacije:

_____

Kakšna je tipična razlika med namembnostjo aplikacijskih kontejnerjev (Docker) in OS-kontejnerjev (LXC)?

_____

1.4 Orkestracija in mikrostoritve

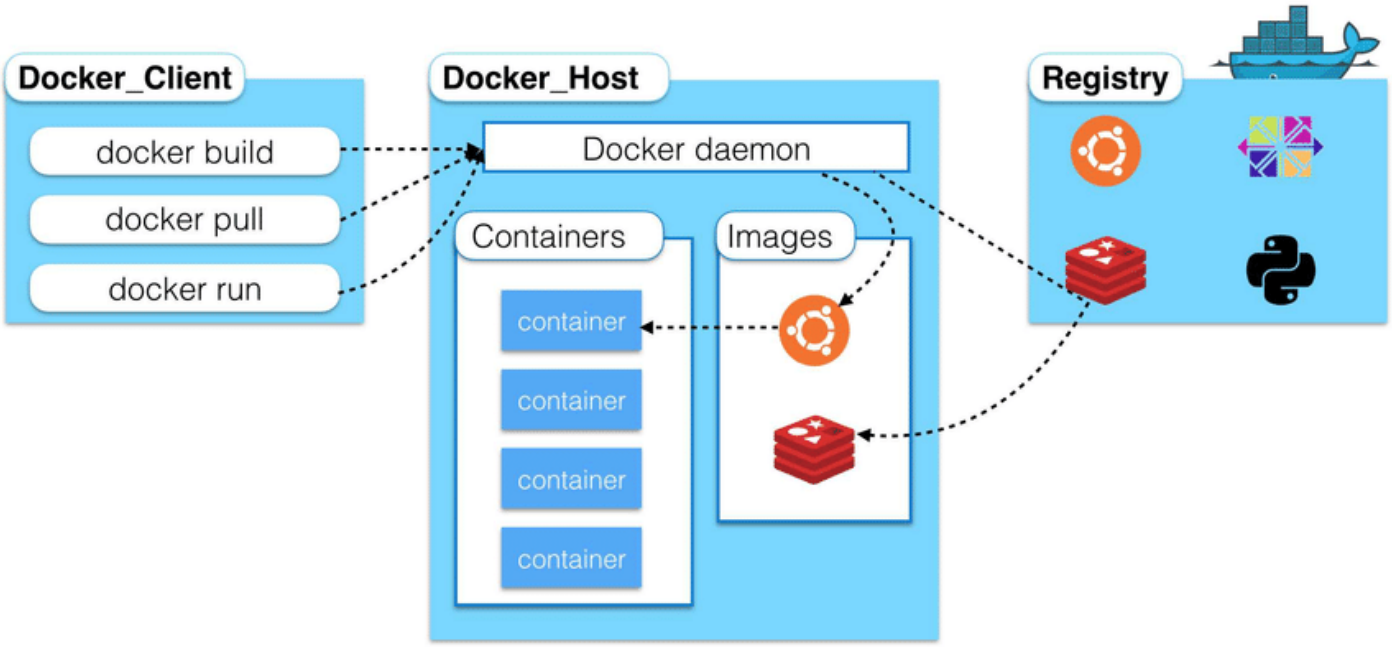
Z vse večjo uporabo virtualizacije in kontejnerizacije se pojavlja tudi potreba po sistemih za množično in/ali samodejno upravljanje VM-jev ter kontejnerjev (orkestracijo). Za upravljanje z VM in OS kontejnerji tipično uporabljamo virtualizacijska okolja in orodja, npr. VMWare vCenter, Proxmox, ipd.

Hkrati sodobni prstopi k razvoju programske opreme narekujejo tudi **funkcionalno ločevanje** posameznih delov aplikacij v samostojne mikro kontejnerje, ki so medsebojno povezani z različnimi omrežji – t.i. **arhitektura mikrostoritev**. Tovrstno drobljenje programske logike omogoča večjo prožnost aplikacije in njeno **skalabilnost.** Za upravljanje z veliko količino aplikacijskih kontejnerjev in njihovih virov pa poskrbimo z orodji kot je Docker Compose ali orkestracijskimi sistemi kakršna sta npr. Docker Swarm in Kubernetes. Le-ta omogočajo samodejno podvajanje storitev ob povečani obremenitvi (skaliranje) in zagotvaljanje visoke zanesljivosti s samodejnim upravljanjem kontejnerjev na različnih fizičnih gostiteljih.

2 Uvod v okolje Docker

2.1 Docker arhitektura

Docker je odprtokodna platforma za avtomatizacijo razvoja, dostavo in izvajanje aplikacij z uporabo kontejnerjev. Kontejnerji so lahki, prenosljivi in zagotavljajo dosledno okolje za izvajanje aplikacij, ne glede na to, kje se nahajajo. Osnovne arhitektura okolja Docker je povzeta na spodnji sliki.



Osnovne komponente okolja Docker zajemajo:

Docker Engine	Docker Client (API/GUI/CLI)	Dockerfile	Docker Swarm
Docker Images	Docker Networking	Docker Compose	Kubernetes
Docker Containers	Docker Volumes	Docker Hub (Repository)	

Osnova za delovanje kontejnerja je nejgova **sistemska slika**. Ta sestoji iz več **slojev**. Ko sliko poženemo smo ustvarili **kontejner**. Sistemske slike predpripravljenih aplikacij si razvijalci delijo med seboj v **registrih**. Shranjevanje slik v slojih omogoča prostorsko optimizacijo in preprosto dograjevanje aplikacij. Izgradnjo slik razvijalci definirajo s predpisom dejanjv datotekah **Dockerfile**. Sočasno poganjaje večih kontejnerjev iz slik in upravljanje njihovih nastavitev lahko predpišemo v datoteki **docker-compose**.

2.2 Naloga: postavitev razvojnega spletnega strežnika (iz predpripravljene aplikacijke slike)

Opomba: pred pričetkom vaje bo morda potrebna registracija in prijava v Docker GUI z Hub računom.

Na laboratorijskih računalnikih je že nameščeno okolje Docker for Windows (temelji na WSL2). **Z uporabo Dockerja ustvarite in zaženite nov spletni strežnik s podporo PHP. Strežnik naj bo z vašega računalnika dostopen na vratih 8080. Z gostiteljskim operacijskim sistemom naj si deli indeksni direktorij na namizju.** Uporabite predpripravljene sistemske slike ter klientna orodja Docker GUI ali CLI. V pomoč so vam lahko sledeči ukazi.

```
docker --help
docker pull php:apache
docker run -d \
    --name streznik \
    --publish 8080:80 \
    --mount type=bind,source=/pot/do/namizja/mapa,target=/var/www/html \
    php:apache

docker ps
docker logs <container>
docker inspect <container>
docker container ls (-a)
docker image ls
docker stop <container>
docker start <container>
docker exec -it <container>
docker cp <container>:
```

Ustvarite vzorčno PHP datoteko in preverite delovanje strežnika. Nato datoteko spremenite in osvežite stran.

Za vsebino strani lahko uporabite primer preproste PHP datoteke (index.php).

```
<?php
      echo “Hello World, this is PHP speaking!”;
?>
```

Kakšen je vpliv argumenta –mount na zunanje spremembe datotek med delovanjem kontejnerja?

_____

Kako deluje argument –publish?

_____

Na kakšne načine lahko preverite verzijo in podrobnosti PHP v kontejnerju?

_____

V tej nalogi ste z dvemi ukazi iz Docker slike vzpostavili celoten PHP in Apache strežnik. Kot zanimivost lahko primerjate količino potrebnega dela, če bi strežnik konfigurirali brez predpripravljene Docker slike: https://www.digitalocean.com/community/tutorials/how-to-install-lamp-stack-on-ubuntu

2.3 Naloga: priprava produkcijske različice spletne aplikacije (izdelava lastne aplikacijske slike)

Po končanem razvoju spletne strani želimo pripraviti produkcijo različico Docker projekta (torej sliko, ki vsebuje kodo spletne strani in strežnik za njeno izvajanje). Priprava produkcijske različice lahko vključuje kopiranje programske kode v Docker zabojnik (sedaj je koda namreč v mapi na našem računalniku ta zgolj »povezana« v Docker zabojnik), spremembe nekaterih PHP nastavitev (npr.: onemogočanje izpisa napak), namestitve dodatne programske opreme v Docker zabojnik ter varnostno preverjanje uporabljenih knjižnjic in sistemskih slik.

Za namen vaje se osredotočimo na **najpreprostejši primer zgolj s kopiranjem izvorne kode**, drugih nastavitev za produkcijsko rabo strežnika ni potrebno spreminjati. Uporabite lahko isto sistemsko sliko kot za razvoj:
https://hub.docker.com/_/php

Recept za pripravo novih zabojnikov podajamo v datotekah **Dockerfile**. Oglejte si povezavo z navodili za kontejnerizacijo aplikacij (https://docs.docker.com/get-started/02_our_app/) ter v nekaj nekaj vrsticah zapišite Dockerfile za vaš projekt.

_____
_____
_____
_____

Sedaj lahko izdelate Docker image. Pomagajte si z naslednjim ukazom:

```
docker build -t moja-aplikacija .
```

Preglejte izgrajene slike in zaženite nov kontejner z vašo aplikacijo:

```
docker image ls
docker run -d \
    --name moj-kontejner \
    --publish 8080:80 \
    moja-aplikacija
```

Če ste prijavljeni v Docker Hub lahko po želji svojo aplikacijo javno objavite po sledečih navodilih:
https://docs.docker.com/get-started/docker-concepts/building-images/build-tag-and-publish-an-image/

Kateri korak v vaši Dockerfile datoteki je aplikacijski sliki dodal nov sloj?

_____

2.4 Naloga: uporaba Docker Compose

Orodje Docker Compose omogoča definiranje pravil za zaganjanje več Docker kontejnerjev hkrati. Na ta način se izognemo poganjanju z ročnim dodajanjem atributov ob ustvarjanju kontejnejra. Hkrati lahko nastavitve kontejnerjev spreminjamo na enem mestu, brez potrebe po ročni odstranitvi prejšnjih različic kontejnerjev.

Orodje Docker Compose deluje na osnovi **docker-compose.yml** datoteke. Prikazana je vzorčna datoteka za naš razvojni strežnik:

```
version: '3.9'

services:
  streznik:
    image: php:apache
    container_name: streznik
    ports:
      - "8080:80"
    volumes:
      - /pot/do/namizja/mapa:/var/www/html
```

Nov strežnik lahko sedaj poženete ali ustavite z ukazom

```
docker compose up (-d)
docker compose down
```

Kako vam Docker Compose olajša spremembe konfiguracij kontejnerja? Spremenite vrata na 8090.

_____

Nazadnje v isto docker-compose.yml datodeko dodajte predpis za še en enak kontejner, ki bo stregel rahlo spremenjeno vsebino na portu 9000. Kakšna bo vaša nova docker-compose.yml datoteka?

_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____