Softver otvorenog koda

Žarko Zečević
Elektrotehnički fakultet
Univerzitet Crne Gore

Predavanje 6

Uvod u kontejnere

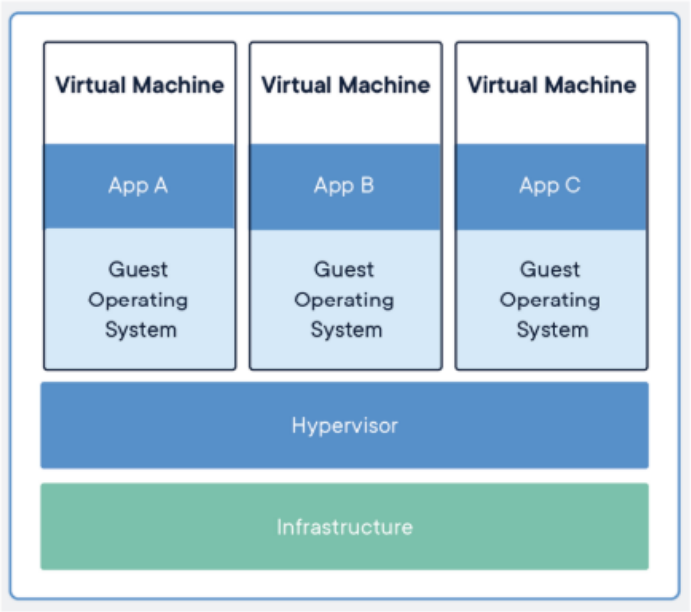
Ishodi učenja:

Nakon savladavanja gradiva sa ovog predavanja studenti će moći da:

- Definišu osnovne pojmove vezane za računarstvo u oblaku i virtuelizaciju
- Naprave razliku između različitih tipova implementacije i servisnih modela Cloud-a
- Nabroje najznačajnija open-source rješenja iz domena Cloud-a

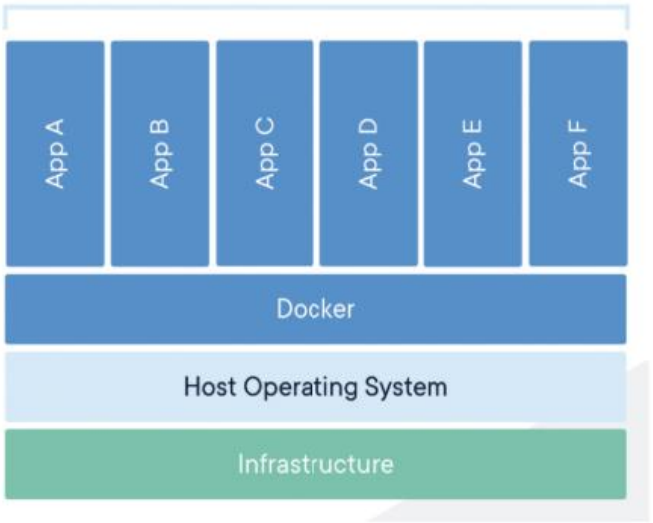
Virtuelne mašine

- Izoluju aplikacije i vrše alokaciju resursa potrebnih za pokretanje aplikacije
- Virtuelne mašine su prenosive (portabilne) – mogu se upakovati u slike i šerovati
- Ne zavise od operativnog sistema hosta
- Na istom hostu možemo pokrenuti više virtuelnih mašina koristeći hipervizor



Kontejneri

- Paketi softverskog koda, zavisnosti i konfiguracionih fajlova
- „Lakši“ od virtuelnih mašina (zauzimaju manje memorije, dijele resurse OS-a, bolje performanse)
- Portabilni (mogu se šerovati kao Docker slike)
- Nepromjenljive
- Ne zavise od OS-a i moguće je pokrenuti više kontejnera istovremeno

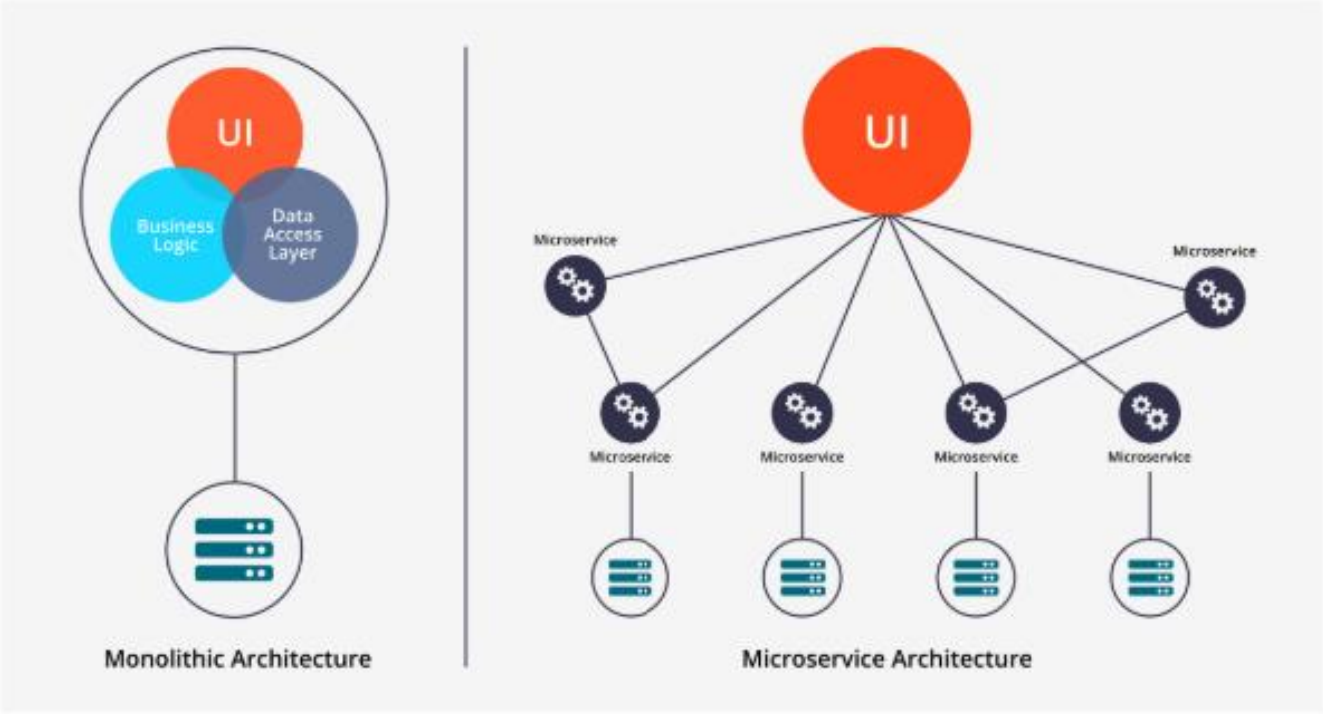


Kontejneri vs virtuelne mašine

What's Diff?	VMs	Containers
size	Heavyweight (GB)	Lightweight(MB)
BootTime	Startup time in minutes	Startup time in seconds
Performance	Limited performance	Native performance
OS	Each VM runs in its own OS	All containers share the host OS
Runs on	Hardware-level virtualization(Type1)	OS virtualization
Memory	Allocates required memory	Requires less memory space
Isolation	Fully isolated and hence more secure	Process-level isolation, possibly less secure

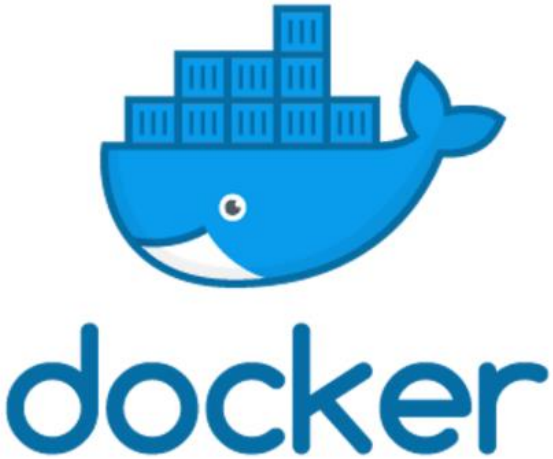
Mikroservisna arhitetktura

- Velika aplikacija se razbija na više manjih komponenti
- Održavanje i testiranje nekad može biti zahtjevnije
- Fleksibilnost – jednostavniji prelazak na nove tehnologije
- Horizontalno skaliranje (resursi svake komponente se mogu nezavisno povećavati)



Zašto koristimo kontejnere?

- Razvoj aplikacije koje će raditi na bilo kojem OS-u
- Jednostavno šerovanje kontejnera sa članovima tima
- Jednostavno skaliranje na više servera
- Velike aplikacije se razbijaju na više kontejnera (npr. jedan za svaki mikroservis)
- Odljično rješenje za Cloud computing (serverless – ne zavisimo više od infrastrukture, različiti mikroservisi na različitim cloud provajderima)
- Velika zajednica i biblioteka docker slika



Docker kontejneri

Tri osnovne komponente:

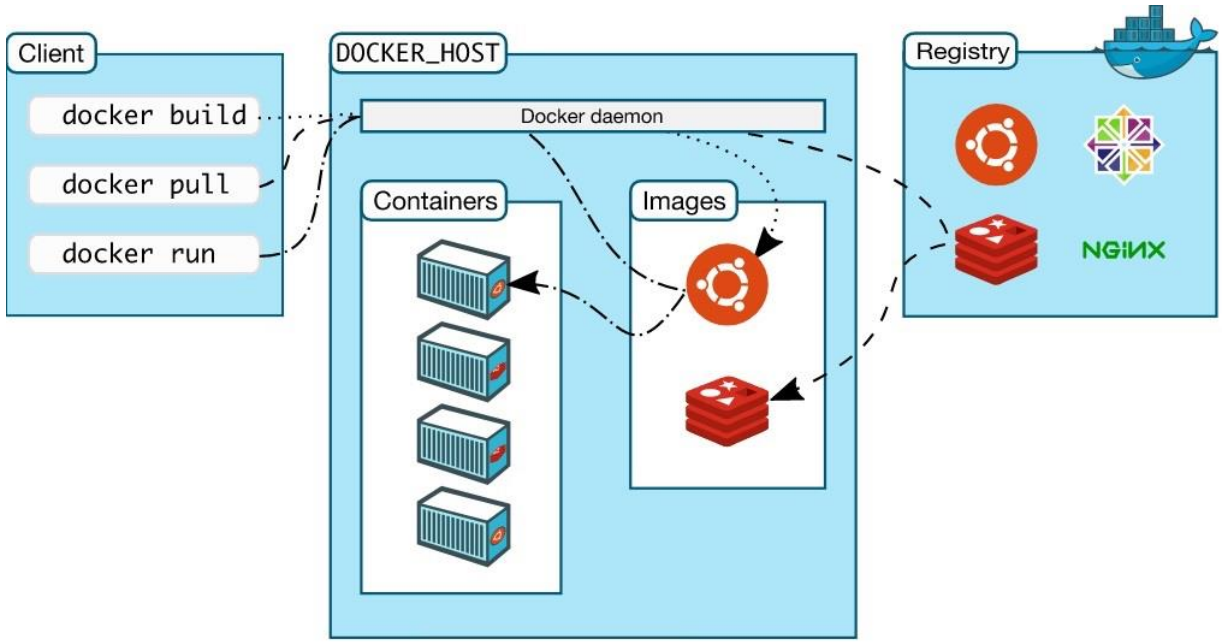
- Demon
- Klijent
- Registar slika

Slike kontejnera

- Dockerfile
- Slika kontejnera

Kontejner

- Instanca slike
- Volumen
- Port
- Environment varijable



Docker kontejneri

Tri osnovne komponente:

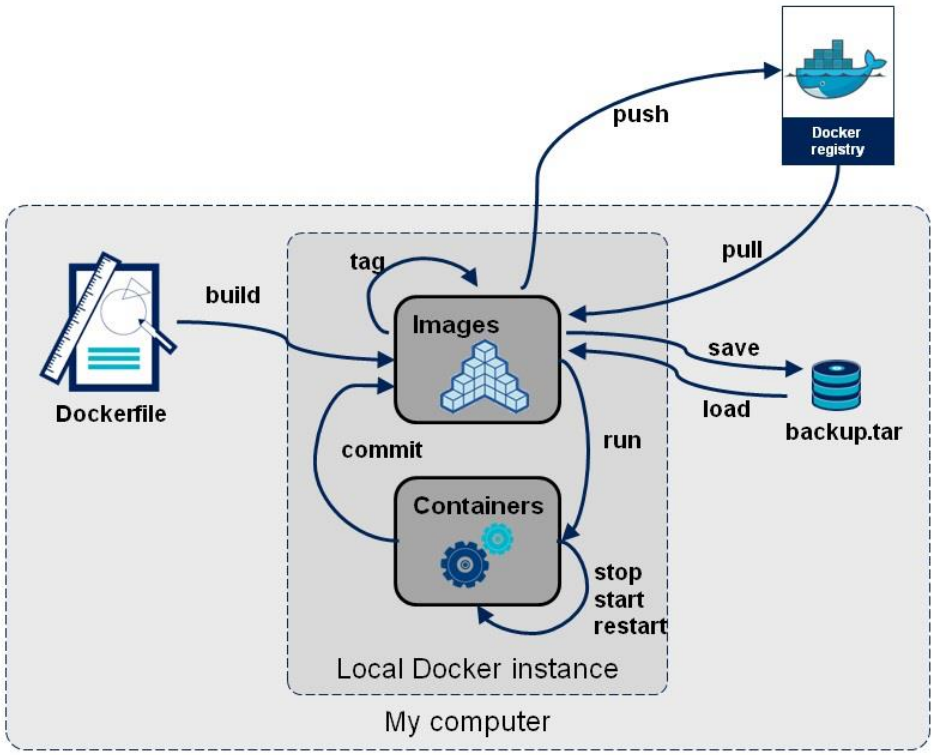
- Demon
- Klijent
- Registar slika

Slike kontejnera

- Dockerfile
- Slika kontejnera

Kontejner

- Instanca slike
- Volumen
- Port
- Environment varijable



Docker slika zapravo predstavlja snapshot fajl sistema

Docker kontejneri

Tri osnovne komponente:

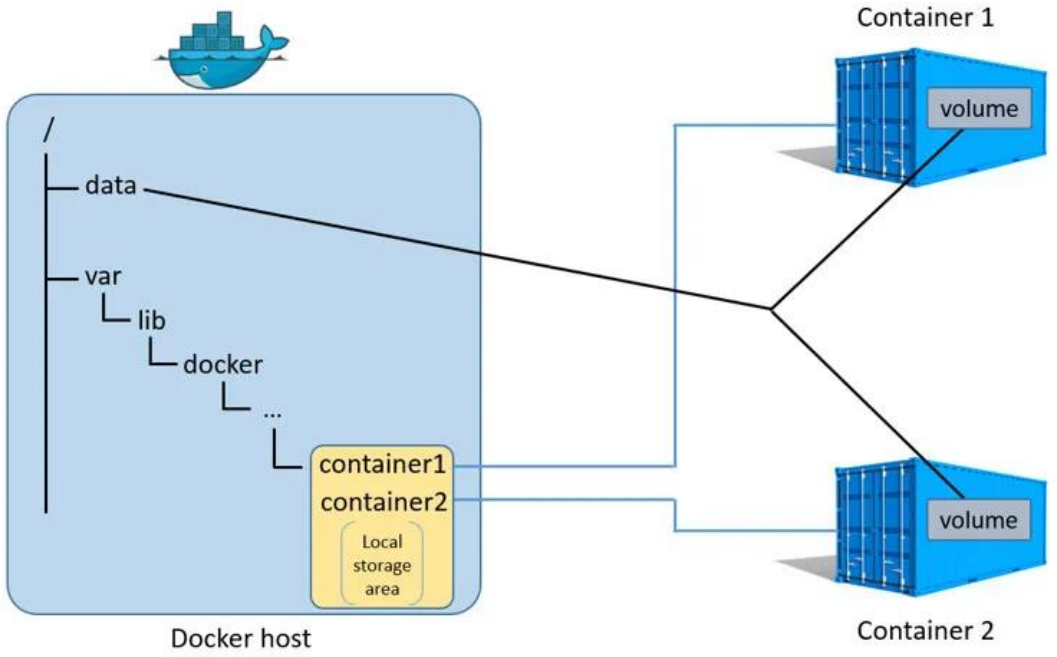
- Demon
- Klijent
- Registar slika

Slike kontejnera

- Dockerfile
- Slika kontejnera

Kontejner

- Instanca slike
- Volumen
- Port
- Environment varijable



- Privremeni fajl sistem (novi sloj povrh slike)
- Grupa procesa (glavni proces se pokreće prilikom pokretanja kontejnera. Kada se ovaj proces stopira, ubijaju se i svi ostali podprocesi u kontejneru)

Rad sa kontejnerima

Komanda	**Opis**
`docker create image [ command ]` `docker run image [ command ]`	Kreiranje kontejnera = kreiranje+startovanje
`docker rename container new name` `docker update container`	promjena imena ažuranje konfiguracije kontejnera

`docker start container...`
`docker stop container...`
`docker kill container...`
`docker restart container...`
	startovanje kontejnera
	stopiranje
	forse stop
	= stop + start

`docker pause container...` `docker unpause container...`	pauziranje kontejnera pokretaanje pauziranog kont.
`docker rm [ -f ] container...`	brisanje kontejnera (-f se dodaje ako kontejner prethodno nije prethodno stopiran)

Kreiranje i pokretanje kontejnera

- Kontejner se kreira pomoću komande *docker create*, pri čemu se kao argument zadaje naziv slike kontejnera. Nakon kreiranja, kontejner dobija ID i može se startovati pomoću *docker start* komande.
- Komanda *docker run* se koristi za kreiranje i pokretanje kontejnera (zamjenjuje *docker create*+*docker start*).

```
>> docker create hello-world
2ffd5f2c5a7562fbf1d7b89a14c11a52e5843dd7938f380a8cd53f3952da99de
>> docker container start 2ffd5f2c5a7562fbf1d7…
2ffd5f2c5a7562fbf1d7…
>> docker run hello-world
Hello from Docker!
This message shows that your installation appears to be working correctly.
# Na ovaj način smo kreirali dva različita kontejnera, oba zasnovana na slici hello-world
```

Kreiranje i pokretanje kontejnera

Prilikom pokretanja kontejnera, nakon naziva slike moguće je unijeti komandu koja će se izvršiti unutar kontejnera. Kontejner može da se pokrene u „foreground“ i „detached“ modu. Foreground mod je podrazumijevani mod, dok se opcija $-d$ koristi za „detached“ mod. U foreground modu, *stdout* i *stderr* iz kontejnera se preusmjeravaju u terminal!

```
>> docker run ubuntu whoami
root
# ova komanda kreira kontejner pomoću slike ubuntu i pokreće komandu whoami. Komanda whoami vraća rezultat root i taj rezultat vidimo na ekranu jer je kontejner pokrenut u interaktivnom modu
>> docker run -d ubuntu whoami
09d06df6f1b1a047d01a899a36e3256efa3af4048ea7e1c1f854aa67f2762
# sada ne vidimo rezultat izvršenja komande whoami, već kao rezultat dobijamo ID kontejnera, jer je kontejner pokrenut u detached modu
```

Kreiranje i pokretanje kontejnera

Komanda *run* ima i brojne druge opcije. Na primjer $-i$ se koristi za pokretanje kontejnera u interaktivnom modu (reaguje na *stdin* terminala), dok opciju *-t* koristimo za alokaciju pseudo terminala za kontejner. Ove dvije opcije se obično koriste zajedno! Provjerite šta u čemu je razlika ako se koristi samo opcija *-i*, i obrnuto.

```
>> docker run -it ubuntu bash
root@c2fbfaa28e7f:/#
# Pokrenuli smo kontejner u interaktivnom modu i otvorili njegov terminal (komanda bash)
root@c2fbfaa28e7f:/# pwd
/
# sada izvršavamo komande unutar kontejnera. Možemo uočiti da se nalazimo u root direktorijumu, a u prethodnom primjeru smo vidjeli da smo mi root korisnik (administrator)
root@c2fbfaa28e7f:/# exit
# zatvaranje terminala
```

Kreiranje i pokretanje kontejnera

Pored ID-a, kontejner prilikom kreiranja dobija i slučajno generisano ime. Ukoliko želimo da dodijelimo ime kontejneru koristimo opciju „*--name*“ (ovo nam olakšava pamćenje). Nakon izvršenja komandi, kontejner se automatski stopira (ukoliko se unutar njega ne izvršava neki pozadinski proces) i ostaje u memoriji! Kontejner se opet može pokrenuti pomoću komande *docker start*. Ukoliko želimo da se kontejner automatski obriše nakon izvršenja komandi, koristi se opcija *--rm*.

```
>> docker run --name proba -d ubuntu bash
2ffd5f2c5a7562fbf1d7b89a14c11a52e5843dd7938f380a8cd53f3952da99
# kreiramo i pokrećemo kontejner pod imenom proba
>> docker run --rm --name proba1 -d ubuntu bash
f35ebdf020839091bdb5256797179ea3195a10cefc30ed3565ba2119a4c561
# ime kontejnera mora biti jedinstveno! Kontejner proba1 će se
automatski obrisati u ovoj varijanti komande. Probajte da opet
kreirate kontejner pod istim imenom.
```

Brisanje kontejnera

U prethodnim primjeru smo kreirali veliki broj kontejnera. Svi oni nakon izvršenja i stopiranja ostaju u memoriji, sem ako nijesmo koristili opciju --rm. Kontejneri se brišu pomoću komande *docker rm* koja za argument ima ID ili naziv kontejnera. Spisak svih kontejnera (aktivnih i stopiranih) možemo dobiti pomoću komande *docker ps -a*. Sve stopirane kontejnere možemo obrisati sa *docker container prune!*

```
>> docker rm proba
# brišemo kontejner proba
>> docker ps –a
Container ID   Image         Name    Created      Status
e1dc694b9d59   hello-world   "/hello 4 weeks ago Exited (0) 4 weeks ago
                           fervent_Clarke
>> docker rm ed1
# dovoljno je navesti nekoliko prvih karaktera ID-a ili naziv
kontejnera
>> docker container prune  # docker system prune --all
# brišemo sve kontejnere sa statusom Exited (0)
```

Pregled informacija o kontejneru

Komanda	**Opis**
`docker ps` `docker ps -a`	Prikazivanje aktivnih kont. Prikazivanje svih kontejnera
`docker logs`	Prikazivanje izlaza kontejnera (korisno za utvrđivanje greške)
`docker top container` `docker stats [ container ]`	Prikaz informacija o aktivnim procesim u kontejneru (slično komandama ps i top u Linuxu)
`docker diff container`	Prikazuje razlike kontejnera u odnosu na originalnu sliku
`docker port container`	Spisak mapiranih portova (između host-a i kontejnera)
`docker inspect container ...`	Detaljnije informacije o sadržaju kontejnera

Interakcija sa kontejnerima

Komanda	**Opis**
`docker attach container`	„kačenje“ terminala na stdin, stdout ili stderr kontejnera
`docker cp container:path hostpath` `docker cp hostpath container:path`	Kopiranje fajlova iz kontejnera na host, i obrnuto
`docker export container`	Eksportovanje sadržaja kontejnera u tar arhivu
`docker exec container args ...`	Izvršavanje komande unutar kontejnera
`docker wait container`	Ova komanda čeka i vraća *exit code* kad kontejner završi sa radom
`docker commit container image`	Čuvanje snapshot-a kontejnera u novu **docker sliku**

Interakcija sa kontejnerima

```
>> docker run -it --name novi ubuntu
root@aab57c7718d7:/# echo "proba" > /etc/novifajl
# kreiramo kontejner novi i fajl unutar njega
>>  docker diff novi
C /root
A /root/.bash_history
C /etc
A /etc/novifajl
# upoređujemo kontejner sa originalnom slikom
>> docker commit novi nova_slika
sha256:a2ba89a07750fb442693f1fe...
# kreiramo novi sliku na osnovu kontejnera
>> docker rm –f novi
>> docker run -it --name novi nova_slika
root@a8abbfdf5af9:/# ls /etc
# kreiramo novi kontejner koristeći sliku nova_slika. Ukoliko
# izvršimo ls komandu unutar kontejnera, vidjećemo da u
# folderu /etc postoji fajl novifajl
```

Rad sa volumenima

U toku rada sa kontejnerima, često će se ukazati potreba da neke podatke iz kontejnera treba da sačuvamo na host ili pak da obezbijedimo kontejneru pristup konfiguracionim fajlovima sa host mašine. Ovo možemo postići pomoću komande

```
>> docker run -v /hostpath:/ctrpath[:ro] ...
```

gdje */hostpath* predstavlja putanju do foldera (**ili fajla**) na hostu, a */ctrpath* putanju (automatski se kreira) u kontejneru na koju ćemo da montiramo direktorijum hosta. Opcija :ro nije obavezna i koristi se kada želimo da dozvole nad fajlovima budu read-only.

Nažalost, jedini način za montiranje foldera je prilikom pokretanja kontejnera!

```
>> docker run -it -v ~/:/mojhome ubuntu
# monitoramo naš home direktorijuma na putanju /mojhome
>> ls /mojhome
# provjera sadržaj folder /mojhome
```

Rad sa volumenima

Pored toga što možemo da montiramo eksterne volumene, moguće kreirati i koristiti takozvane imenovane volumene. Imenovani volumeni se kreiraju pomoću komande

```
>> docker volume create my-volume
```

Imenovani volumeni se čuvaju u folderu */var/lib/docker* (ovdje se čuvaju i ostali podaci o kontejnerima i slikama!), njima se upravlja pomoću komande *docker volume*, a pored toga putem API-a je moguće pristupiti njihovom sadržaju (korisno za cloud aplikacije).

```
>> docker volume create disk
# kreiranje volumena pod nazivom disk
>> sudo ls /var/lib/docker/volumes/disk/_data
# ovo je putanja do volumena na host-u
>> docker run -it -v disk:/mojdiks ubuntu
# monitamo diska na putanju /mojdisk
>> echo >> /mojdisk/prvifajl
```

Konfigurisanje mreže

Kontejneri ne mogu da imaju statičke IP adrese. Prilikom kreiranja, kontejner se podrazumijevano pridružuje mreži pod nazivom ***bridge*** i dobija IP adresu iz mreže 172.17.0.0/16. Jedino kontejneri koji se nalaze unutar iste mreže mogu međusobno da komuniciraju.

Nova mreža se kreira pomoću komande

```
>> docker network create NETWORK
```

Kontejneri se pridružuju mreži na sljedeći način (prilikom kreiranja):

```
>> docker run --net=NETWORK ...
```

Kontejneri mogu i dinamičkim putem da se pridruže/uklone iz mreže

```
>> docker network connect NETWORK CONTAINER
>> docker network disconnect NETWORK CONTAINER
```

Komunikacije je moguća samo između kontejnera u istoj mreži, ali jedna kontejner se može pridružiti većem broj mreža.

Korisna mogućnost je da kontejneri u istoj mreži mogu da prepoznaju jedni druge i preko imena kontejnera (ovo ne važi za bridge mrežu).

Konfigurisanje mreže

S obzirom da kontejneri imaju privatne adrese, ne možemo im direktno pristupiti putem interneta. Međutim, zahtjeve možemo uputiti hostu, a on će ih proslijediti kontejneru:

```
>> docker run -p [ipaddr:]hostport:containerport ...
```

Gornja komanda obezbjeđuje da se svi zahtjevi upućeni hostu preko IP adrese *ippaddr:hostport* preusmjere u kontejner na port *containerport.* Ukoliko se ne navede IP adresa, onda se će host prosljeđivati zahtjeve sa svih interfejsa (ukoliko ima više IP adresa).

```
>> docker run -d --name host1 -p 900:80  httpd
>> docker run -d --name host2 -p 901:80  httpd
# kreiranje dva kontejnera (koristi se ubuntu+apache slika)
>> wget localhost:900
# iz terminala hosta šaljemo zahtjev web serveru prvog kont.
>> docker inspect host2
# pronalazimo IP adresu hosta 2
>> docker exec host1 wget 172.17.0.6:80
# iz terminala host 1 saljemo zahtjev web serveru hosta 2
```

Rad sa docker slikama

Docker slike služe za kreiranje kontejnera. U suštini one predstavljaju snapshot fajl sistema (+metapodaci). Docker slike su nepromjenljive, imaju svoj ID, mogu imati tag i mogu se koristiti za kreiranje novih slika. Pomoću komande *docker pull naziv_slike:tag* docker slika se preuzima sa docker hub-a (repozitorijum slika) i čuva na sistemu. Ukoliko se tag ne navede, preuzima se zadnja verzija slike. Prilikom pokretanja komande *docker run*, ukoliko slika ne postoji na sistemu, slika će se automatski preuzeti sa docker hub-a (ukoliko postoji na njemu). Komanda *docker images* prikazuje spisak slika koje se nalaze na hostu.

```
>> docker pull httpd
# ova slika sadrži linuxu+apache2 server
>> docker images
REPOSITORY    TAG         IMAGE ID         CREATED          SIZE
httpd         latest      359570977af2     7 weeks ago      168MB
hello-world   latest      9c7a54a9a43c     6 months ago     13.3kB
```

Rad sa docker slikama

Komanda	**Opis**
`docker images`	Spisak slika
`docker history image`	Prikaz informacija o istoriji slike (njenim prethodnim verzijama)
`docker inspect image ...`	Detaljnije informacije o sadržaju slike
`docker export container`	Eksportovanje sadržaja kontejnera u tar arhivu
`docker tag image tag`	Tagovanje docker slike
`docker commit container image`	Kreiranje nove slike (snaphsot kontejnera)
`docker import url|- [tag]`	Kreiranje slike importovanjem tar arhive
`docker rmi image ...`	Brisanje slike sa sistema

Kreiranje docker slike

Da bi kreirali svoju docker sliku potrebno je da kreiramo docker fajl pod nazivom *Dockerfile*. U docker fajlu definišemo kako slika treba da izgleda. Docker fajl treba sačuvati u nekom folderu, pri čemu taj folder može da ima i dodatni sadržaj.

Slika se kreira/izgrađuje pomoću komande

```
>> docker build [ -t tag ] putanja do foldera
```

Prethodna komanda pravi tarball arhivu od sadržaja foldera i arhivu prosljeđuje docker deamon-u. Fajlovi koji se nalaze u .dockeringore se ignorišu. Docker deamon na osnovu docker fajla kreira sliku. Slika se sastoji iz više slojeva – nakon svake run komande definisane u docker fajlu vrši se commit (pravi novi snapshot slike).

Opciono, sliku možemo imenovati i tagovati.

```
>> docker build -t proba:2.0 ~/doker
# kreira se nova slika pod nazivom proba i tagom 2.0
```

Sadržaj docker fajla

```
FROM ubuntu:latest
# specificiramo naziv osnovne slike na koju ćemo da dodamo
# dodatne slojeve. Primjeri: alpine:latest, debian, itd.
RUN apt-get update && apt-get -y dist-upgrade
# ažuriranje softvera
RUN apt-get -y install nginx
# instalacija nginx web servera
CMD ["nginx", "-g", "daemon off;"]
# podešavanje defaultne komande koja će se izvršavati prilikom
# pokretanja kontejnera. Ovaj proces će imati PID 1.
# Konkretno, podešavamo da se nginx startuje u prvom planu
# prilikom (foreground), a ne u pozadini
EXPOSE 80
# obavještavamo docker mašinu da će postojati proces koji
osluškuje na portu 80
```

Instrukcije unutar docker fajla

Instrukcija	**Opis**
`FROM image`	Odabir osnovne slike
`COPY path dst`	Kopiranje sadržaja folder path (na hostu) u folder dst (u kontejneru)
`ADD src dst`	Slično kao copy, ali podržava URL kao prvi argument
`RUN command`	Izvršavanje komandi tokom kreiranja slike
`CMD command` `ENTRYPOINT command`	Komanda koja se izvršava unutar kontejnera Najčešće dopunjuje komandu CMD na način što se u CMD unose argumenti komande definisane u ENTRYPOINT. Podrazumijevana komanda je /bin/sh –c.

Instrukcije unutar docker fajla

Instrukcija	**Opis**
`USER name[:group]`	Username korisnika koji pokreće kontejner (podrazumijevano root)
`WORKDIR path`	Radni direktorijum (podrazumijevano /)
`ENV name="value"`	Environment varijable
`EXPOSE port. . .`	TCP/UDP portovi koje treba otvoriti
`VOLUME path. .`	Putanja do eksternog drajva/diska
`ARG name[=value]`	Varijable koje se koriste u procesu instalacije slike
`LABEL name="value"…`	Proizvoljni metapodaci
`ON BUILD instruction`	Komanda koja se izvršava prilikom kreiranje slike

Docker ekosistem

- Na kraju ćemo pomenuti da postoji još jedan način za kreiranje slika – korišćenjem docker compose komande. Compose fajl ima YAML format i u njemu je moguće konfigurisati više kontejnera koji se kasnije istovremeno pokreću pomoću *compose up* komande.
- Kontejnere je moguće pokrenuti na klasteru (grupi od više hostova). Međutim, za tu svrhu su nam potrebni *alati za orkestraciju* koji će da vode računa o automatskom deploymentu i skaliranju kontejnera.
- Postoje brojna rješenja za orkestraciju kontejnera na klasteru
  - Docker swarm
  - Apache Mesos
  - Kubernetes