# Vežba 1:

* Instalirati Docker i potrebne zavisnosti
  + sudo apt-get update
  + sudo apt-get install curl ca-certificates software-properties-common
  + sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg
  + sudo echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/ubuntu $(lsb\_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
  + sudo apt update
  + sudo apt install docker-ce
  + sudo systemctl status docker
* Izlistati image-e na trenutnom sistemu
* Dovući Ubuntu 18.04 image i 20.04 image-e i izlistati trenutne image-e
* Obrisati Ubuntu 18.04 image
* Pokrenuti 20.04 kontejner i ispisati Hello World (*docker run ubuntu:20.04 /bin/echo ‘Hello World’*)
* Izlistati kontejnere pomoću komande *ps* sa i bez *–a* parametra
* Zašto je kontejner stopiran?
* Pokrenuti 20.04 kontejner ponovo ali sada u interaktivnom okruženju
* Kada je kontejner pokrenut u interaktivnom okruženju, u njemu se mogu izvršavati različite komande (kao da ste na hostu)
* Izaći iz konejnera i proveriti da li je stopiran
* Obrisati kontejnere i proveriti da li su obrisani

# Vežba 2:

* Pokrenuti ponovo kontejner u interaktivnom okruženju i dati mu ime *my\_container* (koristeći parametar *--name*). Ako želite da vidite koje sve parametre određena Docker komanda može da ima, možete je uvek pokrenuti sa parametrom *--help*.
* Izvršiti komandu *ping* u terminalu
* S obzirom da image koji smo povukli sa DockerHub-a ima samo minimalan softver, ova komanda nije prepoznata
* Ažurirati sistem pomoću *apt-get update* komande
* Instalirati komandu ping pomoću komande *apt-get install iputils-ping* i pokrenuti *ping* ponovo
* Posle uspešnog izvršavanja ove komande, izaći iz kontejnera
* Pošto ne želimo da svaki put radimo ovo, treba da sačuvamo izmene na postojećem image-u
* To je moguće uraditi korišćenjem komande *commit* (možete dodati *--help* da biste izlistali sve opcije te komande).
* Izlistati ponovo image-e (posle naredbe *commit*, napravljen je novi image od baznog)
* Pokrenuti ponovo kontejner sa novim image-om (potrebno mu je dati i različito ime od prethodno pokrenutog kontejnera) i potvrditi da *ping* komanda i dalje radi
* Izaći iz kontejnera i obrisati ga

# Vežba 3:

* U prethodnoj vežbi smo dodali novi sloj na bazni image, a u ovoj ćemo napraviti sopstveni image
* Obrisati image kreiran u prethodnoj vežbi
* Napraviti fajl sa imenom Dockerfile i otvorite ga u editoru
* Dodajte komande da označite koji ćete bazni image koristiti (20.04 koji smo dovukli sa DockerHub-a) i dodajte labelu (*LABEL*) sa imenom autora:
  + *FROM ubuntu:20.04*
  + *LABEL author=‘Petar Petrovic’*
* Dodajte dve *RUN* komande, prva koja radi ažuriranje sistema i druga koja instalira ping komandu (dodati opciju –y posle druge komande)
* Pomoću komande *build*, napraviti image koristeći naredbu u upravo kreiranom Dockerfile-u. Pri pokretanju naredbe, morate postaviti putanju do foldera gde se traži Dockerfile (u našem slučaju će to biti tekući direktorijum). Možete nazvati image koji kreirate (tag parametar) *my\_image*
* Dok se izvršava komanda *build*, obratiti pažnju da svaka komanda pravi novi commit
* Taj deo je važan jer ubrzava ponovno kreiranje image-a, jer se promene koje nisu pogođene novim izmenama samo iskorišćavaju
* Izlistati image-e i potvrditi da je image koji smo upravo kreirali prisutan u listi

# Vežba 4:

* Kao što vidite, veličina image-a se značajno povećala iako nismo ništa značajno instalirali
* Razlog za to je *apt-get* naredba koja instalira i dosta nepotrebnih paketa u našem slučaju. To uključuje sve promene u fajl sistemu kao što su logovi, privremeni podaci i sl.
* Možemo probati da dodamo *apt-get clean* komandu na kraj Dockerfile-a:
  + *RUN apt-get clean && cd /var/lib/apt/lists && rm –rf \*Release\* \*Packages\* && truncate –s 0 /var/log/\*log*
* Pokrenuti ponovo *build* komandu i proveriti veličinu fajla nakon toga
* Vidimo da je veličina i dalje ista. Razlog za to je što se sve prethodne promene koje su već napravljene jednom *RUN* komandom neće biti pregažene novom. Prisetimo se da Docker radi tako što pravi image-e dodavajući nove slojeve na postojeći image, tako da će izmena koja je napravljenja prvom *RUN* komandom biti permanentna.
* Spojiti 3 *RUN* komande u jednu tako da se doda čišćenje nepotrebnih paketa i logova nakon instalacije i ažuriranja sistema
* Izlistati ponovo image-e i potvrditi da je veličina fajla smanjena

# Vežba 5:

* Ako bismo želeli da sačuvamo stanje između kontejnera, potrebno je da koristimo Docker volume
* Prvo ćemo pokrenuti Apache server sledećom komandom:
  + *docker run --name apache -p 80:80 httpd:2.4*
  + Ova naredba startuje kontejner baziran na HTTP 2.4, daje mu ime apache i kači se na port 80. Takođe, dodata je i opcija da se kontejner obriše kad se stopira
* Sada možete pokrenuti komandu (iz drugog taba/terminala) *curl localhost* i dobićete podrazumevanu veb stranu servera
* Možemo napraviti sopstveni *index.html* fajl drugačijeg sadržaja i kopirati ga sa hosta u kontejner: *docker cp index.html apache:/usr/local/apache2/htdocs/*
* Komanda *cp* je slična komandi *scp* i kopira fajlove sa hosta u kontejner
* Kao argument smo dali ime kontejnera u koji hoćemo da kopiramo (apache) i putanju odakle se učitava odgovarajuća veb stranica
* Ako ponovo pokrenemo komandu *curl localhost* videćemo da je sadržaj izmenjen
* Ako stopiramo kontejner i pokrenemo ga ponovo videćemo da naše izmene nisu sačuvane. Ovo je poznat problem gde se ne možemo osloniti da će naše izmene zauvek biti tu, jer se u praksi može desiti da se kontejneri ugase iz bilo kog razloga i naše promene će biti izgubljene.
* Obrisati kontejnere

# Vežba 6:

* Prvo ćemo proći kroz osnovne komande vezane za volume:
  + *docker volume ls* – Izlistamo volume-e i vidimo da ih trenutno nema
  + *docker volume create myvolume* – Kreiramo volume sa datim imenom. Izlistamo volume-e ponovo i vidimo da je upravo kreirani volume prisutan u listi.
  + *docker volume rm myvolume* – Brišemo naš volume.
* Kreirati volume sa imenom *httpd\_htdocs*
* Ponovo pokrenuti kontejner na sledeći način:
  + *docker run --name apache -p 80:80 -v httpd\_htdocs:/usr/local/apache2/htdocs/ httpd:2.4*
  + Ovde smo dodali mount parametar –v, obratite pažnju na komande s početnih slajdova da biste razumeli šta tačno ova komanda radi
* Kopirati ponovo naš *index.html* fajl i pokrenuti *curl localhost* posle toga da potvrdimo da smo uspešno kopirali
* Stopirati apache i pokrenuti kontejner ponovo sa istom komandom (potrebno je dati mu drugo ime)
* Ako sada pokrenemo *curl localhost*, videćemo da su naše promene sačuvane
* Sada kada imamo ovaj volume kreiran, možemo ga postaviti na bilo koji HTTPD kontejner, i na taj način možemo menjati verzije kontejnera bez gubitka podataka
* Stopirati apache, obrisati kontejnere i obrisati volume
* Kao alternativu, takođe možemo koristiti i direktorijum sa host mašine kao volume
* Ovo je jako korisno kada radimo neki razvoj, jer možemo lokalno menjati direktorijume i menjati sadržaj Docker kontejnera bez potrebe da pokrećemo build komandu konstantno.
* Pokrenuti ponovo run komandu sa opcijom *–v*, ali ovaj put postaviti lokalni direktorijum gde smo prethodno kreirali *index.html* fajl kao volume (umesto *httpd\_htdocs*)
* Ako sada pokrenemo *curl localhost*, videćemo sadržaj *index.html* fajla iz našeg lokalnog direktorijuma
* Izmeniti sadržaj *index.html* fajla i ponovo pokrenuti *curl*. Videćemo da su naše promene odmah vidljive bez potrebe za kopiranjem.
* Stopirati apache i obrisati kontejner

# Vežba 7:

* U ovoj vežbi ćemo se upoznati sa umrežavanjem dva kontejnera i uspostavljanjem komunikacije među njima
* Kada pokrenete *docker network ls*, možete videti da postoje 3 mreže
* Svi novi kontejneri se podrazumevano dodaju u *bridge* mrežu. Ovo zapravo predstavlja vezu sa ethernet-om na hostu i omogućava Docker-u da pristupi internetu.
* Pokretanjem komande *docker network inspect bridge* možemo videti informacije o toj mreži
* Vidimo da je sadržaj polja *Containers* prazan jer nije dodat nijedan kontejner
* Možemo pokrenuti kontejner iz vežbe 4sledećom komandom:
  + *docker run -it --name dummy my\_image*
  + Pokrenućemo i *inspect* komandu ponovo (iz drugog taba/terminala) da bismo pronašli IP adresu kontejnera
* Možemo pokrenuti još jedan kontejner (iz drugog taba/terminala) i postaviti da uradi ping do prvog kontejnera. Kao što vidite, ping-ovanje kontejnera je uspešno. Jedina mana je što smo ovde koristili IP adresu koja se može promeniti. Zbog toga umesto IP adrese možemo zadati ime prvog kontejnera. Ako pokušamo da uradimo to, desiće se greška jer nam podrazumevana *bridge* mreža to neće dozvoliti. Zato ćemo u sledećoj vežbi napraviti sopstvenu mrežu.
* Stopirati i ukloniti *dummy* kontejnere.

# Vežba 8:

* Napraviti mrežu sledećom komandom:
  + *docker network create skynet*
* Izlistati postojeće mreže i uraditi inspect upravo kreirane mreže
* Ponovo pokrenuti kontejner kao u prošloj vežbi, ovaj put sa parametrom *--network* koji nam služi da zadamo ime mreže u kojem će se kontejner pokrenuti
* Ponovo pokrenuti novi kontejner kao na prethodnom slajdu. Potrebno je i dalje imati *--network* parametar, jer se taj kontejner ne nalazi u podrazumevanoj bridge mreži.
* Ako sada probamo da uradimo ping do prvog kontejnera koristeći njegovo ime, videćemo da je ping uspešan. Ovo je mogućnost Docker-ovog DNS-a i jako je korisno kada imamo više kontejnera u jednoj aplikaciji kao što su baze podataka, veb server itd. Ne moramo znati njihove IP adrese, već samo zadamo ime i iskoristimo mogućnosti DNS-a.
* Stopirati i obrisati oba kontejnera i potvrditi da su obrisani
* Obrisati *skynet* mrežu i proveriti da je obrisana
* Obrisati sve image-e i potvrditi da su obrisani