Kontejneri u Linux okruženju

Kovačević, Karlo

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Stručni studij

KONTEJNERI U LINUX OKRUŽENJU

Završni rad

Karlo Kovačević

Osijek, 2020.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac Z1S: Obrazac za imenovanje Povjerenstva za završni ispit na preddiplomskom stručnom studiju

Osijek, 18.09.2020.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za završni ispit na preddiplomskom stručnom studiju

Ime i prezime studenta:	Karlo Kovačević
Studij, smjer:	Preddiplomski stručni studij Elektrotehnika, smjer Informatika
Mat. br. studenta, godina upisa:	AI 4615, 24.09.2019.
OIB studenta:	00162234746
Mentor:	Izv. prof. dr. sc. Irena Galić
Sumentor:	Dr. sc. Hrvoje Leventić
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	Doc.dr.sc. Mirko Köhler
Član Povjerenstva 1:	Izv. prof. dr. sc. Irena Galić
Član Povjerenstva 2:	Dr. sc. Krešimir Romić
Naslov završnog rada:	Kontejneri u Linux okruženju
Znanstvena grana rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Zadatak završnog rada	Svrha ovog završnog rada je opisati prednosti, mane i primjenu kontejnerizacije na poslužiteljima koje pogoni Linux operativni sustav. U teorijskom dijelu potrebno je opisati koncepte virtualizacija, što su to Linux kontejneri i po čemu se razlikuju od virtualnog stroja, opisati kontejnerske tehnologije (prvenstveno Docker, ali ukratko i ostale) te opisati koncept kontejnerizacije servisa u svrhu uspostavljanja servisa (eng. deployment). U praktičnom dijelu potrebno je prikazati primjenu Dockera za deployment jednostavne web aplikacije i pratećih servisa (baze podataka, analitičkih servisa i slično). Tehnologija: Linux, Docker, virtualizacija Tema rezervirana za: Karlo Kovačević Sumentor s FERIT-a: Hrvoje Leventić
Prijedlog ocjene pismenog dijela ispita (završnog rada):	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 3 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene mentora:	18.09.2020.

Potpis mentora za predaju konačne verzije rada u Studentsku službu pri završetku studija:	Potpis:
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA
I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O ORIGINALNOSTI RADA

Osijek, 01.10.2020.

Ime i prezime studenta:	Karlo Kovačević
Studij:	Preddiplomski stručni studij Elektrotehnika, smjer Informatika
Mat. br. studenta, godina upisa:	AI 4615, 24.09.2019.
Turnitin podudaranje [%]:	2

Ovom izjavom izjavljujem da je rad pod nazivom**: Kontejneri u Linux okruženju**

izrađen pod vodstvom mentora Izv. prof. dr. sc. Irena Galić

i sumentora Dr. sc. Hrvoje Leventić

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

SADRŽAJ

1. UVOD 1
1.1. Zadatak završnog rada 1
2. VIRTUALIZACIJA 2
2.1. Općenito 2
2.2. Povijesni pregled 3
2.3. Detalji 5
2.4. Virtualni stroj 8
3. KONTEJNERI 12
3.1. Općenito 12
3.2. Docker i ostale tehnologije 15
3.3. Razlike u odnosu na virtualni stroj 19
3.4. Koncept kontejnerizacije u svrhu uspostavljanja servisa 20
4. USPOSTAVLJANJE WEB APLIKACIJE KORISTEĆI DOCKER 22
4.1. Detalji implementacije 22
4.2. Postupak implementacije 22
5. ZAKLJUČAK 34
LITERATURA 35
SAŽETAK 38
ABSTRACT 38
ŽIVOTOPIS 39
PRILOG 1 – SADRŽAJ DOCKERFILE DATOTEKE 40
PRILOG 2 – SADRŽAJ DOCKER-COMPOSE DATOTEKE 41

1. UVOD

U današnje vrijeme rasprostranjenost računala je velika. Sukladno toj tvrdnji može se zaključiti kako je i cijena hardvera za obične korisnike u velikom dijelu svijeta prihvatljiva, prije nekoliko desetljeća situacija je bila puno drugačija. Velike tvrtke morale su pronalaziti načine kako bi postojeće resurse iskoristili u potpunosti, kao rezultat ove težnje javila se potreba za virtualizacijom. Virtualizacija je omogućila puno bolju iskoristivost postojećeg hardvera bez dodatnih troškova. Temeljni cilj virtualizacije bio je napraviti mnoštvo virtualnih kopija postojećih resursa koje bi mogle zasebno funkcionirati i na taj način pridonijeti boljoj iskoristivosti. Sama ideja konceptualno ostala je ista, ali kako je vrijeme odmicalo tako su se i pojavljivali bolji i napredniji koncepti virtualizacije, samim time pojavio se koncept kontejnerizacije koji je pronašao svoju svrhu u određenim situacijama. Princip virtualizacije danas je sveprisutan i može se susresti u bilo kojem dijelu tehničke industrije.

U teorijskom dijelu rada na sažet način pokušava se opisati situacija kojom je došlo do potrebe za virtualizacijom, te se navode određeni detalji i podjele. Također, veći dio rada temelji se na konceptu virtualnih strojeva, kontejnerima u Linux okruženju, njihovim sličnostima i razlikama koje se mogu shvatiti kroz tekst. Praktični dio kroz jedan primjer pokazuje određene teorijski opisane tvrdnje, te kako bi izgledala jedna jednostavna implementacija koristeći konkretnu kontejnerizacijsku tehnologiju Docker.

Drugo poglavlje temelji se na virtualizaciji i virtualnim strojevima. Treće poglavlje temelji se na kontejnerima u Linux okruženju i uključuje jedno potpoglavlje koje opisuje razlike u odnosu na virtualni stroj. Četvrto poglavlje prikazuje postupak i detalje implementacije koristeći tehnologiju Docker.

1.1. Zadatak završnog rada

Svrha ovog završnog rada je opisati prednosti, mane i primjenu kontejnerizacije na poslužiteljima koje pogoni Linux operativni sustav. U teorijskom dijelu potrebno je opisati koncepte virtualizacija, što su to Linux kontejneri i po čemu se razlikuju od virtualnog stroja, opisati kontejnerske tehnologije (prvenstveno Docker, ali ukratko i ostale) te opisati koncept kontejnerizacije servisa u svrhu uspostavljanja servisa (eng. deployment). U praktičnom dijelu potrebno je prikazati primjenu Dockera za uspostavljanje jednostavne web aplikacije i pratećih servisa (baze podataka, analitičkih servisa i slično).

2. VIRTUALIZACIJA

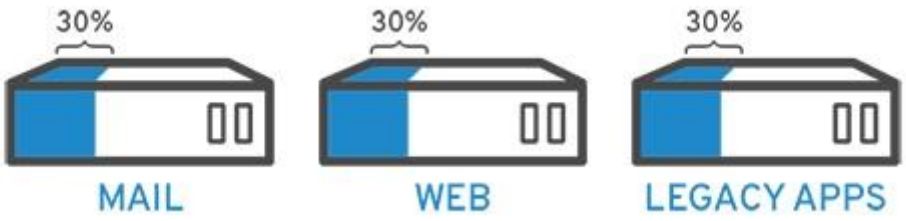
U ovom se poglavlju opisuje pojam virtualizacija, na koji način je došlo do potrebe za njenom implementacijom te na koji se način s tehničke strane omogućava ista. Također, objašnjava se što je to virtualni stroj.

2.1. Općenito

Virtualizacija (engl. *virtualization*) je tehnologija koja daje korisniku mogućnost stvaranja više simuliranih okruženja ili dodijeljenih resursa korisnicima, s jednog hardverskog sustava. Softver pod imenom hipervizor (engl. *hypervisor*) spaja se direktno na hardver i pruža mogućnost dijeljenja sustava u više odvojenih, zasebnih i sigurnih okruženja koja se nazivaju virtualni strojevi (engl. *virtual machines*). Virtualni strojevi oslanjaju se na mogućnost hipervizora kojom on razdvaja resurse sustava (na kojem se vrši virtualizacija) od hardvera i na njihovu pravilnu raspodjelu. Virtualizacija daje mogućnost bolje iskoristivosti postojećih resursa.

Hardver zajedno s hipervizorom, naziva se *host*, dok više virtualnih strojeva koji koriste te resurse nazivaju se *guests.* Virtualni strojevi tretiraju resurse - procesor, memoriju i prostor za pohranu – kao skupinu koja se po potrebi može lagano dodijeliti opet. Različiti operatori kontroliraju dijelove procesora, memorije, prostora za pohranu i ostalih resursa koji će se dodijeliti na način da ih virtualni strojevi dobiju onda kada ih zatrebaju [1].

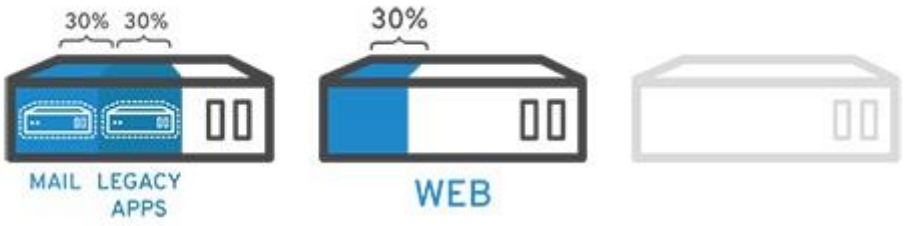
Gledajući s praktične strane, prikazuje se situacija u kojoj postoje tri fizička servera, gdje svaki od njih zasebno obavlja svoju zadaću. Jedan je primjerice server elektroničke pošte (engl. *e-mail*), drugi je *web* server, a treći se bavim određenim aplikacijama. Svaki server koristi 30% svog kapaciteta, dakle samo dio od onoga koliko bi zapravo mogao. Smatra se da server koji se bavi aplikacijama sadrži bitne stvari pa će se on i server koji ga *hosta* pokušati zadržati.



Sl. 2.1. Primjer tri odvojena servera od kojih je svaki djelomično iskorišten. Izvor: [2]

Prema slici 2.1. tako bi izgledao tradicionalan način raspodjele poslova. Bilo je jednostavnije i pouzdanije pokretati svaki zadatak zasebno, primjerice jedan server, jedan operacijski sustav,

jedan zadatak. Teško je bilo jednom serveru dati više „mozgova“. Koristeći virtualizaciju prvi server sa slike podijelio bi se u nova dva koja bi bila u mogućnosti neovisnog obavljanja zadataka. U ovom konkretnom slučaju, treći server koji se bavi aplikacijama migrirao bi se na prvi. Radi se o istom hardveru, samo se efikasnije koristi. To bi u praksi izgledalo kao na slici 2.2.



Sl. 2.2. Primjer servera gdje smo treći oslobodili migrirajući njegov zadatak na prvi server. Izvor: [2]

Naime, postoji i mogućnost dijeljenja prvog servera na još jedan dio u koji bi bio migriran zadatak s drugog servera. Na taj način povećava se njegova iskoristivost s 30, na 60 i na kraju na 90%, no također mora se pripaziti i na sigurnost. Ovim postupkom od početna tri fizička servera oslobodila bi se dva, oni se sada mogu iskoristiti za neke druge zadatke ili trenutno ugasiti i na taj način smanjiti troškove održavanja [2].

2.2. Povijesni pregled

Priča seže u poprilično davno razdoblje 1960ih godina. Veliki znanstveni institut u kojem je zaposleno nekoliko stotina ljudi posjeduje samo jedno računalo. Ako bi koji znanstvenik trebao određen izračun, napisao bi svoj zadatak i proslijedio ga operateru koji je upravljao računalom. Operater bi pokrenuo program sa zadatkom te bi rezultate proslijedio nazad znanstveniku.

U tadašnje vrijeme računala su bila rijetka, spora i vrlo skupa. Ni tadašnji programeri nisu imali individualan pristup računalu jer se to smatralo neučinkovito, dok bi programeri unosili podatke ili razmišljali o mogućem rješenju svoga problema računala bi bila u pripravnom stanju, bez zadatka. Kako bi se to spriječilo, programeri i računala bili su odvojeni.

Nakon nekoliko godina stručnjaci su došli na ideju kojom bi omogućili većem broju korisnika korištenje računala u isto vrijeme. Dok bi jedan korisnik unosio podatke, računalo bi se bavilo zadacima drugih korisnika. Takav pristup minimizirao bi problem pripravnog stanja. Ideja je dobila naziv koncept vremenske podjele (engl. *Time-sharing concept*).

Prvi operacijski sustav koji je podržavao koncept vremenske podjele nazivao se Multics, predak današnje Unix obitelji. Iako je našao svoju svrhu, sam sustav bio je spor, nepouzdan i nesiguran. Želja za napretkom je postojala, ali hardverska ograničenja stvarala su problem. Rješenje tog problema zahtijevalo je suradnju s proizvođačima hardvera, nakon nekog vremena to je i ostvareno.

Godine 1968. IBM izumio je središnje računalo (engl. *mainframe*) koje je podržavalo CP/CMS sustav (Control Program/Conversational Monitor System), razvijen u suradnji sa znanstvenicima s Cambridgea. Bio je to prvi operacijski sustav koji je podržavao virtualizaciju. Zasnovan je na monitoru za upravljanje virtualnim strojem (engl. *virtual machine monitor*), takozvanom hipervizoru.

Hipervizor je bio korišten u svrhu stvaranja više virtualnih strojeva. Prednosti ovog pristupa u odnosu na koncept vremenske podjele bilo je mnogo, neke od njih su veća efikasnost zbog podjele resursa središnjeg računala umjesto starog „jedan po jedan“ pristupanja. Još jedna bitna stvar je ta da je svaki korisnik imao svoj operacijski sustav, na taj način cijeli je sustav postao pouzdaniji i niti jedan korisnik nije utjecao na podatke drugog korisnika.

Nakon nekog vremena CP/CMS bio je poboljšan, preimenovan i pripremljen za prodaju. Postao je baza za VM/370 (Virtual Machine Facility/370) operacijski sustav koji je korišten na jednom od najpopularnijih IBM-ovih središnjih računala, imena System/370.

Takvi sustavi uvelike su strukturno nalikovali na ove koji su današnjim modernim korisnicima poznati. Na središnje računalo spajali su se terminali (engl. *terminals*), središnje računalo bilo je veliko i snažno računalo dok su terminali bili uređaji s ekranom, tipkovnicom i mišem. Korisnici su radili na terminalima na način da su unosili podatke i zadatke koje je trebalo obaviti, a središnje računalo bi se pobrinulo za ostatak. Jedno središnje računalo moglo je posluživati stotine terminala.

Pred kraj 20. stoljeća osobna računala počela su se masovno širiti što je utjecalo na industriju koje se bazirala na središnjim računalima, kao i na virtualizacijske tehnologije. No, zbog visoke cijene i glomaznosti osobnih računala poduzeća su i dalje preferirala središnja računala u kombinaciji s terminalima.

Kako je vrijeme odmicalo tako su osobna računala postala manja i cjenovno prihvatljivija za privatna poduzeća. Kao produkt toga, osamdesetih godina 20. stoljeća osobna računala zamijenila su terminale. Središnja računala zajedno sa virtualizacijom prešla su u drugi plan, ali ne zadugo.

Kako su se u tom trenu osobna računala dovoljno proširila operacijski sustavi postali su funkcionalniji, ali manje pouzdani. Greška u jednoj aplikaciji mogla je utjecati na druge aplikacije, a u gorem slučaju i dovesti do rušenja cijelog operacijskog sustava. Kako bi poboljšali stabilnost sustava sistemski administratori (engl. *system administrators*) konfigurirali su svaki uređaj za samo jednu aplikaciju. Produkt toga bila je stabilnost, ali to je znatno utjecalo na troškove opreme. Ovo je bio trenutak kada se virtualizacija vratila na scenu, zbog mogućnosti korištenja više virtualnih strojeva na jednom fizičkom računalu umjesto ranijeg pristupa gdje bi se više računala zauzimalo za isti zadatak, ovo je bilo vrlo efikasno rješenje.

Razvojem operacijskih sustava javljala se i potreba za pokretanjem jedne vrste operacijskog sustava unutar druge vrste operacijskog sustava. Virtualizacija je mogla riješiti i taj problem. Godine 1988, predstavljen je SoftPC. Bio je to softver koji je dopuštao pokretanje Windows i MS-DOS aplikacija na drugim operacijskim sustavima. Nedugo nakon toga pojavio se i VirtualPC koje je omogućio pokretanje drugih operacijskih sustava na Windows operacijskom sustavu. Razvoj mrežnih tehnologija bio je još jedan razlog za nastavak korištenja virtualizacijskih tehnologija [3].

2.3. Detalji

Sama virtualizacija može se podijeliti na više zasebnih vrsta. Postoji mnogo podjela, jedna od podjela je na virtualizaciju hardvera, virtualizaciju softvera, virtualizaciju memorije, virtualizaciju pohrane, virtualizaciju mreže i *desktop* virtualizaciju [4]. Bitna činjenica za naglasiti je ta da u konkretnoj primjeni virtualizacije najčešće dolazi do kombinacije više navedenih vrsta u funkcionalnu cjelinu.

Virtualizacija hardvera je vrsta virtualizacije u kojoj se postojeći hardver virtualizira i stvaraju se virtualne kopije primjerice procesora, diska za pohranu i/ili grafičke kartice koje se dodjeljuju po potrebi. Sloj koji dodjeljuje i upravlja resursima naziva se hipervizor (engl. *hypervisor*).

Virtualizacija softvera se za razliku od virtualizacije hardvera temelji se na kontroliranju virtualiziranih okruženja, ali na razini softvera. Također daje mogućnost stvaranja više okruženja s jedne platforme. Ostvaruje se mogućnost upravljanja aplikacijama, operacijskim sustavima, raznim procesima potrebnim u virtualiziranom okruženju.

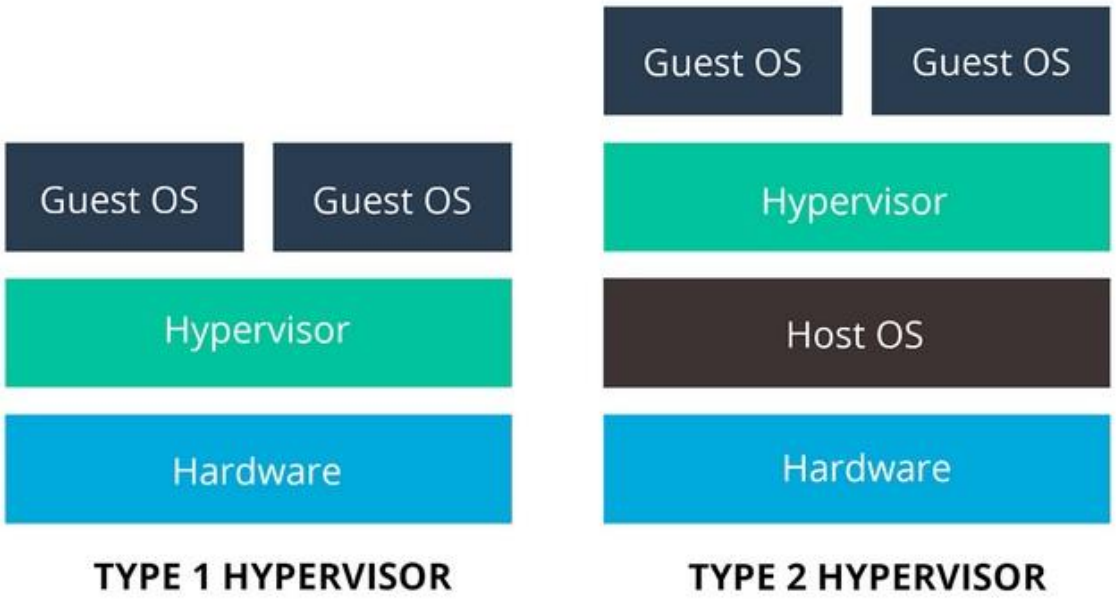
Virtualizacija memorije temelji se na principu agregacije postojeće fizičke memorije s više individualnih sustava u jednu cjelinu, svi operacijski sustavi imaju pravo na pristup toj memoriji. Same aplikacije pristupaju agregiranoj cjelini kroz *API* (Application Programming Interface).

Virtualizacija pohrane razdvaja fizički dio za pohranu u malu „mrežu“ jedinica za pohranu koje će virtualizirana okruženja koristiti. Svim manjim jedinicama može se pristupiti preko konzole za pristup. Ova vrsta virtualizacije pruža mnoge prednosti, neke od njih su mogućnost *backupa* i oporavka podataka [4].

Virtualizacija mreže koristi softver koji administrator mreže koristi kako bi upravljao mrežom s jedne konzole. Vrši se apstrakcija elemenata hardvera kao što su ruteri, mrežni preklopnici (engl. switches) i apstrahiraju se u softver koji je pokrenut na hipervizoru. Kao rezultat toga mrežni administratori dobivaju mogućnost upravljanja i modificiranja same mreže bez kontakta s fizičkim komponentama što uvelike pojednostavljuje sam proces upravljanja [5].

***Desktop* virtualizacija** pruža mogućnost stvaranja simuliranih okruženja na stotinama fizičkih uređaja, svi oni mogu biti upravljani i održavani od strane središnjeg administratora. Jedna od prednosti ove vrste virtualizacije je što se krajnji korisnici ne moraju brinuti o samom konfiguriranju ili instaliranju potrebnih stvari [2].

Na nekoliko se mjesta kroz tekst spominjao pojam hipervizor, radi se o vrlo važnoj tehnologiji koji omogućava mnoge vrste virtualizacije, no najčešće se spominje u kontekstu virtualnih strojeva. Koristi se kao sloj koji uzima fizičke resurse sustava na kojem se nalazi, dijeli ih i poslužuje virtualizirane sustave s resursima koji su im potrebni za normalno funkcioniranje. Hipervizori se mogu podijeliti u dvije kategorije: hipervizor tipa 1 i hipervizor tipa 2.



Sl. 2.3. Skica hipervizora tipa 1 (lijevo) i hipervizora tipa 2 (desno). Izvor: [10]

Slika 2.3. na pojednostavljen način prikazuje kako koji tip hipervizora izgleda. S lijeve strane nalazi se tip 1, drugog naziva *bare-metal* hipervizor. Kako mu samo ime govori, ovaj hipervizor radi izravno na hardveru računala na kojem se nalazi. Dakle, ne iziskuje potrebu za operacijskim sustavom kako bi obavljao svoje funkcije. Zbog direktnog pristupa hardveru sve potrebe za resursima dolaze puno brže do samih dijelova sustava koji su zatražili određeni resurs. Također, mnogi sigurnosni propusti specifični su za operacijske sustave stoga se ova vrsta smatra sigurnijom opcijom. S druge (desne) strane prikazana je shema hipervizora tipa 2, može se uočiti da je jedina razlika još jedan dodatni sloj, operacijski sustav. Naime, ova vrsta instalira se na već postojeći operacijski sustav, sve potrebne funkcije oslanjaju se osim na hipervizor i na operacijski sustav računala stoga dolazi do sporijeg izvršavanja zatraženih radnji. Zbog ranije navedenih razloga ova vrsta se također smatra manje sigurnom opcijom. Prilikom odabira koju vrstu koristiti u pitanje dolazi sama brzina, cijena i jednostavnost korištenja ove dvije opcije. Ukoliko se radi o tvrtkama najčešće će odabir pasti na tip 1, dok za prosječne korisnike tip 2 će sasvim zadovoljavajuće obaviti posao [10].

Nastavno na informaciju s početka potpoglavlja kako u konkretnoj primjeni virtualizacije najčešće dolazi do kombinacije više navedenih vrsta u funkcionalnu cjelinu, upravo u tome virtualizacijske tehnologije preuzimaju veliku zaslugu. Naime, kako na tržištu postoje mnoge tehnologije krajnji korisnici moraju odlučiti koje od njih će najbolje pridonijeti njihovim potrebama, radilo se o cijeni licence ili pak o uslugama koje sama tehnologija pruža. Neke od poznatijih su HyperV, KVM, Oracle VM VirtualBox, VMWare Workstation.

HyperV je dio svake novije verzije Windows operacijskog sustava, bez obzira što je sama tehnologija integrirana u operacijski sustav ne može se svrstati pod hipervizor tipa 2, spada pod hipervizor tipa 1, dakle sve radnje obavlja direktno na razini hardvera računala na kojem se nalazi. Dijeli se na dvije particije, na „roditeljsku“ particiju i na jednu ili više „nasljeđivačkih“ particija. Naime, nakon aktivacije ovog svojstva sam operacijski sustav seli se u virtualni stroj kojem je mjesto na ranije spomenutoj „roditeljskoj“ particiji, a na „nasljeđivačke“ particije smještaju se drugi, novi virtualni strojevi [6].

KVM punog imena *Kernel-Based Virtual Machine* je softver (hipervizor) otvorenog koda (engl. *open source*), dio je svakog Linux operacijskog sustava nastalog nakon 2007. godine. Sam KVM pretvara Linux u hipervizor tipa 1, zato što je dio Linux jezgre (engl. *kernel*) sadrži sve potrebne komponente pomoću kojih kontrolira resurse za potencijalne virtualne strojeve. Velika prednost ovog softvera je mogućnost vrlo brzog obavljanja operacija. Za uspješno korištenje potrebno je

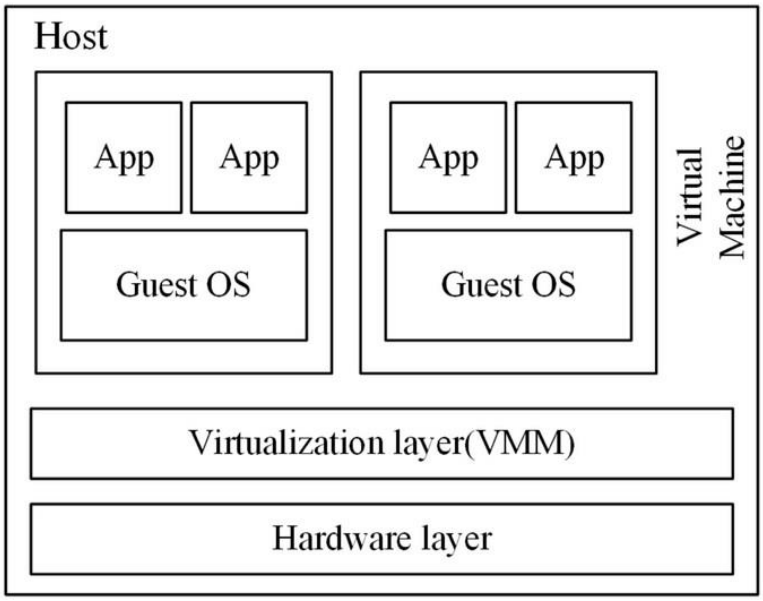
imati noviju verziju Linux sustava, kao što je ranije spomenuto, kao i hardver koji podržava virtualizaciju [7].

Oracle VM VirtualBox je također još jedan softver (hipervizor) otvorenog koda, razvijen od strane Innotek tvrtke 2007. godine. Svrstava se pod hipervizor tipa 2 i može biti instaliran na Windows, macOS, Linux, Solaris i OpenSolaris operacijskim sustavima. Također podržava virtualne strojeve svih navedenih operacijskih sustava. Sastoji se od dobrog grafičkog sučelja lakog za korištenje u kojem krajnji korisnik dobiva mogućnost pojedinog konfiguriranja svakog virtualnog stroja [8].

VMWare Workstation je softver (hipervizor) razvijen od tvrtke VMWare godine 1999. Kao i njegov prethodnik svrstava se pod hipervizor tipa 2 i sastoji se od dobrog grafičkog sučelja koje omogućava konfiguriranje svakog stroja zasebno. No postoje neke razlike, postoje dvije verzije od kojih se jedna plaća (VMWare Workstation Pro), a druga ne (VMWare Workstation Player). Može biti instaliran samo na Windows i Linux operacijskim sustavima, a podržava Windows, Linux, BSD i MS-DOS kao sustave za novonastale virtualne strojeve [9].

2.4. Virtualni stroj

Virtualni stroj je virtualno okruženje koje zapravo funkcionira kao normalan računalni sustav, sadrži svoju zasebnu memoriju, procesor, mjesto za pohranu podataka. „Ispod“ svakog virtualnog stroja nalazi se virtualizacijski sloj, odnosno hipervizor. Hipervizor obavlja sve potrebne funkcije upravljanja virtualnim strojevima, kao primjerice briga za zauzimanje svih potrebnih resursa koje bi jedan sustav (u ovom slučaju virtualni) trebao. Svaki taj virtualni sustav (stroj) pokreće svoj zasebni operacijski sustav i aplikacije, koristeći resurse koje hipervizor zauzima. Na jednom fizičkom računalu može se nalaziti više virtualnih strojeva, fizičko računalo naziva se *host* dok se svi kreirani virtualni strojevi nazivaju *guests*. Nadalje, svaki virtualni stroj ponaša se kao zasebno okruženje i ne utječe na ostale strojeve, niti na fizičko računalo na kojem se nalazi [11, 12]. Slika 2.4. na pojednostavljen način prikazuje virtualni stroj opisan u tekstu iznad. Naime, prvi (najniži) sloj predstavlja fizičko računalo na kojem se odvija virtualizacija, virtualizacijski sloj iznad predstavlja hipervizor, a iznad virtualizacijskog sloja prikazana su dva zasebna virtualna stroja, svaki sa svojim zasebnim operacijskim sustavom i aplikacijama.



Sl. 2.4. Skica sustava s virtualnim strojevima. Izvor: [13]

U odnosu na klasično računalo virtualni stroj pruža određene prednosti, a to su bolja iskoristivost resursa, portabilnost (engl. *portability*), fleksibilnost (engl. *flexibility*), sigurnost.

Zbog mogućnosti pokretanja više virtualnih strojeva na jednom fizičkom uređaju sami korisnici ne moraju kupovati novi uređaj u slučaju potrebe za drugim operacijskim sustavom, dakle puno bolje mogu iskoristiti postojeći hardver (resurse).

Virtualni stroj se može promatrati kao jedna datoteka (engl. *file*), kao rezultat toga javlja se mogućnost vrlo lako premještanja u slučaju potrebe [12]. Također, važno je spomenuti i razjasniti značenje par važnih pojmova usko povezanih s virtualnim strojem. *Snapshot* je stanje virtualnog stroja u određenom vremenu, analogno fotografiji. Daje mogućnost da se virtualni stroj vrati na snimljeno stanje, uklanjajući sve izmjene koje su se dogodile nakon samog *snapshota*. Naime, tvrdi disk virtualnog stroja (virtualni disk) zapravo je datoteka određene veličine, odluči li se korisnik napraviti *snapshot* svi dotadašnji podaci spremit će se u novu datoteku (engl. *file*) koja će od sada biti korištena. Sljedeće promjene nakon *snapshota* slagat će se u slojevima na trenutnu datoteku, a prilikom povratka na zadnju verziju *snapshota* odraditi će se uklanjanje novonastalih slojeva. Sljedeći pojam je migracija, radi li se o situaciji gdje je virtualni stroj privremeno zaustavljen i napravljen mu je *snapshot* koji je premješten na drugi fizički uređaj onda se taj proces može nazvati migracija. Ukoliko su prethodne verzije *snapshota* redovno sinkronizirane onda se radi o vrlo brzoj operaciji. Nastavno na migraciju dolazi do javljanja još jednog pojma, *failovera*.

Ukoliko se dogodila greška s fizičkim uređajem prilikom migracije *failover* će omogućiti nastavak rada virtualnog stroja. U ovoj situaciji stroj će nastaviti raditi na drugom raspoloživom fizičkom uređaju koristeći zadnje stanje za koje je napravljena kopija podataka (*backup*) [14].

Nadalje, zbog mogućnosti kloniranja postojećeg virtualnog stroja, koji već sadrži operacijski sustav, stvaranje novog stroja može biti puno brže i jednostavnije od klasičnog instaliranja operacijskog sustava na fizički uređaj.

Uspoređujući standardne operacijske sustave koji su direktno pokrenuti na fizičkom uređaju s virtualnim strojevima, virtualni strojevi su zasigurno bolja opcija u pogledu sigurnosti. Kako je sam virtualni stroj jedna datoteka, ukoliko je došlo do sumnjive radnje, na njoj se može izvršiti skeniranje s vanjskim programom. Također, *snapshot* je i u ovom pogledu od velike koristi jer ukoliko dođe do inficiranja virtualni stroj se može vratiti na stanje zadnje napravljenog *snapshota*. Brzo i jednostavno kreiranje virtualnih strojeva još je jedna od opcija ukoliko nešto pođe po krivu.

No kako svaka tehnologija ima određene nedostatke tako ih imaju i virtualni strojevi. Neki od njih su loše performanse prilikom pokretanja određenog broja virtualnih strojeva na jednom fizičkom uređaju. Virtualni strojevi su također manje efikasni i djelomično sporiji od normalnog računala [15].

Konkretnih primjena virtualnih strojeva je puno, neke od njih koje su korisne kako IT stručnjacima, tako i običnim korisnicima su sljedeće.

Virtualni stroj je odlična opcija za razvojne programere, ukoliko se javlja potreba za specifičnim postavkama koje će pomoći u razvoju softvera onda se svaki virtualni stroj može zasebno konfigurirati i tako stvoriti odgovarajuće okruženje za razvoj.

Virtualni stroj pruža mogućnost testiranja operacijskog sustava po želji, bez utjecaja na primarni operacijski sustav uređaja.

Također su dobra opcija za sve stručnjake koji se bave istraživanjem zlonamjernih programa, iskorištava se mogućnost lakog osposobljavanja svježih strojeva koji će služiti u svrhu testiranja.

Nadalje, situacija u kojoj se virtualni stroj također pokazao korisnim je ona u kojoj se javlja potreba za pokretanjem aplikacije u specifičnom operacijskom sustavu. Ukoliko korisnik primjerice koristi Appleov proizvod no ima potrebu pokrenuti Windows aplikaciju onda će to vrlo brzo realizirati s virtualnim strojem kreiranim s hipervizorom tipa 2.

Virtualni stroj pruža i mogućnost sigurnog pretraživanja internetom, kao i drugih korisničkih radnji. Prilikom bilo kakvog problema moguće je izvršiti povratak na ranije kreirani *snapshot* [12].

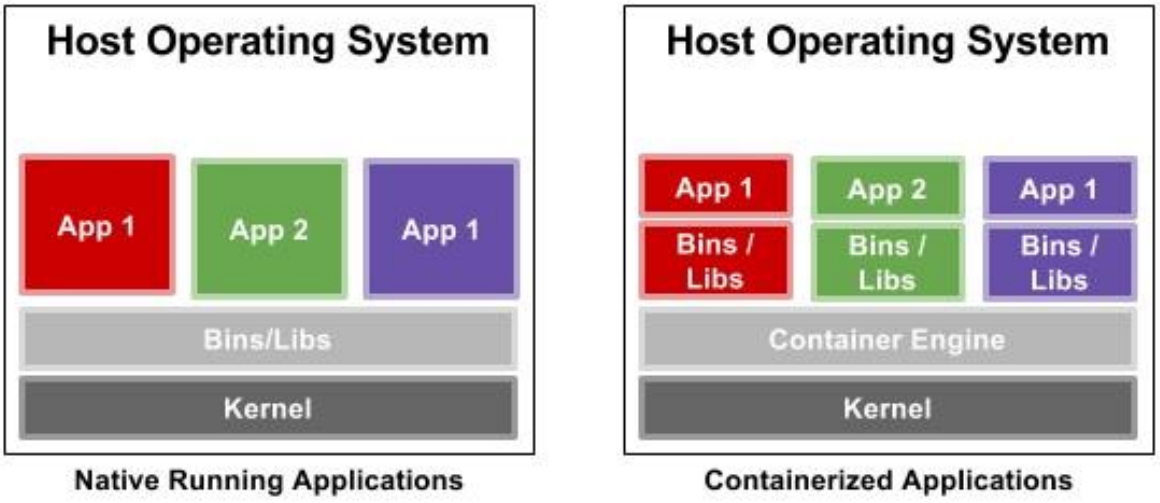
3. KONTEJNERI

U ovom se poglavlju opisuju kontejneri, što su kontejneri općenito, što je dovelo do pojave Linux kontejnera. Također, opisuju se tehnologije za kontejnerizaciju, s naglaskom na tehnologiju Docker. Spominju se i razlike u odnosu na virtualni stroj.

3.1. Općenito

Kontejneri su izvršne jedinice softvera u kojima se nalazi kod aplikacije, cijela jedinica izolirana je od ostatka sustava. Svaki kontejner sadrži potrebne programske biblioteke (engl. *library*) i ostatak konfiguracije potrebne za pokretanje na bilo kojem drugom sustavu ili uređaju. Kako bi se spomenuto postiglo, kontejneri iskorištavaju mogućnosti operacijskog sustava (u ovom slučaju radi se o Linux *kernelu*) na kojem se nalaze kako bi procesima omogućili potrebnu procesorsku snagu, memoriju, prostor i izolaciju. Sve ovo postignuto je kroz određene tehnologije koje su implementirane u ranije spomenutom *kernelu*, najpoznatije i često spomenute su cgrupe (engl. *cgroups*) i imenski prostori (engl. *namespaces*).

Najlakše shvaćanje kontejnera može se postići ukoliko se napravi usporedba s virtualnim strojevima. Svaki virtualni stroj sadrži svoj operacijski sustav, virtualizirani hardver, aplikacije s potrebnim programskim bibliotekama i konfiguracijom za njihovo uspješno pokretanje. Od svih navedenih komponenti kontejneri sadrže samo potrebne programske biblioteke i konfiguracijske datoteke za pokretanje aplikacija koje su unutar njih, odsutnost operacijskog sustava je jedan od glavnih razloga zašto su upravo kontejneri efikasni, brzi i prenosivi.



Sl. 3.1. Skica normalnog sustava (lijevo) i sustava koji sadrži kontejnere (desno). Izvor: [17]

Slika 3.1. prikazuje dvije skice, skica s lijeve strane prikaz je normalnog sustava, a slika desno prikazuje skicu sustava s kontejnerima. Sami kontejneri su na skici odvojeni slojem koji služi za njihovo upravljanje. Kako je navedeno u tekstu iznad, svaki kontejneri osim aplikacije sadrži i potrebne programske biblioteke, kao i ostale konfiguracijske datoteke preko kojih postaje lako prenosiv.

U tekstu koji slijedi navode se pozitivne strane njihovog korištenja. Kako je ranije navedeno, kontejneri dijele *kernel* sustava na kojem se nalaze, bez potrebe za kopijom cijelog operacijskog sustava za svaki kontejner, što ih čini malim datotekama koje ne zahtijevaju puno resursa. Nadalje, zato što je sa svakom aplikacijom „zapakirana“ i bilo koja druga datoteka ili programska biblioteka, na svakom napisanom dijelu softvera ne mora se više vršiti rekonfiguriranje prilikom njegovog ponovnog korištenja na drugom stroju. Kao što virtualni strojevi doprinose boljoj iskoristivosti resursa sustava, tako isto i kontejneri vrlo dobro koriste procesorsku snagu i memoriju fizičkog stroja. No, u određenom dijelu kontejneri puno bolje obavljaju posao. Naime, radi se o velikim aplikacijama koje se mogu rastaviti na manje dijelove, te dijelove moguće je podijeliti u više kontejnera i na taj način povećati iskoristivost sustava, ovaj princip podjele naziva se arhitektura mikroservisa (engl. *microservice architecture*) [16].

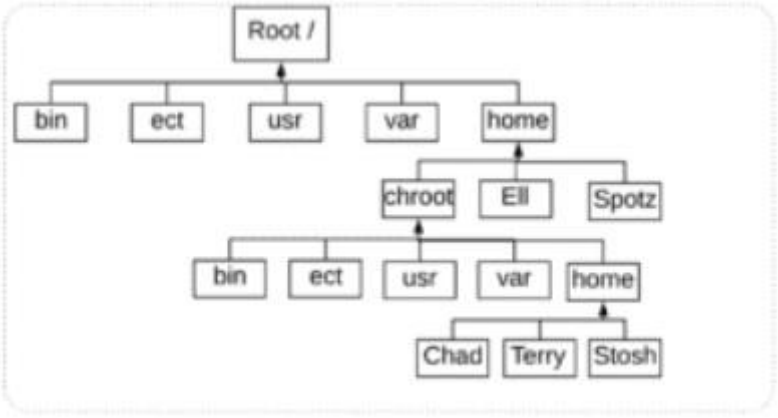
Kao i svaka druga tehnologija kontejneri također imaju određene mane koje je potrebno upoznati prije odluke na njihovo korištenje. Kontejneri svakako koriste resurse opreznije od virtualnih strojeva, no zbog raznih slojeva koji su uključeni prilikom njihovog korištenja ni oni ne mogu pružiti savršene performanse. Nadalje, kako različite tvrtke pružaju različite tehnologije za upravljanje kontejnerima normalno je očekivati određene nekompatibilnosti ukoliko bi došlo do pokušaja kombiniranja više tehnologija. Također, prilikom korištenja kontejnerskih tehnologija u Linux operacijskom sustavu smatra se da korisnik dobro poznaje komandnu liniju, ukoliko bi se javila potreba za grafičkim sučeljem postoje određene tehnike kako bi se to moglo postići. Takva potreba zasigurno komplicira priču i stoga odsustvo grafičkog sučelja, po standardnim postavkama, može se navesti kao mogući nedostatak [28].

Istaknutost kontejnera u posljednje vrijeme povećala se do te mjere da određene tvrtke u budućnosti planiraju koristiti kontejnere kao zamjenu za virtualne strojeve koje su do sada koristili. No, evo nekoliko konkretnih slučajeva u kojima su kontejneri korisni i danas. Ranije spomenuta arhitektura mikroservisa, kako su kontejneri sami po sebi mali potrošači prostora dobar su odabir ukoliko se radi o aplikacijama koje su konstruirane od mnogo malih, samostalnih dijelova. Također, ukoliko se radi o kombinaciji arhitekture mikroservisa i kontejnera kao platforme onda

je to dobar temelj za timove koji koriste *DevOps* (development and operations) način rada. I još jedan od vrlo čestih slučajeva kada se priča o kontejnerima je taj da se koriste kada je potrebno „modernizirati“ aplikacije, proces prebacivanja obične aplikacije u kontejner naziva se kontejnerizacija [16].

U ranijim dijelovima teksta navedena su određena objašnjenja, prednosti i slučajevi korištenja kontejnera, no kako bi se pobliže shvatila cijela priča nastanka ove tehnologije, kao i popratnih tehnologija koje ju omogućuju potrebno se vratiti nekoliko desetljeća nazad.

Godine 1979. predstavljena je *chroot* (change root) naredba, kako joj samo ime govori ova naredba omogućavala je promjenu trenutnog direktorija pokrenutog procesa u *root* direktorij, kao i svih njegovih hijerarhijskih poddirektorija. Ovim se stvorila mogućnost izoliranja procesa u svoj zasebni datotečni sustav koji ne bi utjecao na ostatak sustava. U ožujku 1982. godine, *chroot* naredba postala je dio Unix operacijskog sustava. Slika 3.2. prikazuje na koji način dolazi do promjene u poretku direktorija datotečnog sustava korištenjem ove naredbe, odnosno položaj novonastalog „*root“* direktorija.



Sl. 3.2. Prikaz hijerarhije direktorija u kojem je korištena chroot naredba. Izvor: [18]

Sljedeća važna godina u cijeloj priči je 1990. kada je stručnjak za sigurnost i mreže, Bill Cheswick u jednom svom istraživanju pokušavao shvatiti na koji način bi se haker ponašao ukoliko bi dobio pristup sustavu. U svrhu istraživanja izrađeno je okruženje koje bi mu omogućilo praćenje postupaka hakera, njegovo rješenje bilo je koristiti *chroot* okruženje s određenim izmjenama. U ožujku 2000. godine FreeBSD uveo je *jail* naredbu u svoj sustav. Iako vrlo slična *chroot* naredbi, ova naredba ipak uključuje određenu izolaciju procesa u vidu datotečnog sustava, korisnika, mreža. *Jail* naredba u FreeBSD sustavu davala je mogućnost zasebnog podešavanja i izvršavanja instalacija u svakoj „ćeliji“, no aplikacije unutar „ćelija“ bile su ograničenih funkcionalnosti.

Priča se nastavlja 2006. godine, inženjeri Googlea najavljuju objavljivanje njihove nove tehnologije, proces kontejnera (engl. *process containers*) dizajniranih sa svrhom ograničavanja i izoliranja resursa koje određeni proces koristi. Godinu poslije tehnologija je preimenovana u kontrole grupe (engl. *control groups, cgroups*) kako bi se izbjegla zamjena s pojmom kontejner.

Još jedna važna godina u nizu je zasigurno 2008., godina u kojoj su cgrupe postale dio Linux *kernela,* situacija koja je dovela do LinuX Container (LXC) tehnologije [18]. LXC je bila prva potpuna implementacija Linux kontejner *managera*. Od tehnologija koristi cgrupe i imenske prostore [18].

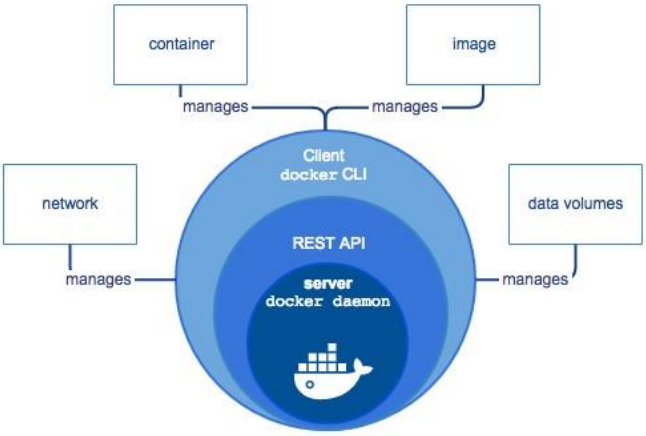
Kontrolne grupe/cgrupe svojstvo su Linux *kernela* koje omogućava procesima da budu organizirani u hijerarhijski oblikovane grupe čije različite skupine resursa tada mogu biti kontrolirane i ograničene. Grupiranje se odvija u jezgri *kernel* koda cgrupa, dok se praćenje, kontroliranje i ograničavanje vrši u podsistemima (engl. *subsystems*) od kojih je svaki zadužen za određenu vrstu resursa (procesor, memorija, ...) [19].

Imenski prostor izvršava apstrakciju globalnog resursa na takav način da proces koji se nalazi u imenskom prostoru ima dojam kako posjeduje svoju izoliranu instancu apstrahiranog globalnog resursa. Što bi značilo da ovo svojstvo omogućava izolaciju globalnih resursa između neovisnih procesa, ovo je uvelike korisno prilikom korištenja kontejnera kako bi se uspješno postigla izolacija. Postoji nekoliko vrsta imenskih prostora, svaka od njih fokusira se na specifičnu skupinu resursa. To su pid imenski prostor (izolacija procesa), net imenski prostor (upravljanje mrežnim sučeljima), ipc imenski prostor (upravljanje međuprocesnom komunikacijom), mnt imenski prostor (upravljanje točke *mountanja* u datotečnom sustavu), uts imenski prostor (izolacija naziva *hosta* i naziva domene), user imenski prostor (izoliranje korisničkog ID-a i ID-a grupe), imenski prostor cgrupe (skrivanje identiteta cgrupe kojoj proces pripada) [20, 21].

3.2. Docker i ostale tehnologije

Docker

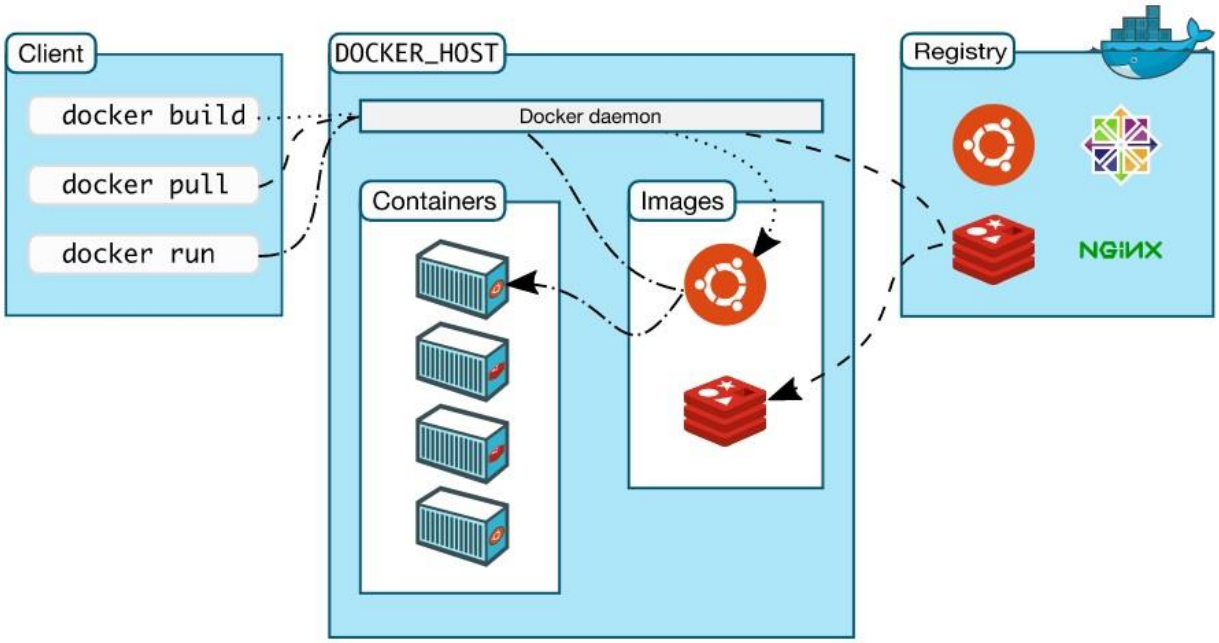
Docker je kontejnerizacijska tehnologija koja s raznim implementiranim alatima pruža mogućnost lakšeg razvoja, prijenosa i objavljivanja željenog softvera. Temelji se na Docker engineu koji se može definirati kao klijent-server komponenta, a sastoji se od dijelova prikazanih na slici 3.3.



Sl. 3.3. Prikaz Docker enginea. Izvor: [22]

Docker client (CLI) prva je linija korisnikove komunikacije s Dockerom. Prilikom unosa naredbi koju počinju s „*docker ...*“, client ih prosljeđuje do Docker daemona koji ih izvršava, a sama komunikacija odvija se preko Docker API sloja. Client posjeduje mogućnost komunikacije s više različitih Docker daemona.

Kako je ranije navedeno zadaća Docker daemona je primanje zahtjeva preko API sloja, a zahtjevi se izvršavaju nad objektima. Pod objekte spadaju slike (engl. *image*), kontejneri, mreže, jedinice za pohranu (engl. *volumes*). Docker daemon također posjeduje mogućnost komunikacije s drugim Docker daemonima ukoliko se javi potreba.



Sl. 3.4. Prikaz Docker arhitekture. Izvor: [22]

Naime, na slici 3.4. prikazano je kako izgleda arhitektura Dockera. Arhitektura je temeljena na odnosu klijent-server. Docker client komunicira s Docker daemonom koji je zadužen za sve konkretne poslove s kontejnerima, client i daemon imaju mogućnost rada s istog sustava ili kao druga opcija je mogućnost spajanja clienta na udaljeni daemon. Nadalje, ukoliko naredba koju je client proslijedio traži gotovu sliku onda se iskorištava registar koji je prikazan s desne strane. Docker Hub je registar koji je postavljen po standardnim postavkama. Također, korisnicima se pruža mogućnost postavljanja i korištenja drugih, kako privatnih tako i javnih registara.

U tekstu koji slijedi objašnjavaju se ranije spomenute slike i kontejneri.

Docker slika (engl. *image*) definira se kao predložak koji sadrži upute kako će izgledati kontejner kreiran iz nje. Jedno od svojstava slike je to da je njen sadržaj neizmjenjiv, dakle moguće ga je samo iščitati i na temelju njega kreirati druge slike s određenim izmjenama. Ukoliko se koristi svojstvo kreiranja vlastite slike na temelju neke druge (s određenim izmjenama) onda je registar od velike pomoći, ali ako korisnik ima potrebu stvoriti potpuno vlastitu onda bi prvi korak trebao biti kreiranje *Dockerfilea*. To je datoteka koja sadrži retke u kojima su napisani koraci koji će se slijediti prilikom kreiranja nove slike. Za svaki redak u *Dockerfileu* stvara se novi sloj u novonastaloj slici, a prvi sloj naziva se osnovna slika (baza).

Docker kontejner definira se kao instanca slike koja se može pokrenuti, svaki kontejner se osim pokretanja može zaustaviti, premjestiti ili obrisati. Svojstva kontejnera određuju se na temelju slike od koje je nastao, kao i dodatnih svojstava koja se definiraju prilikom njegovog pokretanja. Ukoliko bi se povukla poveznica s objektno orijentiranim programiranjem, slike bi predstavljale klasu, dok bi kontejneri (kao instance slika) predstavljali objekte. Prilikom uklanjanja kontejnera svi podaci koji nisu pohranjeni bit će izgubljeni, upravo zbog toga koriste se jedinice za pohranu (engl. *volumes*).

Tehnologije koje Dockeru omogućavaju uspješno izvođenje poslova s kontejnerima su cgrupe, imenski prostori (objašnjeno u prethodnom potpoglavlju 3.1.) i Union datotečni sustavi (UnionFS). Ovi datotečni sustav temelje se na kreiranju slojeva što pridonosi brzini i manjem korištenju resursa [22, 23]. Konkretna primjena ove tehnologije u Dockeru je kako bi se izbjeglo dupliciranje cijele skupine datoteka prilikom pokretanja svakog novog kontejnera, kao i mogućnost ponovnog pokretanja kontejnera s početnog stanja (nakon njegovog uklanjanja) jer omogućava izoliranje promjena datotečnog sustava kontejnera u zasebni sloj [24].

LXC

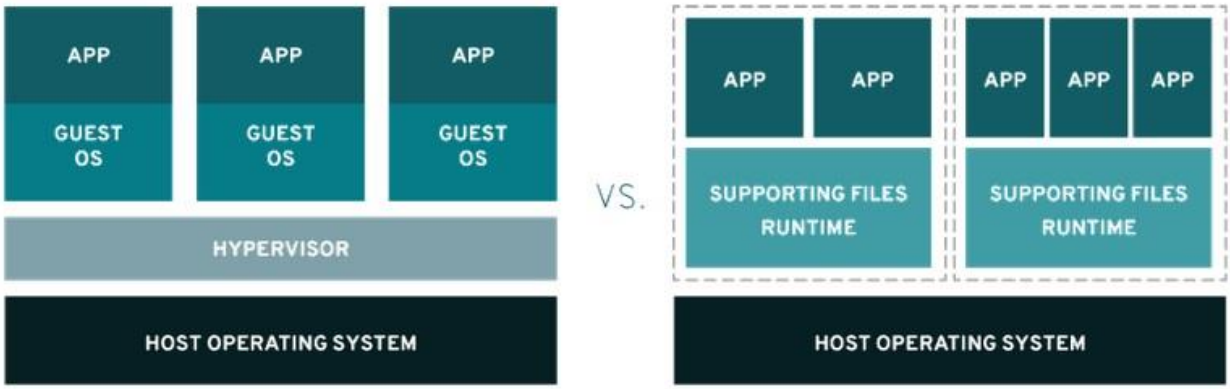
Punog imena LinuX Containers jedna je od tehnologija koja omogućuje kontejnerizaciju. Temelji se na tehnologijama implementiranim u Linux *kernelu*, cgrupama i imenskim prostorima (objašnjeno u prethodnom potpoglavlju 3.1.) koje omogućuju upravljanje resursima i potrebnu izolaciju. Sama tehnologija bila je prva inačica za upravljanje kontejnerima kakve su i danas poznate, sam Docker svoje je temelje gradio upravo na ovoj tehnologiji. Konkretno gledajući LXC pruža mogućnost izolacije aplikacija, ali i kompletnih operacijskih sustava. Tehnologija je fokusirana na kreiranju manje zahtjevnih verzija virtualnih strojeva, u vidu resursa. Kako bi se lakše formirala slika o LXC tehnologiji najbolje ju je usporediti s mogućnostima koje pruža Docker. Docker u većem dijelu pažnju pridaje aplikacijama koje imaju mogućnost rastavljanja na manje nezavisnih dijelova (*microservice architecture*) od kojih se svaki može pokretati u zasebnom kontejneru i stoga je pogodniji za razvojne programere. LXC s druge strane radi na principu inicijalizacije operacijskog sustava za svaki kontejner (što bi značilo više procesa u kontejneru), ovo je postignuto mijenjanjem *root* direktorija kako bi se kreirao novi datotečni sustav. Ova tehnologija također vrši komunikaciju preko sučelja za unos naredbi (CLI) i usko je vezana s cijelom Linux zajednicom stoga većina obično korištenih alata radit će i na aplikacijama koje se vrte unutar LXC-a. Na kraju krajeva LXC kontejneri, kako je spomenuto, u sebi imaju instancu Linux operacijskog sustava. Još par stvari važno je napomenuti, jedna od njih je zastupljenost gotovih slika. Docker zbog svog Docker Huba uvelike prednjači u ovom aspektu i moglo bi se reći da upravo zbog takvih koraka postaje puno prihvaćeniji u zajednici. Također, zbog pristupa sustavu na niskoj razini LXC kao tehnologija smatra se pogodnijom za sistemske administratore [25].

Solaris Containers

Solaris Containers tehnologija je koja kombinira kontroliranje resursa sustava skupa s definiranim ograničenjima koja određuju zone (engl. *zones*). Svaka zona ponaša se kao zasebni virtualni sustav unutar sustava na kojem se nalazi i može se smatrati kao koncept sličan *jail* naredbi FreeBSD sustava (objašnjeno u prethodnom potpoglavlju 3.1.), ali s boljim naglaskom na sigurnosti i s boljom implementacijom u operacijski sustav [26]. Zone se dijele na globalne i ne globalne. Globalna zona ima mogućnost pregleda svih resursa na sustavu bez obzira odnose li se oni na samu globalnu zonu ili na druge ne globalne zone. Nadalje, svaka zona ima vlastiti ID (procesi unutar nje također dijele isti ID), pristup mreži, kao i prostor za pohranu. Minimalan i obavezan resurs je svakako disk za pohranu s podacima o konfiguraciji same zone. Naime, ne postoji nužna potreba

za procesorskom snagom, memorijom ili nekim drugim resursom, no u slučaju potrebe i ti resursi mogu biti dodijeljeni. Također, svaka zona sadrži definirana ograničenja zbog kojih procesi unutar nje nisu u mogućnosti komunicirati ili promatrati procese izvan nje. Zbog ove izolacije, svaka zona ima zasebnu listu korisnika. Ukoliko se radi o dva korisnika, iz dvije različite zone, oni mogu imati isti ID. Svaka zona može se nalaziti u različitim stanjima, a neka od njih su „spremno“, „pokrenuto“, „ugašeno“... Zbog manjka moguće interakcije sa zonama smatra se da ni ova tehnologija nije najpogodnija za razvojne programere, ali je pogodna za pružanje poslužiteljskih usluga [26, 27].

3.3. Razlike u odnosu na virtualni stroj



Sl. 3.5. Skica sustava s virtualnim strojevima i sustava s kontejnerima. Izvor: [29]

Na slici 3.5. s lijeve strane prikazan je sustav s hipervizorom tipa 2 i nekoliko virtualnih strojeva, dok je s desne strane prikazan sustav koji sadrži kontejnere. Prva vidljiva razlika je odsustvo više operacijskih sustava u sustavu s kontejnerima, dakle koristi se samo jedan sustav (onaj na kojem se nalaze) i njegove mogućnosti. Suprotno kontejnerima, svaki virtualni stroj sadrži potpuni vlastiti operacijski sustav. Upravo zbog toga veličina se navodi kao jedna od ključnih razlika ove dvije tehnologije, veličina kontejnera može se mjeriti u megabajtima dok veličina virtualnih strojeva doseže nekoliko gigabajta. Zbog ove činjenice na sustavu se može nalaziti puno više kontejnera nego virtualnih strojeva.

Naime, kako virtualni strojevi koriste operacijski sustav sa svim popratnim programima i postavkama smatra se da je korisnik u prednosti zbog mogućnosti boljeg upravljanja i bolje pokrivenosti resursima. No, činjenica je da potpuni operacijski sustav zbog svoje veličine može potrošiti nekoliko minuta prije nego li je spreman za korištenje, kontejneri uvelike prednjače u ovom aspektu jer postaju spremni za korištenje u puno kraćem roku.

Ranije u radu navedeno je na koji način kontejneri iskorištavaju mogućnosti *kernela* kako bi uspješno funkcionirali, no u vidu sigurnosti ova činjenica u prvu ruku može biti potencijalni rizik. Ukoliko kontejner nije dobro konfiguriran, svaki korisnik s punim pravom pristupa mogao bi pristupiti operacijskom sustavu i prikupiti informacije o drugim kontejnerima. S druge strane virtualni strojevi sami po sebi mogu se smatrati sigurnijim okruženjem jer je sve potrebno sadržano u jednom virtualnom stroju, dakle ne javlja se potreba za komunikacijom (sa sustavom na kojem se nalaze) u količini u kojoj se javlja prilikom korištenja kontejnera [30].

U vidu konkretnog korištenja kontejneri su vrlo korisni ukoliko se javlja potreba za pokretanjem puno manjih programa na jednom sustavu, ukoliko je potrebno pokrenuti isti program nekoliko puta, ukoliko se javlja potreba za sustavom koji ne zahtjeva puno resursa i brzo ga je moguće pokrenuti, ukoliko korisnik ne želi voditi brigu o konfiguraciji i konfiguriranju programa kako bi funkcionirali na različitim sustavima. Virtualni strojevi korisni su u slučajevima potrebe za više različitih operacijskih sustava, ukoliko se javlja potreba za upravljanjem više većih programa na jednom sustavu, ukoliko se pokreće program koji zahtijeva sve funkcionalnosti i resurse operacijskog sustava, ukoliko je potrebna potpuna izolacija i sigurnost [31].

Za kraj, kako bi se slikovito opisala razlika između ove dvije tehnologije moguće je dati jednu vrlo zanimljivu usporedbu. Pojmovi koji su korišteni su kuća (virtualni stroj) i stanovi (kontejneri). Kuća, sa svim svojima instalacijama, kao i određenom dozom privatnosti od neželjenih posjetitelja vrlo je neovisna, ali i velika. Stanovi također sadrže sve iste instalacije kao i kuća, no ipak se te instalacije dijele i s ostalim stanovima. Stanovi dolaze u različitim veličinama i stanari iznajmljuju stan one veličine koja im je zapravo potrebna. Ovo su samo neki od slučajeva zašto se ova usporedba može prenijeti i na tehnologije kao što su kontejneri i virtualni strojevi [23].

3.4. Koncept kontejnerizacije u svrhu uspostavljanja servisa

Prilikom donošenja odluke o kontejnerizaciji važno je donijeti odluku vrijedi li započinjati cijeli proces. Postavljaju se pitanja je li trenutni proizvod spreman za pokretanje u kontejneru, hoće li nakon uspješne pripreme sve dobro funkcionirati i hoće li sve proći bez greške.

Kontejneri su zasigurno vrlo dobra opcija za proizvode koji se mogu podijeliti u više neovisnih dijelova. No, ukoliko se radi o zastarjelom proizvodu koji nije tako lako djeljiv potrebno je uzeti u obzir koliko je vremena potrebno utrošiti u samu pripremu za prebacivanje.

Po prirodi kontejneri se mogu okarakterizirati kao lako prenosivi, što bi značilo da su vrlo lako ponovno iskoristivi. Ukoliko se pokuša napraviti nešto zahtjevnije vrlo lako se može stvoriti „mali

virtualni stroj“. U tom slučaju, radi li se o većem programu, javlja se i veća potreba za kontrolom. Podjela programa na manje funkcionalne dijelove zbog lakšeg rukovanja vrlo je koristan postupak, ali ujedno povlači sa sobom i obveze pravilnog upravljanja s više novonastalih cjelina. Osim pravilnog upravljanja potrebno je i voditi određene detalje o svakom kontejneru jer ih sami po sebi ne pružaju previše što je potencijalni problem prilikom njihovog većeg broja.

Također, ako je proizvod koji je potrebno preseliti u kontejnere proizvođač velikog broja ulaznih/izlaznih operacija koje je potrebno pratiti u obzir se moraju uzeti jedinice za pohranu. Većina zastarjelih proizvoda koji će se kontejnerizirati ovisna je o spremljenim podacima i zbog toga se javlja potreba za konstantnom jedinicom za pohranu. Kako su kontejneri lako promjenjivi novim kontejnerima mora se također osigurati mogućnost korištenja postojećih podataka. Nadalje, u obzir treba uzeti i na koji način će prebacivanje u kontejnere (po mogućnosti više manjih) utjecati na stanja programa koja se spremaju jer će od sada na njih utjecati više manjih jedinica.

Zbog načina na koji kontejneri pristupaju sustavu na kojem se nalaze sigurnost je jedna od tema koje su često spomenute kada se priča o kontejnerizaciji. Osim brige o sigurnosti sustava na kojem se kontejneri nalaze potrebno je voditi i brigu o njihovoj izolaciji. Jedan od načina na koji se to može postići je da im se dodijele najniže privilegije u vidu prava pristupa. Konkretno, radi se o pokretanju programa unutar kontejnera s korisnikom koji ima obična prava pristupa. Također, moguće je ograničiti korištenje resursa po kontejneru i na taj način isključiti napade kojima je cilj iscrpljivanje resursa. Nadalje, prilikom kreiranja novih slika dobro je voditi brigu o osnovnoj slici, o broju njenih skidanja, ocjena i dolazi li iz sigurnih izvora.

Nadgledanje i praćenje stanja je još jedan dio koji je zasigurno važan, kao i svi ostali koji su ranije u tekstu navedeni. Ako bi se ovaj aspekt izostavio onda bi prilikom javljanja grešaka vrlo teško uspjeli saznati uzrok, a samim time i djelovati sukladno. Polazi li se od činjenice da proizvod koji se želi kontejnerizirati već sadrži određena rješenja za nadgledanje i praćenje stanja, u tom slučaju se ne bi trebali javljati veći problemi. Docker primjerice pruža veći broj takvih mogućnosti i omogućuje izvoz povratnih informacija aplikacije u različite formate [32].

Sve do sada navedeno dio je stvari na koje treba paziti kako bi se novonastali prijelaz s „klasičnog“ načina u kontejnere pravilno izveo i kako bi na kraju krajeva pokazao određene rezultate.

4. USPOSTAVLJANJE WEB APLIKACIJE KORISTEĆI DOCKER

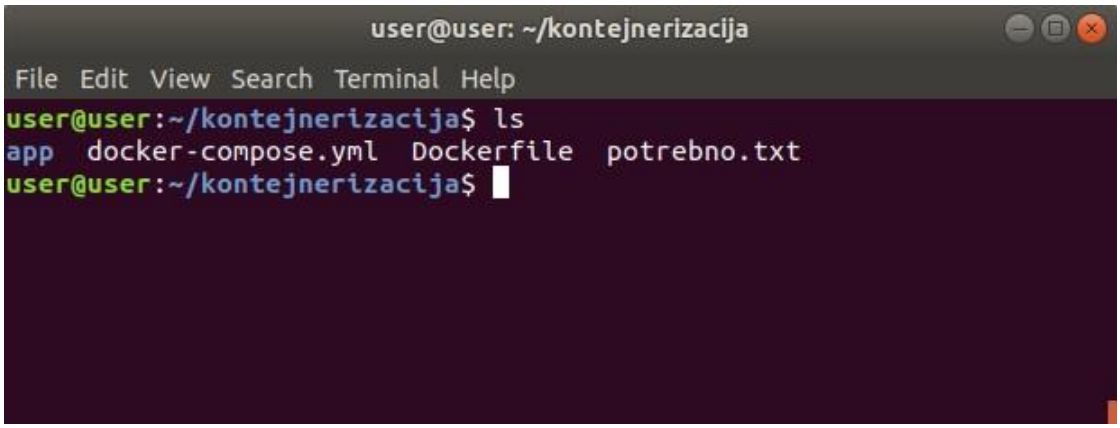
U ovom poglavlju opisana je i prikazana kontejnerizacija i njeni koraci. Opisuju se detalji implementacije, a zatim i sami koraci koji su potrebni kako bi se izvršila kontejnerizacija.

4.1. Detalji implementacije

U narednom tekstu prikazuje se postupak kontejnerizacije jednostavne Djagno web aplikacije koristeći Docker. Kontejnerizacija će se izvesti na način da će se aplikacija, kao i svi njeni popratni dijelovi postaviti u jedan kontejner, dok će se njena baza podataka nalaziti u drugom kontejneru. Komunikacija i pravilno funkcioniranje ta dva kontejnera postići će se specifičnom Docker datotekom. Konkretni koraci i detalji opisani su u sljedećem potpoglavlju.

4.2. Postupak implementacije

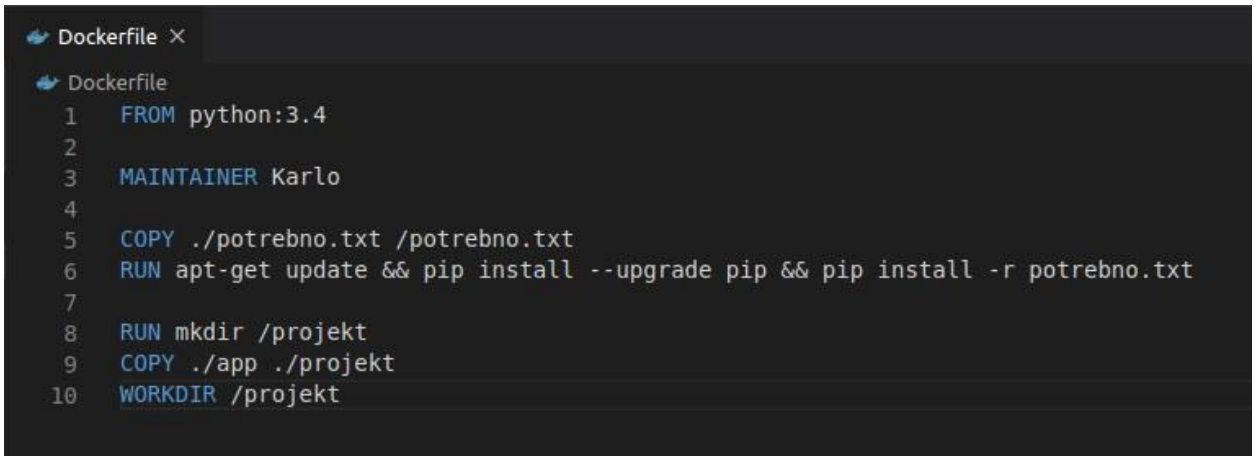
Slika 4.1. prikazuje trenutni direktorij u kojem će se izvršiti kontejnerizacija, unutar njega nalazi se sama web aplikacija, kao i sve potrebne datoteke kojom će se ona kontejnerizirati.



Sl. 4.1. Prikaz trenutnog direktorija i pripadajućih datoteka

Direktorij *app* sadrži jednostavnu Django aplikaciju koja na početnoj stranici ispisuje poruku „Hello World“. Datoteka *docker-compose.yml* služit će za povezivanje dva kontejnera, za njihovu komunikaciju, kao i definiranje određenih potrebnih vrijednosti. *Dockerfile* datoteka je datoteka na temelju koje će se izgraditi slika koja će sadržavati aplikaciju i popratne programe potrebne za pravilno funkcioniranje kontejnera koji će nastati iz te slike. Datoteka *potrebno.txt* vezana je uz *Dockerfile* datoteku, koristit će se za definiranje verzija programa koji će se instalirati unutar slike na temelju *Dockerfilea*.

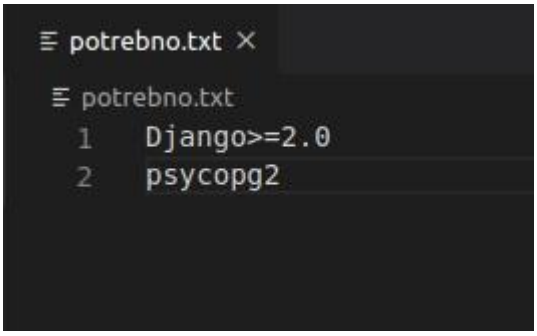
Dockerfile je tekstualna skripta koji sadrži posebne naredbe za izgradnju slika. Naredbe unutar datoteke nazivaju se instrukcije. Instrukcija se sastoji od dva dijela, prvi dio je sama instrukcija, dok u drugi dio spadaju argumenti. Instrukcija se može pisati velikim i malim slovima, no praksa je pisati velikim slovima kako bi se razlikovala od argumenata koji slijede nakon nje. Unutar Dockerfilea postoji mogućnost korištenja gotovo 20 instrukcija [23], no samo neke od njih bile su potrebne kako bi se kontejnerizacija koja slijedi uspješno izvela. Kako bi se Dockerfile datoteka mogla pravilno interpretirati mora imati veliko početno slovo i ne smije imati nastavak.



Sl. 4.2. Prikaz Dockerfile datoteke i instrukcija unutar nje

Slika 4.2. prikazuje konkretnu Dockerfile datoteku koja će služiti za izgradnju slike koja će sadržavati aplikaciju, kao i popratne programe koji će omogućiti njeno funkcioniranje. Instrukcija FROM smatra se najvažnijom instrukcijom Dockerfile datoteke, ona definira osnovnu sliku na kojoj će se temeljiti novonastala slika. Po standardnim postavkama Docker pretražuje sustav na kojem se nalazi kako bi iskoristio sliku za FROM instrukciju, ukoliko na sustavu ne postoji takva slika onda će ju skinuti s javno dostupnog Docker Hub registra [23]. U ovom slučaju za bazu buduće slike koristit će se Python verzije 3.4 (verziju, odnosno *tag*, definira vrijednost nakon dvotočke, ukoliko se *tag* ne definira po standardnim postavkama Docker će koristiti najnoviju verziju dane slike) jer će se unutar kontejnera nastalog iz ove slike nalaziti Django web aplikacija, a Django je temeljen na Python programskom jeziku. Ne postoji poseban razlog zašto je korištena baš verzija 3.4, to je jedna od verzija koja je kompatibilna primjerice s verzijom Djanga koji će biti instaliran u jednoj od sljedećih naredbi. MAINTAINER instrukcija je informativni dio Dockerfile datoteke i sadrži detalje o autoru, može se nalaziti u bilo kojem dijelu datoteke, no praksa je stavljati je odmah nakon FROM instrukcije. U ovom slučaju detalji su samo ime autora. COPY instrukcija daje mogućnost kopiranja datoteka sa sustava na kojem je Docker instaliran u datotečni sustav slike koja će nastati iz Dockerfilea. Prvi argument nakon COPY instrukcije

prikazuje putanju od direktorija u kojem se Dockerfile nalazi, a drugi argument prikazuje putanju unutar slike koja će nastati. Za drugi argument preporučuje se korištenje apsolutne putanje. COPY instrukcija također može stvoriti nove direktorije unutar slike koja će nastati. U petoj liniji Dockerfilea naredba COPY služi kako bi se kopirala tekstualna datoteka *potrebno.txt* koja će biti korištena u naredbi koja dolazi nakon nje. Sadržaj *potrebno.txt* datoteke prikazan je slikom 4.3., unutar nje definirane su potrebne instalacije. Prva je Django verzije 2.0 ili novije, a druga je psycopg2. Psycopg2 najpopularniji je adapter za Postgres bazu podataka koja će kasnije biti korištena u drugom kontejneru, a povezat će se s prvi kontejnerom koji sadrži web aplikaciju.

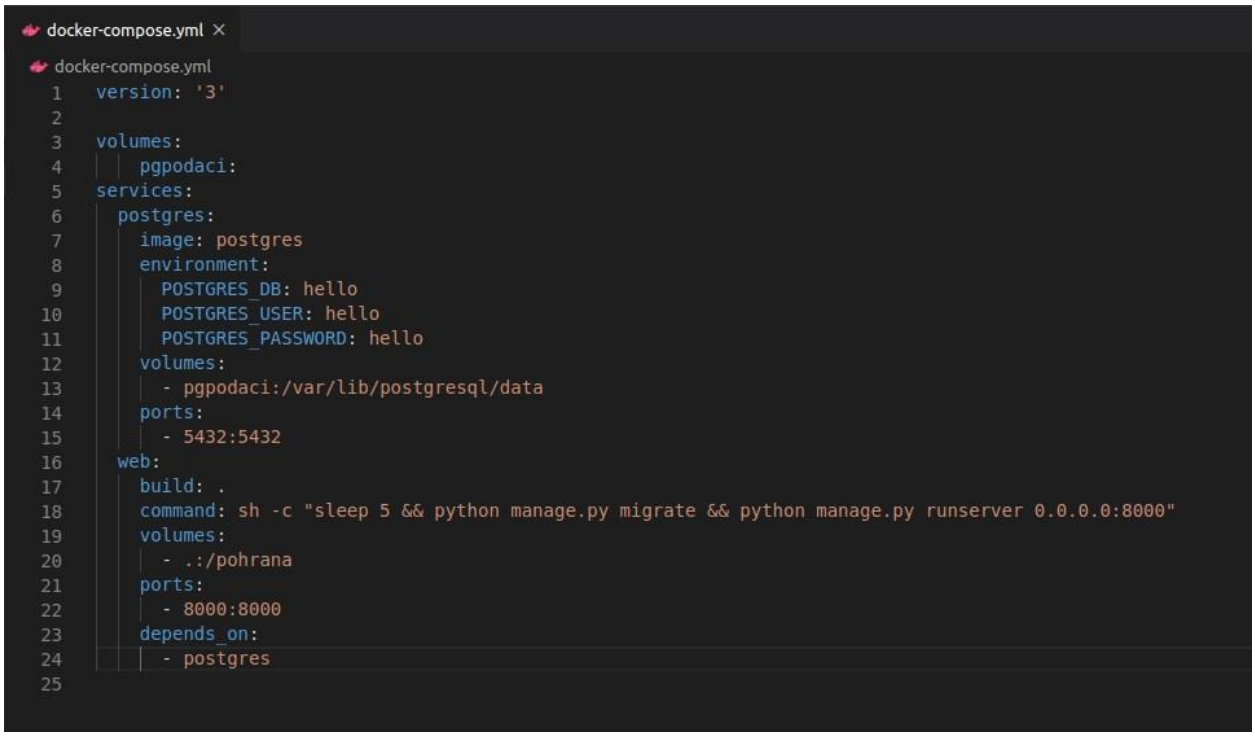


Sl. 4.3. Prikaz sadržaja potrebno.txt datoteke

RUN instrukcija kako joj i samo ime govori služi za pokretanje naredbi. Općenita preporuka je navesti više naredbi unutar iste RUN instrukcije (istog reda Dockerfile datoteke) kako bi se smanjio broj slojeva slike koja će nastati, naime svaki red Dockerfile datoteke stvara jedan sloj slike, dakle slika se zapravo sastoji od više slojeva koji funkcioniraju kao jedna cjelina. Prednost takve slojevitosti je zasigurno neovisnost, ukoliko se dogodi potreba za izmjenom jednog retka Dockerfile će napraviti samo nadogradnju na način da će slojeve koji se nalaze ispod sačuvati netaknute, a ponovno sagraditi onaj promijenjeni i slojeve koji dolaze nakon njega. U šestoj liniji povezane su tri naredbe, *apt-get update* služi kako bi se odradilo ažuriranje liste dostupnih paketa, *pip install –upgrade pip* naredba služi kako bi se instalirale nadogradnje sustava za upravljanje paketima napisanim u Python programskom jeziku, *pip install -r potrebno.txt* na samom kraju služi kako bi se instalirale dvije navedene stvari (u ovom slučaju) unutar datoteke *potrebno.txt*. Ove tri naredbe povezane su operatorom && koji služi kako bi se svaka sljedeća naredba izvršila tek ako je ona prije nje uspješno izvršena. WORKDIR instrukcija mijenja trenutni radni direktorij iz *root* (/) direktorija u direktorij naveden kao argument nakon same instrukcije. Putanja može biti apsolutna ili relativna, ukoliko je relativna onda će biti relativna u odnosu na prošlu putanju postavljenu također WORKDIR instrukcijom. Također, ukoliko dani direktoriji u putanji ne postoji, bit će kreiran. Kako su zadnje tri naredbe na neki način vezane skupa će biti i objašnjene.

Dakle, u samom korijenu datotečnog sustava slike prvo se kreira direktorij imena *projekt* u koji se zatim (iz direktorija sa slike 4.1.) kopira web aplikacija i na samom kraju taj isti direktorij postavlja se (WORKDIR naredbom) kao radni direktorij.

Kako je ranije navedeno, ovaj projekt sastojat će se od dva zasebna kontejnera. Kako bi se njihova komunikacija i povezivanje uspješno i lako izveli koristit će se *docker-compose.yml* datoteka. Docker-compose alat je koji preko Docker enignea preuzima i gradi slike, pokreće kontejnere pravilnim redoslijedom i izvršava ispravno povezivanje više kontejnera u cjelinu, sve navedeno definira se u *docker-compose.yml* datoteci. Nastavak yml ili yaml (engl. *Yaml Ain't Markup Language*) predstavlja lako čitljiv format za serijalizaciju podataka.



Sl. 4.4. Prikaz sadržaja docker-compose.yml datoteke

Slika 4.4. prikazuje na koji način su konfigurirana dva servisa (engl. *services*) odnosno dva buduća kontejnera. *Volumes* i *services* (uz *networks* koja se u ovom slučaju ne koristi) glavne su kategorije ove datoteke. Unutar *services* kategorije navedena su dva servisa (dva kontejnera) i njihova konfiguracija. Prvi imena *postgres* definiran je određenim informacijama koje su u odnosu „ključ-vrijednost“, isto imenovanje primjenjuje se i na drugi *web* kontejner. Ispod imena servisa (kontejnera) može se nalaziti više ključeva, svaki servis (kontejner) izričito mora sadržavati *image* ili *build* ključ kako bi se znalo na kojoj slici će se temeljiti jednom kada bude pokrenut. Nakon jednog od ova dva ključa dodaju se oni ostali.

Unutar sljedećih par rečenica objašnjena je konfiguracija unutar ove datoteke prikazane na slici 4.4. Pod kategorijom *volumes* navedena je jedna jedinica za pohranu imena *pgpodaci* (što bi punim imenom predstavljalo „postgrespodaci“). Kako bi se ova jedinica kreirala dovoljno joj je navesti samo ime, u većini slučajeva može se pronaći unutar standardnog Docker direktorija s putanjom */var/lib/docker/volumes*. U kategoriji *services* prvo je definiran ranije spomenuti servis (budući kontejner) *postgres*, njegov prvi ključ „image“ govori da će se temeljiti na slici postgres, najnovije dostupne verzije. Sljedeći ključ „environment“ sadrži varijable okruženja (engl. *environment variables*) koje služe kako bi se po potrebi mijenjala konfiguracija datoteke, u ovom slučaju to je izvršeno direktno unutar datoteke, no postoji i mogućnost kasnije izmjene u komandnoj liniji dodavanjem *-e* opcije u naredbu. Kako bi nešto postalo varijabla okruženja vrijednost je potrebno zapisati u posebnom formatu, najčešće stavljanjem „$“ znaka ispred vrijednosti. U ovom slučaju unutar servisa (kontejnera) postavljaju se tri vrijednosti, vrijednost za bazu, korisnika i lozinku koje će se uz još par vrijednosti također koristiti u postavkama Django aplikacije kako bi aplikacija imala podatke o bazi podataka koju treba koristiti.

settings.py ×

app > settings.py

```
75
76  #DATABASES = {
77  #    'default': {
78  #        'ENGINE': 'django.db.backends.sqlite3',
79  #        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
80  #    }
81  #}
82
83  DATABASES = {
84      'default': {
85          'ENGINE': 'django.db.backends.postgresql_psycopg2',
86          'NAME': 'hello',
87          'USER': 'hello',
88          'PASSWORD': 'hello',
89          'HOST': 'postgres',
90          'PORT': 5432,
91      }
92  }
93
```

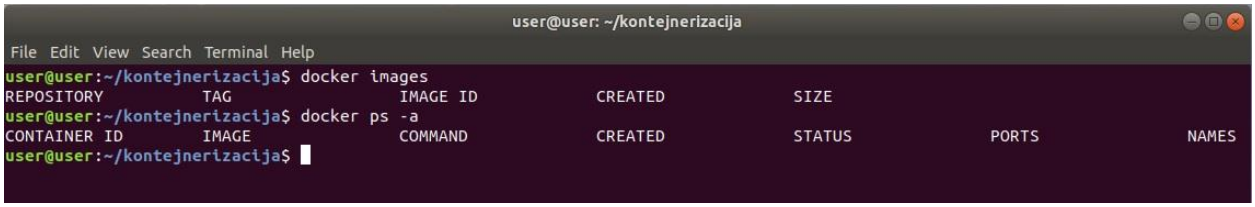
Sl. 4.5. Prikaz starih i novih vrijednosti za postavljenu bazu podataka u Django aplikaciji

Slika 4.5. u gornjem dijelu prikazuje zakomentiranu konfiguraciju standardne baze podataka koja je postavljena unutar datoteke s postavkama Django aplikacije. Ispod zakomentirane konfiguracije prikazane su nove postavke u kojima se definira kako će se od sada koristiti postgres bazu podataka

s psycopg2 adapterom, zatim se prikazuju vrijednosti za pristup koje su postavljene preko varijabli okruženja (unutar *docker-compose.yml* datoteke), a na kraju je definirano ime servisa (kontejnera) u kojem će se nalaziti baza, s pripadajućim portom za pristup.

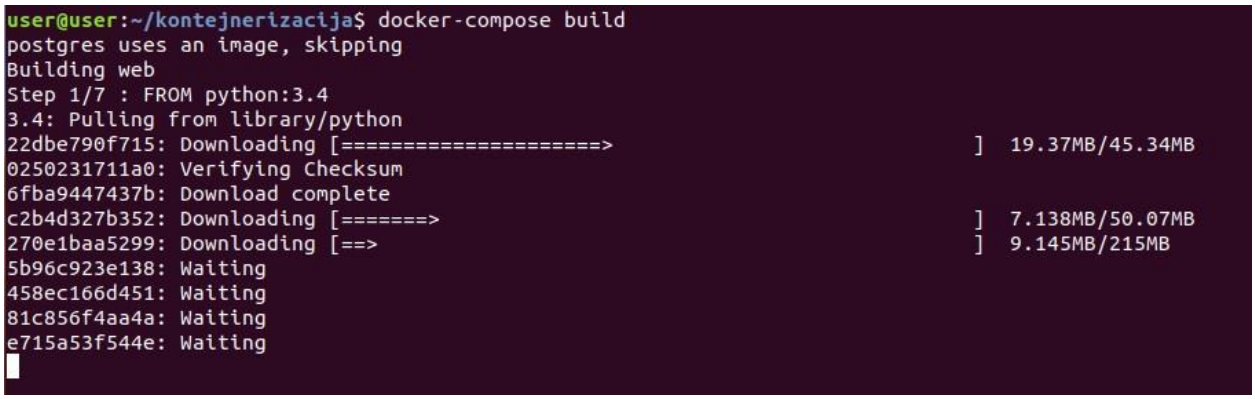
U tekstu koji slijedi nastavlja se opis preostale konfiguracije prikazane na slici 4.4., sljedeći ključ „volumes“ unutar svoje vrijednosti opisuje da će se ranije kreirana jedinica za pohranu *pgpodaci* (koja se nalazi na pravom sustavu) spojiti u direktorij unutar servisa (kontejnera), s putanjom */var/lib/postgresql/data*. Dakle, vrijednost prije dvotočke predstavlja lokaciju jedinice za pohranu na pravom sustavu, a vrijednost nakon dvotočke lokaciju unutar budućeg servisa (kontejnera). Zadnji ključ za servis (kontejner) *postgres* naziva se „ports“ i na port pravog sustava mapira port iz servisa (kontejnera), dakle „-port sustava na kojem se nalazi:port unutar kontejnera“. U ovom slučaju oba su ista stoga ne bi trebalo biti zabune. Sljedeći servis (budući kontejner) koji je definiran unutar kategorije *services* naziva se *web* i sadržavat će Django web aplikaciju s konfiguracijom koja slijedi. Prvi ključ „build“ prikazuje putanju do Dockerfile datoteke, što bi značilo da će se ovaj servis (kontejner) temeljiti na slici koja će nastati iz Dockerfilea. Točka označava da je Dockerfile datoteka u istom direktoriju gdje se i *docker-compose.yml* nalazi. Sljedeći ključ „command“ služi kako bi se pokrenule određene vrijednosti, izvršava se prioritetno i ukoliko je navedeno nekoliko naredbi kao vrijednost tog ključa, izvršit će se samo posljednja, stoga u ovom slučaju naredbe se nalaze unutar *sh -c „...naredbe...“* što označava pozivanje ljuske (engl. *shell*) s naredbama koje se nalaze unutar navodnika. Naredbe unutar navodnika također su odvojene operatorom && jer svaka sljedeća mora ovisiti o rezultatu izvršavanja prošle. Prva naredba koja je navedena služi kako bi se sačekalo 5 sekundi dok se servis (kontejner) koji sadrži bazu podataka uspješno uspostavi, trenutni servis (kontejner) svoje radnje ne može uspješno nastaviti ukoliko onaj s bazom podataka ne funkcionira. Sljedeća je standardna Django naredba za migraciju podataka u bazi, kako se unutar postojeće Django aplikacije postavila nova baza, podatke je potrebno migrirati u nju. Zadnja naredba u ovom lancu od tri naredbe je također standardna Django naredba za pokretanje servera na kojem će se prikazati web aplikacija, uz dodatak 0.0.0.0:8000 koji govori kako se aplikaciji može pristupiti s bilo koje adrese sve dok je port 8000, ovo se koristi jer je mreža unutar servisa (kontejnera) drugačije konfigurirana i stoga je ovo potreban korak kako bi kasnije ostvario željeni pristup, s pravog sustava. Idući ključ „volumes“ služi kako bi se u servis (kontejner) spojila jedinica za pohranu, no ovaj put jedinica neće biti smještena u Dockerovu standardnu lokaciju (*/var/lib/docker/volumes*) nego je korisnički definirana. Lokacija na pravom sustavu je trenutni direktorij u kojem se nalaze sve datoteke prikazane na slici 4.1., a direktorij u servisu (kontejneru) tek će biti kreiran i imat će naziv *pohrana*.

Predzadnji ključ „ports“ na port 8000 u pravom sustavu mapirat će port 8000 iz servisa (kontejnera), ista pravila vrijede kao i za ključ „ports“ u *postgres* servisu (kontejneru), dakle „-port sustava na kojem se nalazi:port unutar kontejnera“. Kasnije će se preko web preglednika s pravog sustava pristupati web aplikaciji, a adresa će biti 127.0.0.1. (predstavljajući lokalni poslužitelj) s portom 8000 koji je mapiran iz servisa (kontejnera). Zadnji ključ koji se naziva „depends_on“ prikazuje ovisnost trenutnog *web* servisa (kontejnera) o onom koji sadrži bazu podataka. Što konkretno docker-compose datoteci daje do znanja da bez obzira koji je servis (kontejner) prvi naveden, ovaj koji sadrži web aplikaciju treba se pokrenuti tek nakon baze podataka. S ovim ključem objašnjene su konfiguracije unutar datoteka, kako bi se one uspješno izvršile i kako bi pokrenuli web aplikaciju kao i bazu podataka, u terminal je potrebno unijeti sljedećih par naredbi. Naredbe koje će se koristiti bit će pozvane preko docker-compose datoteke jer su s njom povezane i preostale dvije datoteke iz direktorija sa slike 4.1.



Sl. 4.6. Izlistanje Docker slika i kontejnera

Slika 4.6. za početak prikazuje pokrenute dvije naredbe, *docker images* i *docker ps -a* kako bi se prikazalo da na sustavu nema Docker slika, niti Docker kontejnera u bilo kojem stanju.



Sl. 4.7. Prva docker-compose naredba

Slika 4.7. prikazuje prvu docker-compose naredbu koja je izvršena, *docker-compose build* naredbu. Ova naredba koristi se za izgradnju slika na kojima će biti temeljeni servisi (kontejneri) unutar *docker-compose.yml* datoteke. Ukoliko se pogleda prvi redak nakon same naredbe, terminal ispisuje poruku kako servis (kontejner) *postgres* već koristi gotovu sliku stoga na njega *build*

naredba nema utjecaja. Nakon toga vidljiv je postupak izgradnje *web* servisa (kontejnera), postupak se temelji na Dockerfileu sa slike 4.2. i vidljivo je kako će slika imati 7 slojeva. Prvi sloj je osnovni sloj i kako se za nju koristi druga slika python:3.4 (a nema je na sustavu), Docker će ju prvo preuzeti s Docker Hub registra, nakon njenog preuzimanja izgradnja će se nastaviti po definiranim Dockerfile instrukcijama.

```
user@user:~/kontejnerizacija$ docker images
REPOSITORY              TAG                 IMAGE ID            CREATED             SIZE
kontejnerizacija_web    latest              aca3f285fc88        14 seconds ago      988MB
python                  3.4                 8c62b065252f        17 months ago       924MB
```

Sl. 4.8. Stanje Docker slika nakon *docker-compose build* naredbe

Nakon izlistavanja Docker slika na slici 4.8. prikazano je kako sada na sustavu postoji slika python:3.4 čije preuzimanje je spomenuto u prethodnim rečenicama, te je korištena za drugu „kontejnerizacija_web“ sliku koja će kasnije biti iskorištena za instanciranje servisa (kontejnera) web aplikacije. Docker-compose smatra svaku *docker-compose.yml* datoteku kao projekt, stoga će se kao ime projekta po standardnim postavkama uzimati direktorij u kojem se nalazi sama *docker-compose.yml* datoteka. Navedenu postavku moguće je promijeniti navođenjem *-p* opcije i novog imena projekta nakon nje, prilikom pozivanja *docker-compose build* naredbe.

```
user@user:~/kontejnerizacija$ docker-compose create
WARNING: The create command is deprecated. Use the up command with the --no-start flag instead.
Pulling postgres (postgres:latest)...
latest: Pulling from library/postgres
bf5952930446: Pull complete
9577476abb00: Pull complete
2bd105512d5c: Pull complete
b1cd21c26e81: Pull complete
34a7c86cf8fc: Pull complete
274e7b0c38d5: Pull complete
3e831b350d37: Pull complete
38fa0d496534: Pull complete
c989da35e5c0: Pull complete
26dc6fdd7b2d: Pull complete
3c5032512cf3: Pull complete
26910ececf99: Pull complete
0339413523e8: Pull complete
d61df7db53da: Pull complete
Digest: sha256:9f325740426d14a92f71013796d98a50fe385da64a7c5b6b753d0705add05a21
Status: Downloaded newer image for postgres:latest
Creating kontejnerizacija_postgres_1 ...
Creating kontejnerizacija_postgres_1 ... done
Creating kontejnerizacija_web_1 ...
Creating kontejnerizacija_web_1 ... done
```

Sl. 4.9. Docker-compose naredbe za kreiranje servisa (kontejnera)

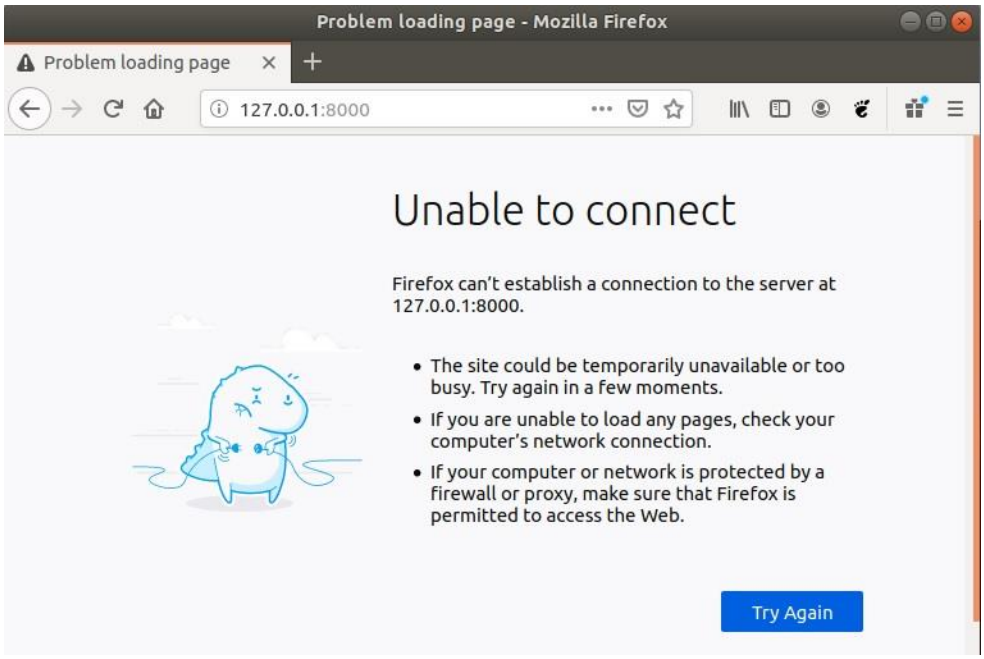
Prema slici 4.9. vidljivo je da nakon korištenja naredbe *docker-compose create* terminal ispisuje upozorenje kako je ova naredba zastarjela, te ukoliko korisnik želi stvoriti servise (kontejnere), bez njihovog pokretanja, trebao bi koristiti *docker-compose up –no-start* čija podnaredba *up*

ukoliko se sama nalazi služi za stvaranje i pokretanje servisa (kontejnera), no u kombinaciji s ovom *–no-start* opcijom izvršilo bi se samo stvaranje, bez pokretanja. Nakon ove obavijesti vidljivo je kako se vrši preuzimanje postgres:latest slike koja će služiti za servis (kontejner) s bazom podataka, a bila je preskočena prilikom pozivanja *docker-compose build* naredbe. Nakon uspješnog preuzimanja slike vidljive su dvije obavijesti kako su kreirani servisi (kontejneri) za web aplikaciju i za bazu podataka. Ukoliko bi izvršili naredbu *docker ps -a* koja ispisuje sve Docker kontejnere na sustavu, rezultat bi bio ekvivalentan slici 4.10.

```
user@user:~/kontejnerizacija$ docker ps -a
CONTAINER ID        IMAGE                  COMMAND                  CREATED             STATUS
33ca382cd42f        kontejnerizacija_web   "sh -c 'sleep 5 && p…"   12 minutes ago      Created
7dbbafda2b46        postgres               "docker-entrypoint.s…"   12 minutes ago      Created
```

Sl. 4.10. Prikaz kreiranih Docker kontejnera

Kako je ispis dugačak, u nastavku su još ispisana dva stupca s informacijama PORT i NAMES, PORT stupac je prazan, dok NAMES ispisuje nazive za ova dva kontejnera. Prvi kontejner naziva se „kontejnerizacija_web_1“, a drugi „kontejnerizacija_postgres_1“. Kako bi se uvjerili da servisi (kontejneri) još nisu pokrenuti i kako nije izvršeno njihovo povezivanje pristupit će se ranije spomenutoj lokalnoj adresi 127.0.0.1:8000 preko web preglednika. Ukoliko bi sve bilo ispravno pokrenuto prikazala bi se web stranica, no u ovom slučaju stanje web preglednika prikazano je slikom 4.11.

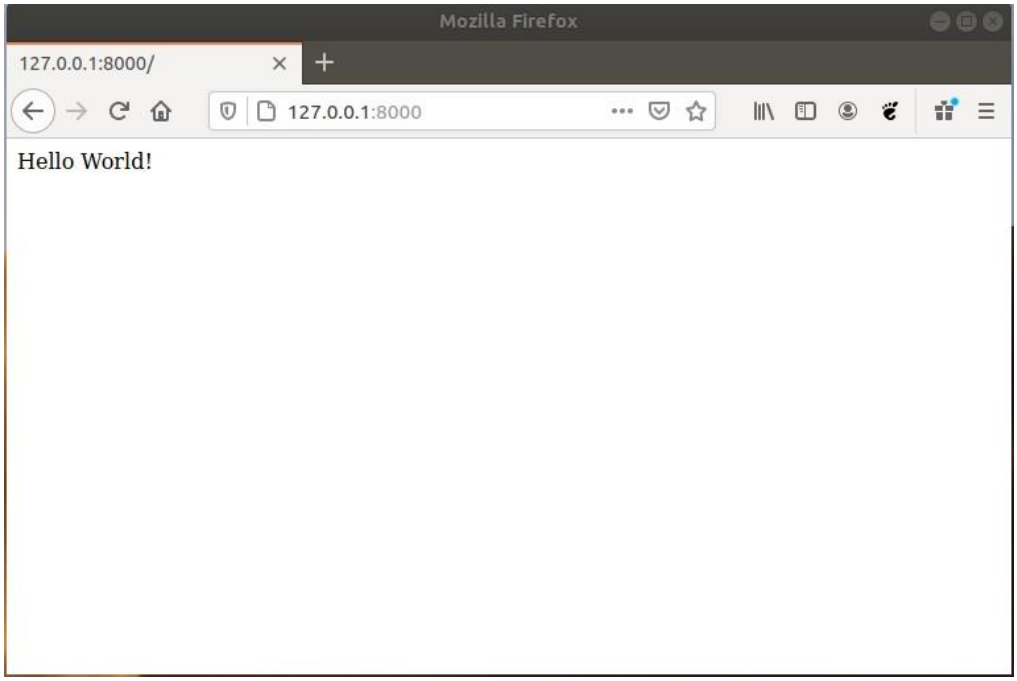


Sl. 4.11. Prikaz stanja web preglednika

```
user@user:~/kontejnerizacija$ docker-compose up
Creating network "kontejnerizacija_default" with the default driver
Starting kontejnerizacija_postgres_1 ...
Starting kontejnerizacija_postgres_1 ... done
Starting kontejnerizacija_web_1 ...
Starting kontejnerizacija_web_1 ... done
Attaching to kontejnerizacija_postgres_1, kontejnerizacija_web_1
```

Sl. 4.12. Docker-compose naredba za osposobljavanje kontejnera i njihove mreže

Slika 4.12. prikazuje zadnju naredbu koja je potrebna kako bi se uspješno pokrenuli servisi (kontejneri) te osposobilo njihovo pravilno komuniciranje. Odmah nakon naredbe prikazuje se obavijest kako je kreirana mreža za ovaj projekt, unutar koje je izvršeno povezivanje servisa (kontejnera). Od sada svaki servis (kontejner), ukoliko je potrebno, pristupanje drugom može izvršiti preko imena servisa definiranog u *docker-compose.yml* datoteci (*postgres* i *web*, u ovom slučaju) [23]. Također, vidljivo je da se izvršilo pokretanje ranije kreiranih kontejnera i njihovo povezivanje. Nakon ovih par linija za svaki od dva postojeća kontejnera nalazi se niz informacija koje opisuju što se točno događa s njima, kontejner s bazom ne ispisuje informacije dok onaj s web aplikacijom ovisno o aktivnosti unutar preglednika ipak prikazuje određen dio informacija. Ukoliko sada osvježimo istu adresu web preglednika, web aplikacija postat će funkcionalna i na ekranu će se prikazati poruka „Hello World!“ prikazana slikom 4.13. Ovim korakom dobili smo potvrdu kako je kontejnerizacija uspješno izvršena.



Sl. 4.13. Prikaz stanja web preglednika

```
user@user:~/kontejnerizacija$ docker ps
CONTAINER ID        IMAGE                  COMMAND                  CREATED
33ca382cd42f        kontejnerizacija_web   "sh -c 'sleep 5 && p…"   About an hour ago
7dbbafda2b46        postgres               "docker-entrypoint.s…"   About an hour ago
```

Sl. 4.14. Prvi dio ispisa pokrenutih Docker kontejnera

```
STATUS              PORTS                     NAMES
Up 29 minutes       0.0.0.0:8000->8000/tcp    kontejnerizacija_web_1
Up 29 minutes       0.0.0.0:5432->5432/tcp    kontejnerizacija_postgres_1
```

Sl. 4.15. Drugi dio ispisa pokrenutih Docker kontejnera

Moguće je napraviti još jednu potvrdu. Naime, ukoliko bi otvorili novu karticu terminala i upisali naredbu *docker ps* koja ispisuje pokrenute kontejnere, kao rezultat dobili bi ispis koji je zbog svoje duljine prikazan u dvije slike 4.14. i 4.15., one potvrđuju da se zaista radi o pokrenutim kontejnerima. U odnosu na prošli ispis, prikazan slikom 4.10., stupac STATUS više nema vrijednost „created“ nego ispisuje vrijeme koliko su kontejneri pokrenuti, a osim toga stupac PORTS dobio je vrijednosti prikazane na slici 4.15.

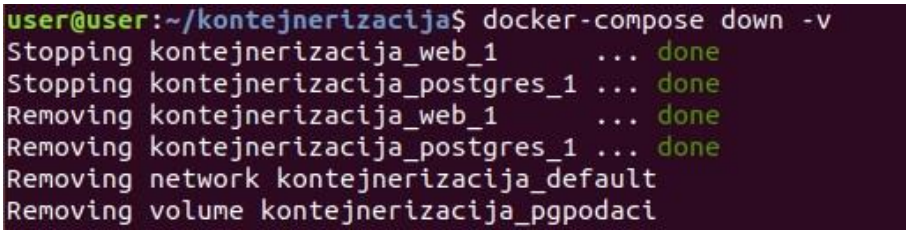
Također, na slici 4.12. prikazan je način pokretanja web stranice koji nije najoptimalniji, događa se zauzimanje trenutnog terminala i ispisuju se informacije spomenute ispod iste slike. Općenita praksa prilikom pokretanja servera (u ovom slučaju web stranice) je izvršavanje naredbe za pokretanje u pozadini, time će se ujedno i riješiti problem zauzimanja trenutnog terminala raznim informacijama o situaciji u kontejnerima. Pokretanje u pozadini izvršit će se vrlo jednostavno, koristit će se ista naredba kao sa slike 4.12., no s dodatkom opcije *-d* (punog naziva *detach*). Prikaz naredbe i poruka nakon nje vidljiv je na slici 4.16.

```
user@user:~/kontejnerizacija$ docker-compose up -d
Creating network "kontejnerizacija_default" with the default driver
Creating kontejnerizacija_postgres_1 ...
Creating kontejnerizacija_postgres_1 ... done
Creating kontejnerizacija_web_1 ...
Creating kontejnerizacija_web_1 ... done
user@user:~/kontejnerizacija$
```

Sl. 4.16. Docker-compose naredba za osposobljavanje kontejnera i njihove mreže, ali u pozadini

Slikom 4.16. prikazuje se gotovo identičan ispis onom na slici 4.12., osim što su nakon ispisanih poruka kontejneri pokrenuti u pozadini, a korisnik (za razliku od prošlog puta) dobiva mogućnost za upisivanje novih naredbi u terminal.

Ukoliko se javi potreba za zaustavljanjem servisa (kontejnera), njihovim uklanjanjem, kao i uklanjanjem kreirane mreže za njihovu komunikaciju to će se postići naredbom *docker-compose down* koja je prikazana slikom 4.17.



Sl. 4.17. Docker-compose naredba za zaustavljanje kontejnera, njihovo uklanjanje, uklanjanje mreže i jedinice za pohranu

U ovom konkretnom slučaju u naredbu je dodana i opcija *-v* (punog naziva *volumes*) koja će ukloniti i ranije kreiranu jedinicu za pohranu. Ova naredba je vrlo jaka jer u sebi sažima par drugih. Ako sam korisnik nema potrebu za potpunim uklanjanjem svega ranije kreiranog nego je situaciji u kojoj će uskoro opet pokretati servise (kontejnere) možda bi bilo korisnije koristiti kombinacije drugih docker-compose naredbi, primjerice *docker-compose pause* za pauziranje, *docker-compose stop* za zaustavljanje ili *docker-compose rm* za uklanjanje zaustavljenih kontejnera.

5. ZAKLJUČAK

Kao polazni zadatak ovog rada navodi se potreba za navođenjem prednosti, mana i konkretne primjene kontejnerizacije koristeći Linux operacijski sustav. Cijela priča širi se također na virtualizaciju, virtualne strojeve, te usporedbu virtualnih strojeva s kontejnerima. Primjenom virtualizacije uvelike će se doprinijeti boljoj iskoristivosti postojećih resursa, bez posebnih troškova.

Za samu srž korištenja virtualnih strojeva i uporabe kontejnera smatra se upravo ista ta potreba za boljom iskoristivosti, uvjet za odabir jedne od ove dvije opcije leži u situaciji za koju će se rješenje primjenjivati. Ukoliko se javi potreba za robusnijim rješenjem koji u sebi sadrži sve funkcionalnosti jednog operacijskog sustava onda se s virtualnim strojem zasigurno ne može pogriješiti, no ukoliko se javlja potreba za rješenjem koje je manje zahtjevno u vidu resursa onda će se kontejneri pokazati kao bolja opcija.

Upravo odsutnost operacijskog sustava u kontejnerima temeljna je razlika i od te činjenice polazi velik broj razlika prilikom usporedbe ova dva rješenja. Razlika koja se često navodi je prostor, a nakon njega brzina pokretanja, pa onda i lakoća s kojom se svako od ova dva rješenja mogu prenositi.

Kako bi se pokazale funkcionalnosti kontejnera i sam postupak kontejnerizacije, opisan je jedan jednostavniji slučaj u kojem se koristi kontejnerizacijska tehnologija Docker. Naime, korišten je Dockerfile kako bi se izgradila slika koja će sadržavati web aplikaciju, iz nje će se stvoriti jedan kontejner. U drugom kontejneru nalazit će se baza podataka, a komunikacija između ta dva kontejnera postići će se koristeći docker-compose datoteku. U ovom primjeru već se mogu vidjeti određene mogućnosti kontejnera koje korisnicima uvelike olakšavaju uspostavljanje aplikacije.

LITERATURA

[1] Red Hat, Understanding virtualization, (https://www.redhat.com/en/topics/virtualization), pristup ostvaren 8.4.2020.

[2] Red Hat, What is virtualization?, (https://www.redhat.com/en/topics/virtualization/what-is-virtualization), pristup ostvaren 8.4.2020.

[3] ISPsystem, A brief history of virtualization, or why do we divide something at all, (https://www.ispsystem.com/news/brief-history-of-virtualization), pristup ostvaren 10.4.2020.

[4] accuwebhosting, Technical Overview of Virtualization Technologies, (https://www.accuwebhosting.com/blog/technical-overview-of-server-virtualization-technologies), pristup ostvaren 29.6.2020.

[5] IBM, Virtualization, (https://www.ibm.com/cloud/learn/virtualization-a-complete-guide), pristup ostvaren 30.6.2020.

[6] dummies, Virtualization: Understanding the Hyper-V Hypervisor, (https://www.dummies.com/programming/networking/virtualization-understanding-the-hyper-v-hypervisor), pristup ostvaren 1.7.2020.

[7] Red Hat, What is KVM, (https://www.redhat.com/en/topics/virtualization/what-is-KVM), pristup ostvaren 1.7.2020.

[8] Wikipedia, VirtualBox, (https://en.wikipedia.org/wiki/VirtualBox), pristup ostvaren 1.7.2020.

[9] Wikipedia, VMWare Workstation, (https://en.wikipedia.org/wiki/VMware_Workstation), pristup ostvaren 1.7.2020.

[10] Medium, Type 1 and Type 2 Hypervisors: What Makes Them Different, (https://medium.com/teamresellerclub/type-1-and-type-2-hypervisors-what-makes-them-different-6a1755d6ae2c), pristup ostvaren 2.7.2020.

[11] Red Hat, What is a virtual machine?, (https://www.redhat.com/en/topics/virtualization/what-is-a-virtual-machine), pristup ostvaren 3.7.2020.

[12] IBM, Virtual Machines (VMs), (https://www.ibm.com/cloud/learn/virtual-machines), pristup ostvaren 3.7.2020.

[13] medium, Virtualization and Hypervisors, (https://medium.com/@devanshagarwal121/virtualization-and-hypervisors-9c4c8f4ab27d), pristup ostvaren 3.7.2020.

[14] Wikipedia, Virtualization, (https://en.wikipedia.org/wiki/Virtualization), pristup ostvaren 3.7.2020.

[15] vmware, Virtual Machine, (https://www.vmware.com/topics/glossary/content/virtual-machine), pristup ostvaren 3.7.2020.

[16] IBM, Containers, (https://www.ibm.com/cloud/learn/containers?mhsrc=ibmsearch_a&mhq=containers), pristup ostvaren 9.7.2020.

[17] Linux Jorunal, Everything You Need to Know about Linux Containers, Part II: Working with Linux Containers (LXC), (https://www.linuxjournal.com/content/everything-you-need-know-about-linux-containers-part-ii-working-linux-containers-lxc), pristup ostvaren 9.7.2020.

[18] linuxacademy, The History of Container Technology, (https://linuxacademy.com/blog/linux-academy/history-of-container-technology), pristup ostvaren 9.7.2020.

[19] man7.org, cgroups(7) – Linux manual page, (https://man7.org/linux/man-pages/man7/cgroups.7.html), pristup ostvaren 9.7.2020.

[20] man7.org, namespaces(7) – Linux manual page, (https://man7.org/linux/man-pages/man7/namespaces.7.html), pristup ostvaren 10.7.2020.

[21] medium, Linux Namespaces, (https://medium.com/@teddyking/linux-namespaces-850489d3ccf), pristup ostvaren 10.7.2020.

[22] docker docs, Docker overview, (https://docs.docker.com/get-started/overview), pristup ostvaren 10.7.2020.

[23] J. S. Chelladhurai, V. Singh, P. Raj, Learning Docker, Second Edition, Packt Publishing, Birmingham – Mumbai, 2017.

[24] stackoverflow, Why does Docker need a Union File System, (https://stackoverflow.com/questions/32775594/why-does-docker-need-a-union-file-system), pristup ostvaren 10.7.2020.

[25] UpGuard, LXC vs Docker, Why Docker is Better, (https://www.upguard.com/blog/docker-vs-lxc), pristup ostvaren 11.7.2020.

[26] Softpanorama, Slightly Skeptical View on Solaris Zones, (http://www.softpanorama.org/Solaris/Virtualization/zones.shtml), pristup ostvaren 11.7.2020.

[27] wikipedia, Solaris Containers, (https://en.wikipedia.org/wiki/Solaris_Containers), pristup ostvaren 11.7.2020.

[28] channelfutures, Docker Downsides: Container Cons to Consider before Adopting Docker, (https://www.channelfutures.com/open-source/docker-downsides-container-cons-to-consider-before-adopting-docker), pristup ostvaren 11.7.2020.

[29] Red Hat, Containers vs VMs, (https://www.redhat.com/en/topics/containers/containers-vs-vms), pristup ostvaren 13.8.2020.

[30] Datamation, Containers vs. Virtual Machines, (https://www.datamation.com/data-center/containers-vs-virtual-machines.html), pristup ostvaren 13.8.2020.

[31] phoenixNAP, Containers Vs Virtual Machines (VMs): What's The Difference?, (https://phoenixnap.com/kb/containers-vs-vms), pristup ostvaren 13.8.2020.

[32] codefresh, Migrating to Docker: How to Plan a Migration and What to Consider, (https://codefresh.io/docker-tutorial/migrating-to-docker), pristup ostvaren 13.8.2020.

SAŽETAK

Glavna svrha ovog završnog rada je opisati pozitivne i negativne strane korištenja kontejnera u Linux okruženju, te prikaz kontejnerizacije u konkretnoj situaciji. Kako bi se upotpunila slika i postiglo šire shvaćanje usko vezanih pojmova, u teorijskom dijelu rad opisuje virtualizaciju i koncepte koji se mogu uz nju vezati. Također, osim virtualizacije, objašnjavaju se virtualni strojevi, te koja je tehnologija zaslužna za njihovo pravilno funkcioniranje. U nastavku rada pokušavaju se razjasniti konkretne razlike između virtualnih strojeva i kontejnera, kao i u kojoj situaciji će odabir jednog od ova dva rješenja biti prikladniji. Kako bi se prikazale određene kontejnerizacijske mogućnosti, korištena je tehnologija Docker kojom se kontejnerizirala jednostavna aplikacija. Rezultat je funkcionalna cjelina koja se sastoji od dva zasebna kontejnera.

Ključne riječi: Docker, kontejner, kontejnerizacija, virtualizacija, virtualni stroj

ABSTRACT

CONTAINERS IN LINUX OPERATING SYSTEM

The main purpose of this paper is to describe positive and negative sides of using containers in Linux environment, and to show containerization in a specific situation. In order to fulfill the picture and achieve broader understanding of closely related concepts, in theoretical part paper describes virtualization and concepts that can be associated with it. Also, in addition to virtualization, virtual machines are explained, and which technology is responsible for their proper functioning. In the following part of paper, the specific differences between virtual machines and containers are in process of being explained, as well as in which situation the choice for one of these two will be more appropriate. To demonstrate certain containerization capabilities, Docker technology was used to containerize a simple application. The result is a functional unit consisting of two separate containers.

Keywords: container, containerization, Docker, virtualization, virtual machine

ŽIVOTOPIS

Karlo Kovačević rođen je 7.1.1999. u Virovitici. Osnovnu školu započeo je 2005. godine u područnoj školi u Crncu, istu je završio 2013. godine u Zdencima. Iste godine upisuje opću gimnaziju u Orahovici, koju završava 2017. godine. U jesen iste godine upisuje stručni studij informatike na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija u Osijeku, na kojem je još student u vrijeme pisanja ovog završnog rada.

Potpis:

_____

PRILOG 1 – SADRŽAJ DOCKERFILE DATOTEKE

```
FROM python:3.4

MAINTAINER Karlo

COPY ./potrebno.txt /potrebno.txt
RUN apt-get update && pip install --upgrade pip && pip install -
r potrebno.txt

RUN mkdir /projekt
COPY ./app ./projekt
WORKDIR /projekt
```

PRILOG 2 – SADRŽAJ DOCKER-COMPOSE DATOTEKE

```
version: '3'
volumes:
    pgpodaci:
services:
    postgres:
        image: postgres
        environment:
            POSTGRES_DB: hello
            POSTGRES_USER: hello
            POSTGRES_PASSWORD: hello
        volumes:
            - pgpodaci:/var/lib/postgresql/data
        ports:
            - 5432:5432
    web:
        build: .
        command: sh -c "sleep 5 && python manage.py migrate &&
        python manage.py runserver 0.0.0.0:8000"
        volumes:
            - .:/pohrana
        ports:
            - 8000:8000
        depends_on:
            - postgres
```