**UNIVERZITET U BEOGRADU**

**FAKULTET ORGANIZACIONIH NAUKA**

**ARHITEKTURA RAČUNARA I OPERATIVNI SISTEMI**

**Docker**

**Mentor: Student:**

**Maja Miljanić, Lazar Stojanović 164/2022**

**Saradnik u nastavi**

**Beograd, 2024. godine**

SADRŽAJ

[1. UVOD 1](#_Toc174201025)

[2. Docker 2](#_Toc174201026)

[2.1. Docker Engine 3](#_Toc174201027)

[2.2. Docker Desktop 3](#_Toc174201028)

[2.3. Docker Hub 3](#_Toc174201029)

[2.4. Docker Compose 3](#_Toc174201030)

[3. Komponente docker-a: 4](#_Toc174201031)

[3.1. Slike (Images) 4](#_Toc174201032)

[3.2. Kontejneri (Containers) 5](#_Toc174201033)

[3.3. Volumeni (Volumes) 6](#_Toc174201034)

[3.4. Mreža (network) 7](#_Toc174201035)

[4. Docker i Virtuelne mašine 8](#_Toc174201036)

[5. ZAKLJUČAK 10](#_Toc174201037)

[LITERATURA 11](#_Toc174201038)

# UVOD

Linux kontejneri predstavljaju kolekciju jednog ili više procesa koji su izolovani od ostatka sistema. Oni funkcionišu kao standardizovani softverski paket koji sadrži sve što je potrebno za pokretanje određenog procesa, uključujući *runtime* i pakete. Fajlovi potrebni za pokretanje kontejnera nalaze se u *slici (image*) koju smo kreirali ili preuzeli, što omogućava konzistentnost i laku deljivost kontejnera na više mašina i okruženja. Ova osobina čini kontejnere jednostavnijim za korišćenje u poređenju sa tradicionalnim metodama replikacije okruženja i paketa.

Postoje tri glavna razloga za upotrebu Linux kontejnera. Prvi razlog je olakšavanje rada na projektima gde učestvuje više programera s različitim razvojnim okruženjima. U tom slučaju, kontejneri omogućavaju da svaki programer radi u istom okruženju, bez potrebe da instalira specifične alate i pakete na svom računaru. Drugi razlog je rad na više projekata koji zahtevaju različite verzije softvera. Kontejneri omogućavaju izolaciju tih projekata i sprečavaju sukobe između različitih verzija. Treći razlog je razlika između razvojne i produkcijske sredine. Kontejneri osiguravaju da aplikacija funkcioniše na isti način u oba okruženja, čime se smanjuju potencijalni problemi prilikom prelaska iz razvoja u produkciju. Iako su ovo tri najčešća razloga za upotrebu kontejnera, oni se mogu koristiti i u drugim situacijama kada je potrebno izolovati aplikaciju od ostatka sistema.

U Linux-u, kontejneri su mogući zahvaljujući ključnim komponentama i tehnologijama koje pružaju izolaciju i upravljanje resursima. Njihova svrha proizašla je iz potrebe za boljom iskorišćenošću hardverskih resursa računara, što je bio prvi korak ka uspešnoj virtuelizaciji. Virtuelizacija omogućava izolaciju i limitiranje resursa koje određeni program koristi, ali tehnologije iz 2000-ih zahtevale su rešenje koje je brže i manje. Umesto repliciranja celokupnog operativnog sistema na svakoj virtuelnoj mašini, pojavila se ideja o limitiranju procesa unutar jednog operativnog sistema.

Prvi korak ka ovoj ideji bio je uvođenje *chroot* funkcionalnosti u Linux-u, koja procesu omogućava pristup samo određenom delu fajl sistema. Nakon toga, *namespaces* su uvedeni kao mehanizam za izolaciju različitih sistemskih resursa, pružajući potrebnu izolaciju za kontejnere. Postoji više tipova namespaces-a, od kojih svaki izoluje različite aspekte sistema, kao što su PID-ovi, mreža, korisnici, i drugi.

*Control groups (cgroups)* predstavljaju funkcionalnost kernela koja kontroliše i ograničava resurse koje koriste procesi ili grupe procesa. *Systemd*, koji je inicijalizacioni sistem, koristi *cgroups* za pružanje bolje kontrole nad procesima, omogućavajući precizno upravljanje resursima unutar sistema.

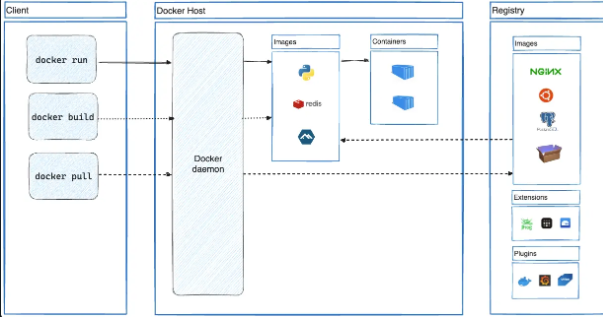
Na kraju, *Union filesystems* pružaju osnovu za kontejnere tako što omogućavaju stvaranje *slojeva (layers)* koje se mogu kombinovati, čime se omogućava deljenje i ponovno korišćenje fajlova između kontejnera. Ova kombinacija tehnologija omogućava Linux kontejnerima da budu efikasni, brzi i laki za korišćenje, čineći ih idealnim rešenjem za modernu softversku infrastrukturu.

# Docker

Docker predstavlja tehnologiju kontejnera i platformu za razvoj, distribuciju i pokretanje aplikacija. Omogućava izolaciju aplikacija tokom razvoja i izvođenja, čime se odvaja od infrastrukture na kojoj se pokreće. Takođe, Docker omogućava upravljanje infrastrukturom na način koji bi inače bio moguć i kroz tradicionalne metode.

Docker koristi klijent-server arhitekturu. *Docker klijent* komunicira sa *Docker daemon-om*, koji izvršava sve potrebne operacije za pokretanje i upravljanje kontejnerima. *Docker klijent i daemon* mogu biti na istom sistemu, ali je moguće da budu distribuirani i povezani preko mreže. Komunikacija između *Docker klijenta* i *daemon-a* odvija se putem REST API-ja, preko UNIX socket-a ili preko mrežnog interfejsa.

Pored osnovnog *Docker klijenta*, postoji i *Docker Compose*. *Docker Compose* je alat koji omogućava definisanje i upravljanje multi-kontejnerskim aplikacijama koristeći konfiguracione fajlove. On pojednostavljuje rad sa aplikacijama koje se sastoje iz više kontejnera, omogućavajući lakše upravljanje i orkestraciju kroz jednostavne komande.



**Slika 1.** Docker arhitektura (Izvor: https://docs.docker.com/guides/docker-overview/)

*Docker daemon* služi za osluškivanje Docker API zahteva i upravljanje ključnim komponentama Docker-a, kao što su slike, kontejneri, mreže i volumeni. Pored toga, *Docker daemon* može komunicirati sa drugim *daemon-ima* kako bi upravljao Docker servisima i omogućio koordinaciju u distribuiranim okruženjima.

*Docker klijent* predstavlja primarni način na koji korisnici interaguju sa Docker-om. Kroz *Docker klijent*, korisnici šalju komande koje koriste Docker API za obavljanje različitih operacija. Docker klijent može komunicirati sa jednim ili više *Docker daemon-a*, omogućavajući fleksibilnost u upravljanju kontejnerima i resursima na različitim sistemima ili mrežama.

## Docker Engine

*Docker Engine* predstavlja celokupan sistem koji omogućava kreiranje, pokretanje i upravljanje kontejnerima. Sastoji se od *Docker daemon-a* i *Docker klijenta*. Pored osnovnih funkcionalnosti, *Docker Engine* uključuje i dodatne alate poput *Docker Compose-a* za upravljanje multi-kontejnerskim aplikacijama i *Docker Swarm*-a za orkestraciju kontejnera u klasterima. Ovi alati omogućavaju jednostavno i efikasno upravljanje aplikacijama u različitim okruženjima, čime *Docker Engine* postaje ključni deo moderne infrastrukture za razvoj i proizvodnju softverskih rešenja.

## Docker Desktop

*Docker Desktop* je aplikacija namenjena operativnim sistemima macOS i Windows koja pruža sve potrebne komponente za pokretanje *Docker Engine*-a na ovim platformama. Kako bi omogućio rad sa Linux kontejnerima na sistemima koji ne koriste Linux kernel, *Docker Desktop* uključuje Linux virtuelnu mašinu (VM) koja se koristi za pokretanje kontejnera. Ova VM obezbeđuje da Linux-specifične funkcionalnosti, kao što su *namespaces* i *cgroups*, budu dostupne i na macOS-u i na Windows-u.

Pored Linux VM-a, *Docker Desktop* uključuje i *Docker Engine*, kao i dodatne alate i komponente za efikasnije upravljanje Docker okruženjem. To uključuje Docker CLI (Command Line Interface) za komande, *Docker Compose* za upravljanje multi-kontejnerskim aplikacijama, i grafički interfejs koji olakšava vizuelno upravljanje kontejnerima, slikama i mrežama.

Docker Desktop automatski upravlja svim potrebnim konfiguracijama, omogućavajući korisnicima da se fokusiraju na razvoj i testiranje aplikacija, bez brige o složenim podešavanjima i konfiguracijama.

## Docker Hub

*Docker Hub* je centralni repozitorijum za skladištenje i deljenje Docker slika. Omogućava korisnicima da objave svoje slike tako da one budu dostupne široj zajednici ili samo određenim korisnicima. Takođe, *Docker Hub* pruža funkcionalnost za preuzimanje slika koje su već dostupne na platformi, što omogućava brzu i jednostavnu distribuciju aplikacija i okruženja. *Docker Hub* podržava automatizovane *build*-ove, integraciju sa drugim alatima i servise za verzionisanje slika, čineći ga ključnim alatom za rad sa Docker kontejnerima.

## Docker Compose

*Docker Compose* je alat koji omogućava definisanje i pokretanje aplikacija koje se sastoje iz više kontejnera. Umesto da se pojedinačne Docker komande upisuju ručno, *Docker Compose* koristi jedan YAML konfiguracioni fajl za specifikaciju svih potrebnih komponenti aplikacije, uključujući slike, kontejnere, mreže i volumene. Ovaj fajl olakšava upravljanje složenim aplikacijama i omogućava jednostavno pokretanje i konfiguraciju kroz jednu komandu (docker-compose up*). Docker Compose* takođe podržava automatsko povezivanje kontejnera i omogućava lakše testiranje i razvoj aplikacija u različitim okruženjima.

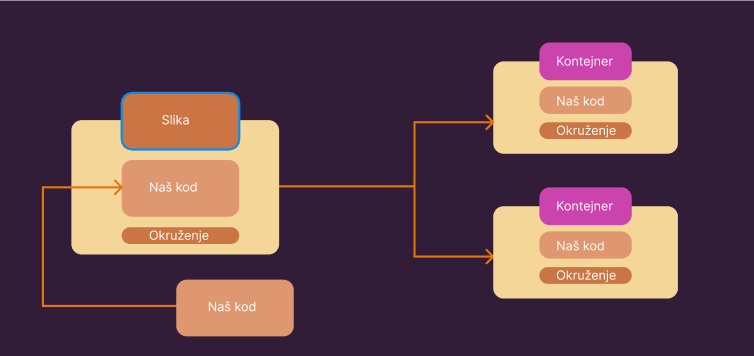
# Komponente docker-a:

Docker funkcioniše kroz ključne komponente koje zajedno omogućavaju upravljanje aplikacijama u kontejnerima. Osnovni gradivni blokovi su slike koje sadrže sve što je potrebno za pokretanje aplikacije, uključujući kod i pakete. Kada se slike pokrenu nastaju aktivni kontejneri koji pružaju izolovano okruženje za izvršavanje aplikacija. Da bi se omogućilo dugotrajno skladištenje i deljenje podataka među kontejnerima, Docker koristi volumene. Ovi volumeni omogućavaju da podaci prežive i nakon što se kontejneri obrišu, čime se očuvavaju važni podaci i konfiguracije. Ove komponente zajedno omogućavaju jednostavno kreiranje, upravljanje i održavanje kontejnerskih aplikacija.

## Slike (Images)

Docker slike sadrže sve što je potrebno za pokretanje određenog kontejnera, uključujući kod, potrebne pakete i runtime okruženje (npr. Node). Mogu se smatrati kao skice ili planovi za pokretanje kontejnera. Na osnovu jedne slike moguće je pokrenuti više kontejnera, pri čemu svaki kontejner predstavlja instancu te slike. Slike su slojevite (*layered*) i svaki sloj je *read-only*, što znači da se ne može menjati. Kontejneri, s druge strane, dodaju još jedan sloj koji je *read-write* i omogućavaju izvršavanje aplikacije u izolovanom okruženju.

Slike se obično kreiraju na osnovu *Dockerfile*-a, gde svaka komanda u *Dockerfile*-u predstavlja jedan sloj slike. Dok je moguće kreiranje sopstvene slike, često se koriste već postojeće slike kao osnova za sopstvene. Ovo omogućava efikasno korišćenje i proširenje postojećih rešenja, čime se pojednostavljuje proces izgradnje i upravljanja slikama.



**Slika 2.** Docker slika

Kroz *Docker klijent* nekoliko najčešćih komandi se koristi za upravljanje Docker slikama:

1. docker pull [IMAGE] - Preuzima sliku iz Docker repozitorijuma (npr. Docker Hub) na lokalni sistem.
2. docker build -t [IMAGE\_NAME]:[TAG] [PATH] - Kreira novu sliku iz Dockerfile-a koji se nalazi na specificiranoj putanji.
3. docker images ili docker image ls - Prikazuje listu svih slika koje su trenutno dostupne na lokalnom sistemu.
4. docker rmi [IMAGE] - Briše navedenu sliku sa lokalnog sistema.
5. docker tag [SOURCE\_IMAGE] [TARGET\_IMAGE] - Dodaje oznaku (tag) postojećoj slici, omogućavajući njeno lakše prepoznavanje i korišćenje.

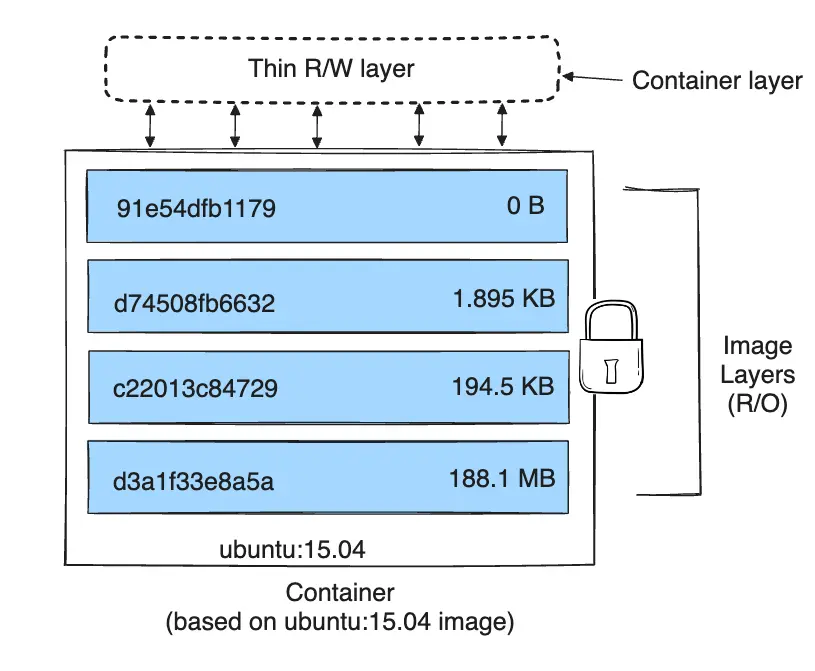
## Kontejneri (Containers)

Kontejneri predstavljaju pokrenute instance određenih slika i funkcionišu kao read-write slojevi koji se dodaju postojećim slikama. Na primer, kontejner može sadržavati dodatni sloj za izvršavanje aplikacije, kao što je *Node* proces. Kontejneri omogućavaju izvršavanje, pauziranje, premestanje ili brisanje aplikacija u izolovanom okruženju. Po prirodi, kontejneri su izolovani od drugih kontejnera i od *host* mašine. Ova izolovanost može se kontrolisati pomoću Docker klijenta kroz izlaganje portova, konfiguraciju mreže ili deljenje volumena.

Kontejneri mogu raditi u *attached* ili *detached* režimu. U *attached* režimu, izlaz kontejnera je priključen na terminal iz kojeg je kontejner pokrenut, što omogućava praćenje njegovih *logova* u realnom vremenu. U *detached* režimu, kontejner se pokreće u pozadini, oslobađajući terminal za druge komande, dok se rad kontejnera nastavlja neprekidno u pozadini.

Kroz *Docker klijent* nekoliko najčešćih komandi se koristi za upravljanje kontejnerima:

1. docker run [OPTIONS] [IMAGE] - Pokreće novi kontejner na osnovu navedene slike sa mogućnošću dodavanja različitih opcija (npr. izlaganje portova, povezivanje sa mrežom).
2. docker ps ili docker container ls - Prikazuje listu svih trenutno pokrenutih kontejnera.
3. docker stop [CONTAINER] - Zaustavlja pokrenuti kontejner sa navedenim ID-om ili imenom.
4. docker start [CONTAINER] - Pokreće zaustavljeni kontejner sa navedenim ID-om ili imenom.
5. docker rm [CONTAINER] - Briše zaustavljeni kontejner sa navedenim ID-om ili imenom.



**Slika 3.** Slojevi kontejnera (Izvor: https://docs.docker.com/storage/storagedriver/)

## Volumeni (Volumes)

Kontejner se može posmatrati kao dodatni sloj koji je se dodaje preko slike, on je svestan fajl sistema koji se nalazi u okviru slike i koristi taj dodatni sloj za formiranje konačnog fajl sistema. Promene koje nastaju u toku rada kontejnera mogu uticati samo na taj dodatni sloj fajl sistema, i one traju sve dok se kontejner ne stopira.

U tu svrhu su nastali volumeni, kako bi bilo moguće da imati perzistentne podatke pri korišćenju kontejnera. Volumeni predstavljaju foldere na *host* mašini koji su mapirani na određene kontejnere. Oni omogućavaju očuvanje podataka čak i kada se kontejner izbriše. Postoje dva tipa volumena: anonimni i imenovani volumeni. U oba slučaja, Docker kreira folder na *host* mašini ali ta tačna lokacija nije poznata korisnicima.

Anonimni volumeni postoje dokle god postoji kontejner. Primer njihovog korišćenja bi bio povezivanje kontejnera sa paketima (node\_modules) jer nije željeno njihovo postojanje bez postojanje kontejnera.

Anonimni volumeni postoje dok god postoji i kontejner. Ne možemo ih kreirati unutar Dockerfile-a, već se kreiraju kada se pokrene kontejner.

Imenovani volumeni se koriste kako bi osigurali perzistentnost podataka čak i nakon brisanja kontejnera (npr. baza podataka) ili za kreiranje vezanih direktorijuma (*bind mount*). Vezani direktorijumi se koriste za ažuriranje rada kontejnera svaki put kada se promene fajlovi koje čine pokrenutu aplikaciju.

Kroz *Docker klijent* nekoliko najčešćih komandi se koristi za upravljanje volumenima:

1. docker volume create [VOLUME\_NAME] - Kreira novi volumen sa navedenim imenom. Ako se ime ne specificira, Docker automatski generiše ime za volumen.
2. docker volume ls - Prikazuje listu svih volumena koji su trenutno dostupni na lokalnom sistemu.
3. docker volume inspect [VOLUME\_NAME] - Prikazuje detaljne informacije o navedenom volumenu, uključujući njegovu putanju na host mašini i konfiguraciju.
4. docker volume rm [VOLUME\_NAME] - Briše navedeni volumen. Volumen se može obrisati samo ako nije povezan sa nijednim kontejnerom.
5. docker volume prune - Briše sve neiskorišćene volumene koji nisu povezani sa bilo kojim kontejnerom, oslobađajući prostor na disku.

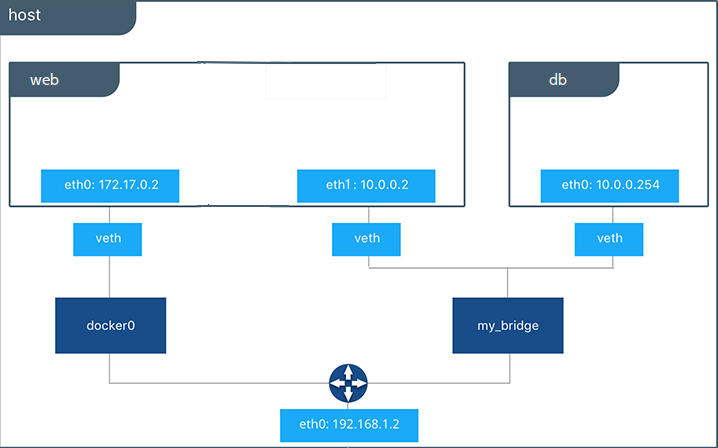
## Mreža (network)

*Docker mreže (networks)* omogućavaju kontejnerima da komuniciraju međusobno i sa spoljnim računarima u kontrolisanom okruženju. Po prirodi, kontejneri su enkapsulirani u sopstvenim mrežama, što znači da su izolovani od drugih kontejnera i spoljnih računara, osim ako se specifično ne omogući komunikacija.

U slučaju kada kontejner treba da komunicira sa *webom*, to može raditi po *defaultu* bez dodatnih konfiguracija. Ako kontejner treba da se poveže sa servisom na lokalnom računaru, koristi se *host.docker.internal* umesto lokalnog *hosta*, što omogućava pristup putem IP adrese *host* mašine. Kada je potrebna komunikacija između kontejnera, koristi se Docker mreža ili se ručno unose IP adrese kontejnera. Docker ne menja naš izvorni kod; umesto toga, detektuje odlazne zahteve i rešava IP adrese za te zahteve.

Kroz *Docker klijent* nekoliko najčešćih komandi se koristi za upravljanje mrežama:

1. docker network create [NETWORK\_NAME] - Kreira novu mrežu sa navedenim imenom. Mogu se specificirati različite opcije za konfiguraciju mreže.
2. docker network ls - Prikazuje listu svih Docker mreža koje su trenutno dostupne na lokalnom sistemu.
3. docker network connect [NETWORK\_NAME] [CONTAINER\_NAME] - Povezuje postojeći kontejner sa navedenom mrežom.
4. docker network disconnect [NETWORK\_NAME] [CONTAINER\_NAME] - Prekida vezu između kontejnera i navedene mreže.



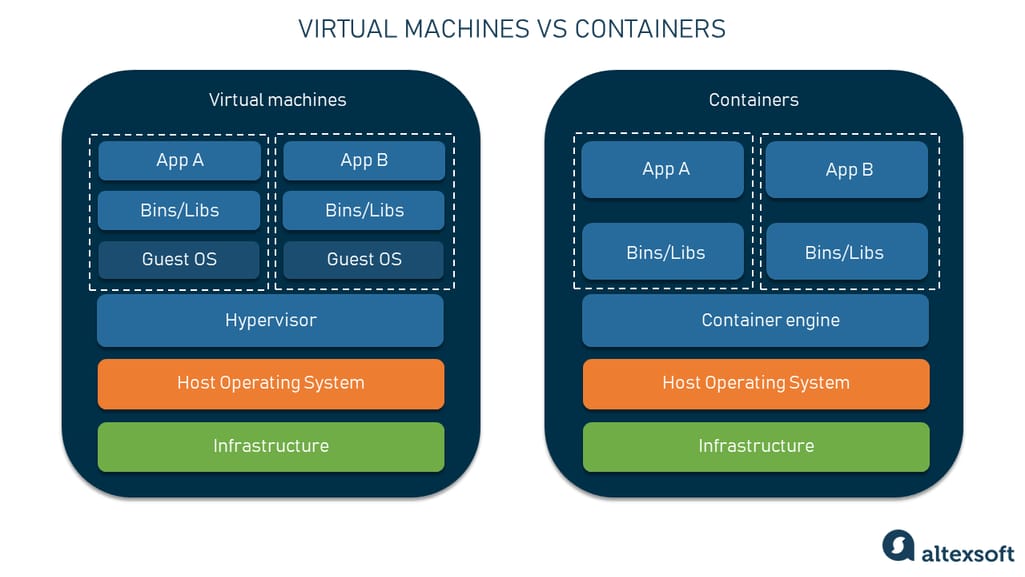
**Slika 4.** Docker mreža

(Izvor: https://docs.docker.com/engine/tutorials/networkingcontainers/)

# Docker i Virtuelne mašine

Virtuelna mašina (VM) je virtualizovani računar koji se sastoji od tri sloja — *host* operativnog sistema, binarnih fajlova i biblioteka, i na kraju aplikacije koja se izvršava na mašini. VM se bazira na hipervizoru poput *KVM-a* ili *Xena* koji služi kao interfejs između *host* operativnog sistema i virtuelne mašine.

Pomoću ovog softvera, cela fizička mašina (CPU, RAM, diskovi, virtuelne mreže, periferni uređaji, itd.) se emulira. Tako, *host* operativni sistem može biti instaliran na ovom virtuelnom hardveru, a zatim se aplikacije mogu instalirati i pokretati na isti način kao i u *host* operativnom sistemu.



**Slika 5.** Virtuelizacija i kontejneri

(Izvor: https://www.altexsoft.com/blog/docker-pros-and-cons/)

S druge strane, kontejneri koriste isti operativni sistem kao i osnovni *host* računar. Hardver se ne virtualizuje. Samo okruženja za izvršavanje sa programskim bibliotekama i konfiguracijama, kao i aplikacije uključujući njihove podatke, su virtualizovani. Kontejnerski *engine* služi kao interfejs između kontejnera i *host* operativnog sistema i dodeljuje potrebne resurse.

Kontejneri zahtevaju manje resurse *host*-a, kao što su procesor, RAM i prostor za skladištenje, u poređenju sa virtuelnim mašinama. Kontejner se pokreće brže od virtuelne mašine jer nije potrebno da se pokrene kompletan operativni sistem.

Za razliku od virtuelnih mašina, koje emuliraju celokupan hardver, kontejneri koriste deljenje *kernel*-a *host* operativnog sistema, što omogućava bržu i efikasniju upotrebu resursa. Ovo takođe znači da se kontejneri mogu pokrenuti i zaustaviti gotovo momentalno, dok virtuelne mašine zahtevaju značajno više vremena zbog pokretanja celog operativnog sistema. Kontejneri takođe omogućavaju lakše skaliranje i prenosivost aplikacija između različitih okruženja, jer su paketi i konfiguracije obuhvaćene unutar samog kontejnera.

# 5. ZAKLJUČAK

Docker predstavlja revolucionarnu tehnologiju koja je značajno unapredila način na koji razvijamo, distribuiramo i pokrećemo aplikacije. Kroz upotrebu kontejnera, Docker omogućava jednostavno i efikasno upravljanje aplikacijama kroz kreiranje standardizovanih okruženja koja su konzistentna bez obzira na infrastrukturu. Kroz ključne komponente Docker-a kao što su Docker Engine, Docker Desktop, Docker Hub i Docker Compose, ovaj sistem pruža sveobuhvatno rešenje za razvoj i implementaciju aplikacija.

U poređenju sa tradicionalnim virtuelnim mašinama, Docker kontejneri nude značajne prednosti u pogledu brzine pokretanja, upotrebe resursa i skalabilnosti. Dok virtuelne mašine emuliraju celokupan hardver, kontejneri koriste deljenje *kernel*-a *host* operativnog sistema, što omogućava bržu i efikasniju implementaciju aplikacija. Ovo ne samo da smanjuje potrebu za hardverskim resursima, već i poboljšava prenosivost i fleksibilnost aplikacija kroz različita okruženja.

U zaključku, Docker predstavlja ključnu tehnologiju za moderni razvoj softvera, koja olakšava rad sa aplikacijama i unapređuje proces njihove implementacije. Kroz svoje inovativne komponente i efikasne metodologije, Docker pomaže u stvaranju robusnih i skalabilnih rešenja za složene softverske zahteve.

# LITERATURA

**Veb sajtovi**

Docker Documentation. <https://docs.docker.com/>

(pristupljeno: 08.10.2024.)

Red Hat. <https://www.redhat.com/en/topics/containers/whats-a-linux-container>

(pristupljeno: 06.10.2024.)

Tech Target. <https://www.techtarget.com/searchitoperations/answer/What-is-a-Docker-container-vs-an-image>

(pristupljeno: 09.10.2024.)

Altexoft. <https://www.altexsoft.com/blog/docker-pros-and-cons/>

(pristupljeno: 07.10.2024.)

Udemy. <https://www.udemy.com/course/docker-kubernetes-the-practical-guide>

(pristupljeno: 20.09.2024)

Medium.<https://erzeghi.medium.com/yet-another-brief-history-of-container-d-2962eac9679e>

(pristupljeno: 10.09.2024)