Zavod za gluhe in naglušne Ljubljana

Naslov naloge

Postavitev spletnega strežnika (Apache-MySQL-PHP) na RaspberryPi z uporabo Docker kontejnerjev

---

Področje

Računalništvo in Informatika

Vrsta naloge

Raziskovalna (razvojno projektna)

Avtor

Luka GRUDNIK, 4Cr

Mentor

Branko VREČAR

Somentor

Matej MENCIN

Ljubljana, marec 2022

Kazalo vsebine

Povzetek in ključne besede

Raziskovalna naloga vključuje več let trajajoč razvojni projekt. Vključuje poglobljeno učenje Linux operacijskega sistema in uporabo zahtevanih sintaks s katerimi sem lahko implementiral ta projekt, izobraževanje na višji zahtevnostni ravni iz računalniških omrežij, pripravo materialov, izdelavo kode, postavitev sistema in testiranje. Sistem je postavljen na izbranem mikroračunaniku Raspberry Pi.

V nalogi je predstavljeno delovanje virtualizacije in kontejnerjev ter namestitveni postopek in upravljanje spletnega strežnika z uporabo Docker kontejnerjev, ki omogočajo poganjanje različnih programov v virtualiziranem stanju. Podrobneje je predstavljen Docker in Docker-compose ter pet različnih kontejnerjev, ki sem jih uporabili za postavitev spletnega strežnika. Docker deluje kot t. i. hipervizor ter omogoča poganjanje vseh kontejnerjev na gostitelju. Docker-compose omogoča upravljanje z večjimi množicami kontejnerjev. Kontejner Portainer omogoča dodajanje ali ustvarjanje novih kontejnerjev ter spreminjanje, brisanje ali upravljanje že obstoječih kontejnerjev. Množica kontejnerjev Apache, MariaDB in phpMyAdmin delujejo kot spletni strežnik, kontejner Pihole pa omogoča blokiranje oglasov, neprimerne vsebine, spletnih strani.

Ključne besede: kontejnerji, virtualizacija, Docker, Raspberry Pi, Docker-compose.

Zahvala

Na prvem mestu se zahvaljujem mentorju mladih raziskovalcev z dolgoletnimi izkušnjami, učitelju strokovnih modulov na srednji šoli ZGN Ljubljana Branku Vrečarju, ki me je ves čas spodbujal ter vedno našel čas za pogovor in iskanje novih rešitev.

Prav tako se zahvaljujem tudi učitelju strokovnih modulov na srednji šoli ZGN Ljubljana Mateju Mencinu, za vse dodatne strokovne napotke pri nastajanju naloge.

Zahvaljujem se tudi staršema za možnost nakupa vse potrebne opreme.

1. Uvod

Danes se tehnologija »virtualizacije« vse bolj pogosto uporablja v IT sektorjih. Vendar ima lahko trenutni način uporabe virtualizacije, kar nekaj težav. Te težave so lahko npr. neskladnost okolji za razvoj oz. produkcijo, kjer npr. vsako (razvojno) okolje uporablja različne verzije iste programske opreme; ali pa da je potrebno (nesmiselno) virtualizirati celoten operacijski sistem, da se lahko testira ali požene neko aplikacijo. To je samo nekaj težav s tradicionalnim načinom virtualizacije. Zato se povečuje uporaba tudi »novejšega« načina virtualizacije t. i. »kontejnerske rešitve«. Ta način virtualizacije, virtualizira, zgolj program, ter knjižnice potrebne za njegovo delovanje in tako reši veliko problemov trenutnega načina virtualizacije.

Sam sem se s tehnologijo virtualizacije prvič srečal v letu 2018, ko sem pričel eksperimentirati z operacijskim sistemom Linux. Preko uporabe aplikacij kot so »Virtual Box« in »VMware Workstation« ali preko interneta, sem spoznal koncept virtualizacije ter njeno delovanje. Vendar sem v naslednjih dveh letih ugotovil, da ta način virtualizacije ne zadostuje vsem mojim potrebam npr. za »domači rekurzivni DNS strežnik«. In tako sem v iskanju možnih rešitev našel koncept virtualizacije z uporabo kontejnerjev, kjer se virtualizira le aplikacija in knjižnice potrebne za njeno delovanje. Ta »nov način« virtualizacije olajša izdelovanje različnih projektov, ker sedaj namesto da bi potrebovali »cel« računalnik/strežnik za izdelovanje projektov, kot so: domači spletni strežnik, pametni dom ali kakšne druge »naredi si sam projekte« okoli doma (ang. *DIY projects*). Lahko v takih primerih uporabimo tudi majhen računalnik, kot je npr. Raspberry Pi, ki je cenovno ugodnejši in omogoča skoraj isto raven uporabnosti, kot fizični računalnik/strežnik (seveda, v manjših zmogljivostih).

Predvidevam, da bo način virtualizacije z uporabo kontejnerjev, v prihodnosti, imel pozitiven vpliv na IT sektor. Z nižjimi stroški strojne opreme in bolj učinkovito uporabo električne energije bodo brez dvoma zadovoljni tudi ljudje, ki želijo svoj dom spremeniti v pametni dom.

2. Teorija - Osnovni pojmi

V tem poglavju bom predstavil: kaj je virtualizacija in kaj je tehnologija vsebnikov oziroma t. i. »kontejnerjev«. Obe »rešitvi« bom nato tudi primerjal. V nadaljevanju bom predstavil kaj je Docker in na kratko predstavil računalnik RaspberryPi. Poglavje o virtualizaciji in tehnologiji vsebnikov je povzeto po Đukić (2016) in Hrastnik (2019), poglavje o Raspberry Pi je povzeto po Raspberry Pi Foundation (2020) ter poglavje od Docker, ki je povzeto po (Schwarzmüller, 2021).

2. 1. Virtualizacija

Virtualizacija ali t**.** i. *navidezni računalnik (stroj)* je »abstrakcija« računalniških virov.

Z ločenjem strojne in programske opreme lahko dosežemo, da ena strojna oprema služi več virtualnim napravam, ki tečejo vsak v svojem izoliranem okolju.

Pred izumom virtualizacije sta bila programska in strojna oprema tesno sklopljeni. Kot je znano ima v tradicionalnem računalniškem okolju, operacijski sistem in druga programska oprema neposreden dostop do osnovnih računalniških komponent, npr. procesor, pomnilnik, trdi disk ipd. Vse to pa oteži premik in konfiguracijo programske opreme na drugo strojno opremo.

Virtualizacija zahteva uporabo hipervizorja, ki deluje na fizični strojni opremi. *Hipervizor* je programska oprema, ki zagotavlja in skrbi za komunikacijo med fizičnimi in virtualnimi viri, ter ustvarja, ugaša in nadzoruje navidezne računalnike (stroje).

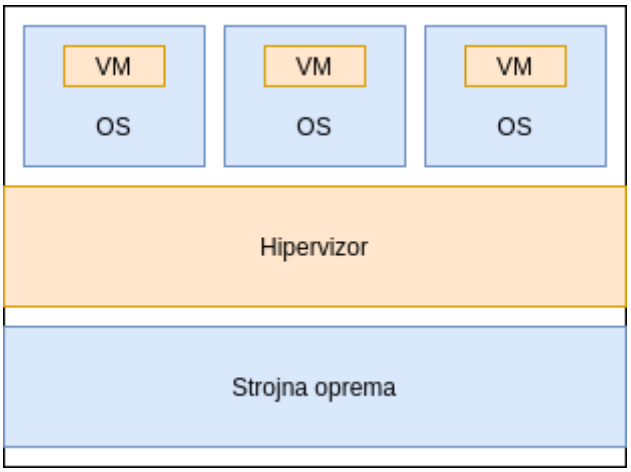
V osnovi poznamo dve vrsti hipervizorjev, to sta:

Hipervizor tipa 1

Hipervizor tipa 1 teče neposredno na gostiteljevi fizični strojni opremi. Ker ima ta hipervizor neposreden dostop do strojne opreme, je to najučinkovitejši pristop. Primeri tega tipa hipervizorjev so »programi«: Microsoft Hyper-V, VMware ESXi, Proxmox in odprtokodni KVM.

Ker hipervizorji tipa 1 delujejo neposredno na fizični strojni opremi so le-ti zelo varni. Varnostne pomanjkljivosti in ranljivosti so izločene, saj med hipervizorjem in strojno opremo ni dodatnega sloja z operacijskim sistemom, kar zagotavlja izolacijo vsakega gostujočega strežnika od zlonamerne programske opreme. V mnogih primerih je na tem v

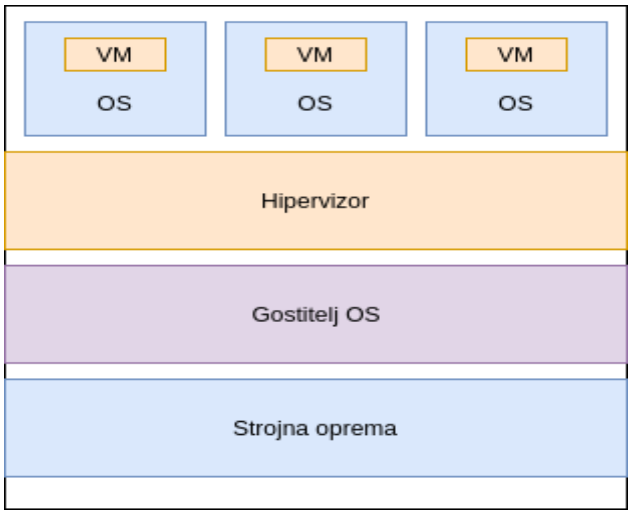
virtualiziranemu sistemu vsaj en virtualni računalnik, ki skrbnikom omogoča nadzorovanje in upravljanje fizičnega sistema z orodji za upravljanje sistema.



SLIKA 1: HIPERVIZOR TIPA 1. POVZETO PO HRASTNIK (2019)

Hipervizor tipa 2

Hipervizor tipa 2 *je* običajno nameščen na obstoječem operacijskem sistemu in se imenuje »gostiteljski hipervizor«. Zanaša se na obstoječi operacijski sistem za upravljanje komunikacije med procesorjem, pomnilnikom, trdim diskom in omrežnimi viri. Primeri hipervizorja tipa 2 so programi: VMware Fusion, Oracle VM VirtualBox in drugi. Ta tip virtualizacije se je uporabljal že od samega začetka virtualizacije, ko so na obstoječe sisteme namestili hipervizor kot dodatno plast programske opreme. Pri uporabi tega tipa je neizogibna zakasnitev, saj gre vsa komunikacija s strojno opremo skozi gostiteljev operacijski sistem. Tako lahko vse morebitne pomanjkljivosti in ranljivosti v gostiteljevem operacijskem sistemu ogrozijo vse virtualne strežnike, ki se izvajajo v gostiteljevem operacijskemu sistemu.



SLIKA 2: HIPERVIZOR TIPA 2. POVZETO PO HRASTNIK (2019).

2. 1. 1. Vrste virtualizacije

Popolna virtualizacija

Pri popolni virtualizaciji, hipervizor, neposredno uporablja komponente fizičnega strežnika. Virtualni strežniki, ki tečejo na istem fizičnem strežniku se med seboj ne vidijo, saj hipervizor poskrbi za izolacijo med virtualnimi strežniki in skrbi za zagotovitev zahtevanih virov posameznim virtualnim strežnikom. To povzroči, da fizični sistem rezervira nekaj procesorske moči za delovanje hipervizorja. Posledično to vpliva na nekoliko zmanjšanje zmogljivosti samega strežnika.

Paravirtualizacija

Tu se vsak virtualni strežnik zaveda drugih virtualnih strežnikov, ki tečejo na istem fizičnem strežniku. Ker vsak virtualni strežniki pozna »potrebe za delovanje« drugih virtualnih strežnikov, hipervizor potrebuje manj procesorske moči za upravljanje virtualnih strežnikov, zato se izboljšajo zmogljivosti samega strežnika.

Strojno podprta virtualizacija

Uporablja strežnikovo strojno opremo kot arhitekturno podporo za upravljanje virtualnih strežnikov. To obliko virtualizacije je prvič uporabilo podjetje IBM leta 1972, ko so raziskovalci odkrili, da se lahko virtualizacijske funkcije veliko učinkoviteje izvajajo na nivoju strojne opreme. To je spodbudilo razširitev ukaznih nizov v procesorjih Intel in AMD. Na ta način je lahko hipervizor enostavno komuniciral s procesorjem in mu neposredno pošiljal

ukazne nize. Tako je procesor prevzel težko delo pri ustvarjanju in vzdrževanju virtualnih strežnikov. S tem so se sistemske obremenitve bistveno zmanjšale, kar pomeni da lahko gostiteljski sistem gosti več virtualnih strežnikov in s tem zagotavlja večjo zmogljivost.

Strojno podprta virtualizacija je najpogostejša uporabljena oblika virtualizacije.

Virtualizacija na nivoju operacijskega sistema

Pri tem tipu virtualizacije ni potrebe po hipervizorju, saj to vlogo zagotavlja operacijski sistem. Pri tej obliki virtualizacije morajo biti operacijski sistemi gostujočih virtualnih strežnikov enaki gostiteljevemu operacijskemu sistemu. Ker virtualizacija na nivoju operacijskega sistema uporablja gostiteljski operacijski sistem kot osnovo za vse neodvisne virtualne strežnike ki tečejo na sistemu, ta način odpravi potrebo po emulaciji gonilnikov. To vodi k boljšemu delovanju in možnosti, da se hkrati poganja več virtualnih strežnikov. Primeri sistemov, ki temeljijo na virtualizaciji operacijskega sistema, so Docker, Solaris Containers in Linux-VServer.

2. 2. Kontejnerji

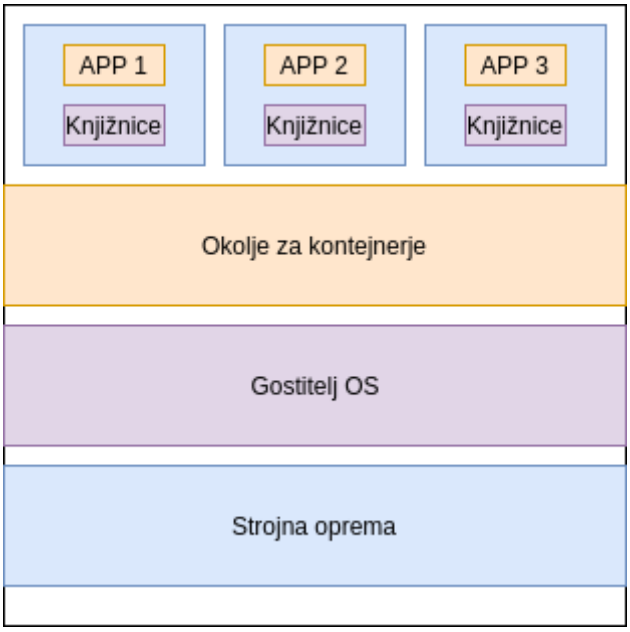
Tehnologija kontejnerjev je skoraj tako stara kot tehnologija virtualnih strojev. Začetki tehnologije kontejnerjev segajo v leto 1979 in je bila prvič uporabljena oz. dodana v Unix 7.

Tehnologija virtualizacije na osnovi arhitekture kontejnerjev je »lahka« alternativa tehnologiji virtualizacije, ki temelji na arhitekturi hipervizorja. Medtem ko hipervizorji delujejo na ravni abstrakcije strojne opreme, je za kontejnerje značilno več izoliranih uporabniških prostorov, ki tečejo na ravni operacijskega sistema in zagotavljajo abstrakcijo neposredno gostujočim procesom. Zato tehnologija virtualizacije s pomočjo kontejnerjev spada v kategorijo virtualizacije na nivoju operacijskega sistema.

Ker kontejnerji izvajajo izolacijo procesov na ravni operacijskega sistema gostiteljske naprave, se izognejo obremenitvam zaradi virtualizirane strojne opreme in gonilnikom navideznih naprav. V vsakem kontejnerju je možno zagnati enega ali več procesov. En sam kontejner se lahko uporabi za poganjanje »majhnega« mikro servisa ali pa tudi za poganjanje večjih aplikacij; pri čimer mora veljati, da vsak posamezni kontejner, vsebuje, vse potrebne izvršljive datoteke, binarne datoteke, knjižnice in nastavitvene datoteke. Ker kontejnerji ne vsebujejo slik operacijskega sistema, so kontejnerji zelo lahki in prenosni.

2. 2. 1. Zgradba in delovanje

Kontejnerji so nameščeni na fizičnem strežniku in njegovem gostiteljskem operacijskem sistemu, to je lahko Windows ali Linux. Vsak kontejner si deli jedro gostiteljskega operacijskega sistema in običajno tudi binarne datoteke in knjižnice. Ker so komponente v skupni rabi samo za branje, so izjemno prostorsko učinkoviti. Porabijo le nekaj megabajtov prostora in se zaženejo v nekaj sekundah, v primerjavi z gigabajti velikosti in minutami za zagon virtualnih strežnikov.



SLIKA 3: ARHITEKTURA KONTEJNERJEV. POVZETO PO HRASTNIK (2019).

2. 2. 2. Prednosti in slabosti »kontejnerjev«

Prednosti »kontejnerjev«:

1. Mobilnost. Kontejnerji so zelo mobilni, saj so neodvisni od platforme, kar pomeni da lahko enostavno in zanesljivo izvajamo aplikacije v različnih okoljih, kot razvojno, produkcijsko itd.

2. Zelo majhna poraba (strojnih) virov. Ker so kontejnerji ekstremno učinkoviti glede porabe virov, pomeni da zasedejo zelo malo prostora in posledično jih lahko na enem trdem disku gostimo zelo veliko.

3. Izolacija. Čeprav se kontejnerji izvajajo na istem strežniku in uporabljajo iste vire, so med sabo izolirani. Kar pomeni, da če en kontejner zaradi napake preneha delovati, to ne bo vplivalo na preostale kontejnerje.

4. Hitrost. Ker so kontejnerji zelo majhni, se zaženejo zelo hitro, saj ne potrebujejo zagona operacijskega sistema.

5. Skaliranje. Velika prednost kontejnerjev je možnost horizontalnega »skaliranja«. To pomeni, da lahko dodamo več enakih kontejnerjev znotraj ene gruče. S pametnim

prilagajanjem lahko drastično zmanjšamo stroške, saj lahko poganjamo le kontejnerje, ki so v danem času pomembni.

6. Operativna enostavnost. Ker kontejnerji izvajajo aplikacijske procese ločeno od osnovnega gostiteljskega operacijskega sistema, pomeni da gostiteljski operacijski sistem ne potrebuje posebne programske opreme za izvajanje aplikacij.

7. Boljša produktivnost. Uporaba kontejnerjev naredi testiranje in odpravljanje napak manj zapleteno in manj zamudno, saj je med lokalnim, testnim in produkcijskim okoljem manj razlik. Tako je enostavno posodabljanje.

Slabosti »kontejnerjev«:

1. Varnostna tveganja. Ker si kontejnerji delijo jedro in druge komponente gostiteljskega operacijskega sistema ter imajo korenski dostop pomeni, da so lahko ogroženi, če je v jedru gostiteljevega sistema ranljivost.

2. Prilagodljivost. Za zagon kontejnerjev z različnimi operacijskimi sistemi je potrebno za vsakega zagnati nov strežnik. Za kompleksne poslovne aplikacije je lahko to resna omejitev.

3. Povezljivost. Namestitev kontejnerjev na dovolj izoliran način in ohranjanje ustrezne omrežne povezave je lahko težavno.

4. Učinkovitost. Kontejnerji porabijo razpoložljive vire učinkoviteje kot virtualni stroji. Zaradi prekrivanja omrežnih povezav, povezovanja med kontejnerji in gostiteljskim sistemom pa niso 100-odstotno učinkoviti.

5. Varnostno kopiranje ali shranjevanje podatkov. Vsi podatki znotraj kontejnerja lahko izginejo, ko se le-ta izklopi. Če želimo podatke shranjevati, je potrebno uporabiti drugačen pristop.

2. 3. Primerjava obeh tehnologij

Tako virtualizacija kot tehnologija vsebnikov imata svoje unikatne prednosti in slabosti. Pri tem obe tehnologiji skušata rešiti isti problem (zmanjšati stroške povezanih z nakupom in vzdrževanjem strojne opreme). Vendar obe tehnologiji to rešujeta na svoj način.

Ker sta si tehnologiji različni, ju je zato najbolje izkoriščati oz. implementirati ali uporabljati komplementarno. S tem se zagotovi, da prednosti ene tehnologije krijejo slabosti druge tehnologije, kar pomeni manj dela s kompenziranjem za slabosti ene ali druge tehnologije.

Spodaj je podana tabela razlik obeh rešitev. Tabela je povzeta po Hrastnik (2019).

Lastnost	**Virtualni strežnik**	**Kontejnerska rešitev**
Gostiteljski operacijski sistem	Vsak virtualni strežnik ima svoj virtualiziran operacijski sistem	Vsi kontejnerji imajo isti operacijski sistem in jedro
Komunikacija	Preko omrežnih naprav	IPC mehanizmi
Varnost	Odvisno od implementacije hipervizorja	Obvezen nadzor dostopa
Zmogljivost	Dosti večja poraba virov zaradi translacije navodil med gostujočim in virtualiziranim operacijskim sistemom	Dosti boljša zmogljivost primerljiva z operacijskim sistemom.
Izolacija	Deljenje datotek in knjižnic med gostujočim in gostiteljskim operacijskim sistemom ni mogoče	Ima možnost deljenja pod direktorijev
Zagonski čas	Nekaj minut	Nekaj sekund
Velikost	Zasede več prostora zaradi potrebe po virtualizaciji dodatnega operacijskega sistema	Zasede manj prostora zaradi deljenja gostujočega operacijskega sistema

2. 4. Tehnologija vsebnikov Docker

Docker je odprtokodna platforma, ki podpira tehnologijo vsebnikov (kontejnerjev) in nam omogoča graditi, dostaviti in poganjati katerokoli aplikacijo kjerkoli. Docker je orodje z ukazno vrstico in je proces oz. storitev, ki teče v ozadju.

Docker sestavljajo tri “jedrne” aplikacije in sicer:

- Docker Engine je aplikacija, ki temelji na arhitekturi odjemalec – strežnik. Sestavljajo jo komponente v ozadju, REST API in odjemalec. Odjemalec v obliki ukazne vrstice uporablja REST API za komunikacijo s procesi v ozadju. Ta »aplikacija« preko prejetih ukazov upravlja z Docker slikami, kontejnerji, omrežjem in diskovnim prostorom.

- Docker Hub je register za hrambo različnih Docker slik.

- Docker Compose je orodje, za definiranje aplikacij, sestavljenih iz različnih Docker kontejnerjev.

SLIKA 4: DOCKER LOGO. PRIDOBLJENO S (WIKIPEDIA CONTRIBUTORS, 2021).

2. 4. 1. Slike (ang. *Images*) in kontejnerji (ang. *Containers*)

Docker slika

Docker slika (ang. *Docker Image*) je »šablona«, ki vsebuje navodila za kreiranje kontejnerjev. Ta slika je pogosto zasnovana na drugi sliki, ki pa ji lahko dodamo posebne prilagoditve. Za osnovo lahko vzamemo sliko z operacijskim sistemom in jo prilagodimo, tako da ji dodamo spletni strežnik in našo aplikacijo. Ob ustvarjanju slike, ji lahko dodamo tudi specifično konfiguracijo za našo aplikacijo. Ustvarjamo lahko lastne slike ali pa uporabimo slike, ki so jih ustvarili že drugi in so objavljene v registrih (npr. na Docker Hub).

Če želimo ustvariti lastno sliko, to storimo tako, da ustvarimo datoteko in uporabimo ustrezno (preprosto) sintakso za definiranje potrebnih korakov za ustvarjanje slike in njeno izvajanje. Vsak ukaz v datoteki je ena plast na sliki. Zaradi tega, da če pride do kakšnih sprememb v datoteki, ko se ta spremeni in se iz tega ponovno zgradi slika, se obnovijo le tiste plasti, ki so se spremenile. Zato so Docker slike tudi tako lahke, majhne in hitre v primerjavi z drugimi tehnologijami za virtualizacijo.

Docker kontejner

Docker kontejner (ang. *Docker Container)* je zagnan primer slike. Ko se kontejner ustvari, se na sliko doda tanka bralno-pisalna plast, tako da se lahko požene več kontejnerjev na eno in isto sliko. Ker vsi kontejnerji delujejo v izolaciji, si ne delijo aplikacijskega stanja ali podatkov.

SLIKA 5: DOCKER SLIKA IN KONTEJNER. POVZETO PO (SCHWARZMÜLLER, 2021).

2. 4. 2. Shranjevanje

Pri uporabi Dockerja se pojavita dva velika problema:

1. **Podatki znotraj kontejnerja se ne obdržijo** - če se kontejner ustavi in odstrani, se izgubijo vsi podatki znotraj kontejnerja.

2. **Kontejner ne more biti v interakciji z gostiteljevim datotečnim sistemom** - če pride do spremembe v gostiteljevi projektni mapi, se to ne pokaže v pognanem kontejnerju. Potrebno je ponovno narediti sliko in pognati nov kontejner.

Omenjena dva problema se reši z »Volumni« (ang. *Volumes)* ali »Vezanimi nosilci« (ang. *Bind Mounts)*:

Volumni. So mape (in datoteke), s katerimi se upravlja na gostiteljevem računalniku in so povezane z mapami/datotekami znotraj kontejnerja. Poznamo dve vrsti volumnov in sicer »anonimne volumne«, to so volumni, ki se ustvarijo in izbrišejo avtomatsko; ali pa »poimenovane volumne«, ki se avtomatsko ne izbrišejo. S tem se podatki lahko tako rekoč delijo med kontejnerji in gostiteljem.

Vezani nosilci. So zelo podobni volumnom, z razliko da je potrebno razvijalcu specificirati pot, ki se jo uporabi za komunikacijo/povezavo med kontejnerjem in gostiteljem. Ta pot mora biti podana absolutno. Vezani nosilci so zelo uporabni za deljenje podatkov s kontejnerjem, ki bi se lahko spremenili med njihovim delovanjem - npr. za deljenje izvorne kode s kontejnerjem za razvijanje. Vezani nosilci naj se ne uporablja, če se želi samo shranjevanje podatkov; v tem primeru naj se uporabi »poimenovani volumen«.

2. 4. 3. Mreženje

Ker se pogosto uporablja več-kontejnerski aplikacije, še posebej kadar se dela z »realnimi aplikacijami«, bodo le ti (kontejnerji) pogosto potrebovali komunikacijo:

1. s svetovnim spletom;
2. z gostiteljevo napravo;
3. eden z drugim (komunikacija med kontejnerji).

Komunikacija s svetovnim spletom je preprosta in samodejna in bo v veliki večini primerov delovala brez potrebe po dodatni konfiguraciji.

Komunikacija z gostiteljevo napravo je prav tako preprosta, ampak ne bo delovalo brez sprememb npr. ukaz »`fetch('localhost:3000/demo').then(...)`« je napačen in ta primer ne bo deloval, ker »localhost« znotraj kontejnerja kaže na kontejner, ne pa na gostiteljevo napravo, ki poganja ta kontejner.

Torej, če želimo da le to deluje pravilno moramo malenkost spremeniti zgornjo kodo in sicer v »`fetch('host.docker.internal:3000/demo').then(...)`«. To je posebni naslov, ki ga docker prevede v IP naslov gostiteljeve naprave.

Komunikacja z drugimi kontejnerji je tudi zelo preprosta. Obstajata dva glavna načina:

- Ročna nastavitev IP naslova kontejnerja
- Z uporabo Docker omrežij (ang. *Docker Networks*) v katera dodamo kontejnerje, ki za katere želimo, da komunicirajo med seboj.

Prva možnost ni tako dobra, saj je potrebno ročno nastavljati IP naslov, za vsak kontejner; slednje pa se lahko spremeni. Druga možnost pa je dosti boljša, ker lahko z Dockerjem ustvariš svoja omrežja z ukazom »`docker network create IME_OMREZJA`« in dodaš več kontejnerjev isto omrežje. Takrat lahko za naslavljanje kontejnerjev uporabiš tudi njihova imena. V teh primerih Docker tudi sam pridobi IP naslov drugih kontejnerjev.

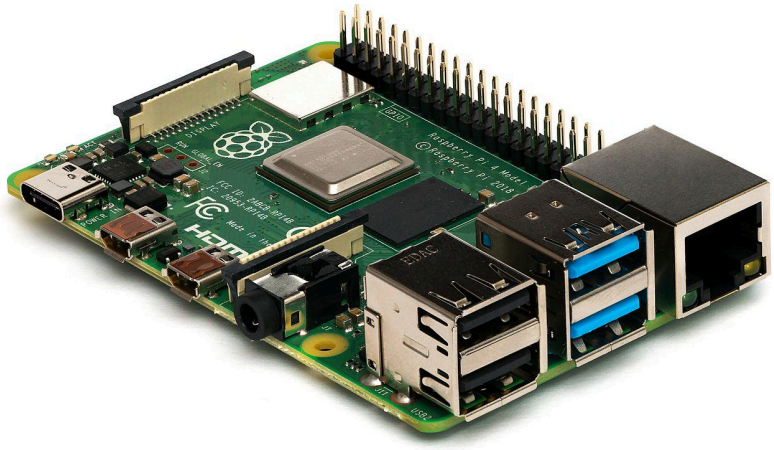
2. 5. Računalnik Raspberry Pi

Računalnik Raspberry Pi je cenovno ugoden računalnik, ki je (približno) v velikosti navadne kreditne kartice. Napravo se lahko priklopi na računalniški monitor ali TV, pri tem pa se uporablja standardno tipkovnico in miško.

Čeprav gre za majhno napravo oz. računalnik, le-ta omogoča, da ljudje vseh starosti raziskujejo svet »računalništva« in se naučijo programirati v preprostih jezikih, kot npr. Python ali Scratch. Ali pa gradijo takšne ali drugačne »hobi« funkcionalne naprave, ki ljudem poenostavijo življenje. Npr. sisteme za avtomatizacijo doma (*ang.* home automation), domače strežnike, ipd.

Sposoben je delati vse kar bi pričakoval od navadnega namiznega računalnika, od brskanja po spletu in predvajanja videoposnetkov v visoki definiciji do izdelovanja razpredelnic in igranja iger.

Obstaja več različnih modelov Raspberry Pi model (Pi Zero, Pi 2/3/4 Model A/A+/B, Pi 400), ki se med seboj razlikujejo glede na strojne lastnosti oz. »sposobnosti«. Sami bomo v zaključni nalogi uporabljali model: Raspberry Pi 4 Model B (4GB Ram).



SLIKA 6: RASPBERRY PI 4 MODEL B. PRIDOBLJENO S (WIKIPEDIA CONTRIBUTORS, 2021).

3 Implementacija

V tem poglavju bom na kratko predstavil, kako sem pripravil RaspberryPi, namestil Docker ter vse potrebne kontejnerje za postavitev spletnega strežnika.

3. 1. Priprava in namestitev RaspberryPi

Na mikro SD kartico se namesti Raspbian (ali Raspberry Pi OS). Nato se v terminalu poženeta ukaza »`sudo apt update`« in »`sudo apt upgrade`«, s čimer posodobimo programsko opremo. Po potrebi se nato v Raspberry Pi omogoči tudi SSH povezava. To storimo z ukazom »`raspi-config`«, v nadaljevanju pa v nastavitvenem meniju omogočimo SSH.

3. 2. Namestitev Docker in Docker Compose

3. 2. 1. Navodila

Najprej se poženeta ukaza: »`curl -fsSL https://get.docker.com -o get-docker.sh`« in »`sudo sh get-docker.sh`«. Slednje namesti Docker Engine. Nato dodamo (privzetega) uporabnika RaspberryPi (to je uporabnik `Pi`) v skupino `docker` z ukazom: »`sudo usermod -aG docker Pi`«.

V nadaljevanju preverimo verzijo Dockerja, z ukazoma: »`docker version`« in »`docker info`«. Na koncu pa še preverimo delovanje Dockerja. V ta namen zaženemo testni kontejner z ukazom: »`docker run hello-world`«.

Kadar »poganjamo« več kontejnerjev hkrati je priporočljivo uporabiti in namestiti Docker Compose. Docker compose je dodatno orodje, ki ga ponuja Docker ter se uporablja pri upravljanju množic kontejnerjev, tako da se jih poveže v skupino (ang. *Stack*). V tej skupini vsi kontejnerji lahko prosto komunicirajo med seboj. Pred namestitvijo Docker Compose je potrebno namestiti še dodatne knjižnice z ukazi: »`sudo apt-get install -y libffi-dev libssl-dev`«, »`sudo apt-get install -y python3 python3-pip`« in »`sudo apt-get remove python-configparser`«.

Nato pa se namesti dejanski Docker Compose s preprostim ukazom: »`sudo pip3 -v install docker-compose`«. Za konec pa še preverimo katero verijo Docker composa,

smo namestili, z ukazom »`docker-compose version`«. Verzijo Docker compose je potrebno vedeti, zaradi pisanja skript.

3. 3. Namestitev kontejnerjev (brez/z uporabo DockSTARTer)

Namestitev vseh danih kontejnerjev poteka na več načinov:

1. DockSTARTER - je skripta, ki jo poženemo v terminalu v njej obkljukamo katere kontejnerje bi želeli namestiti. Ko potrdimo in se vsi kontejnerji avtomatsko namestijo.
2. Z ukazoma »`docker pull IME_SLIKE`« in »`docker run IME_SLIKE`«.
3. Z ukazom »`docker-compose`«.

V naši nalogi sem kontejnerje nameščal z uporabo Docker Compose.

3. 3. 1. Portainer

`Portainer` je kontejner, s katerim lahko upravljamo z drugimi kontejnerji, uporabniki, slikami itn. Pri kontejnerjih lahko konstantno spremljamo stanje, spremljamo uporabo strojne opreme individualnega kontejnerja, vidimo v katerem omrežju so in ga lahko spremenimo, ali pa ustvarimo cela nova omrežja. Prav tako lahko uvažamo nove slike ali jih izvažamo ali pa naredimo popolnoma svoje. Vse to naredi Portainer zelo uporaben kontejner v vsakem Docker sistemu ali projektu.

```
portainer:
  image: "portainer/portainer-ce"
  restart: "always"
  container_name: "portainer"
  ports:
    - "8000:8000"
    - "9000:9000"
  volumes:
    - "/var/run/docker.sock:/var/run/docker.sock"
    - "portainer_data:/data portainer/portainer-ce"
```

SLIKA 7: DOCKER COMPOSE KODA ZA PORTAINER.

3. 3. 2. Apache + MariaDB + phpMyAdmin

`Apache + PHP + MariaDB + phpMyAdmin` so kontejnerji, ki jih skupaj uporabimo za spletni strežnik. Lahko jih uporabljamo individualno, ampak lahko (in vsaj v mojem mnenju je tako lažje) jih tudi povežemo v svoje omrežje z ukazom »`docker-compose`«.

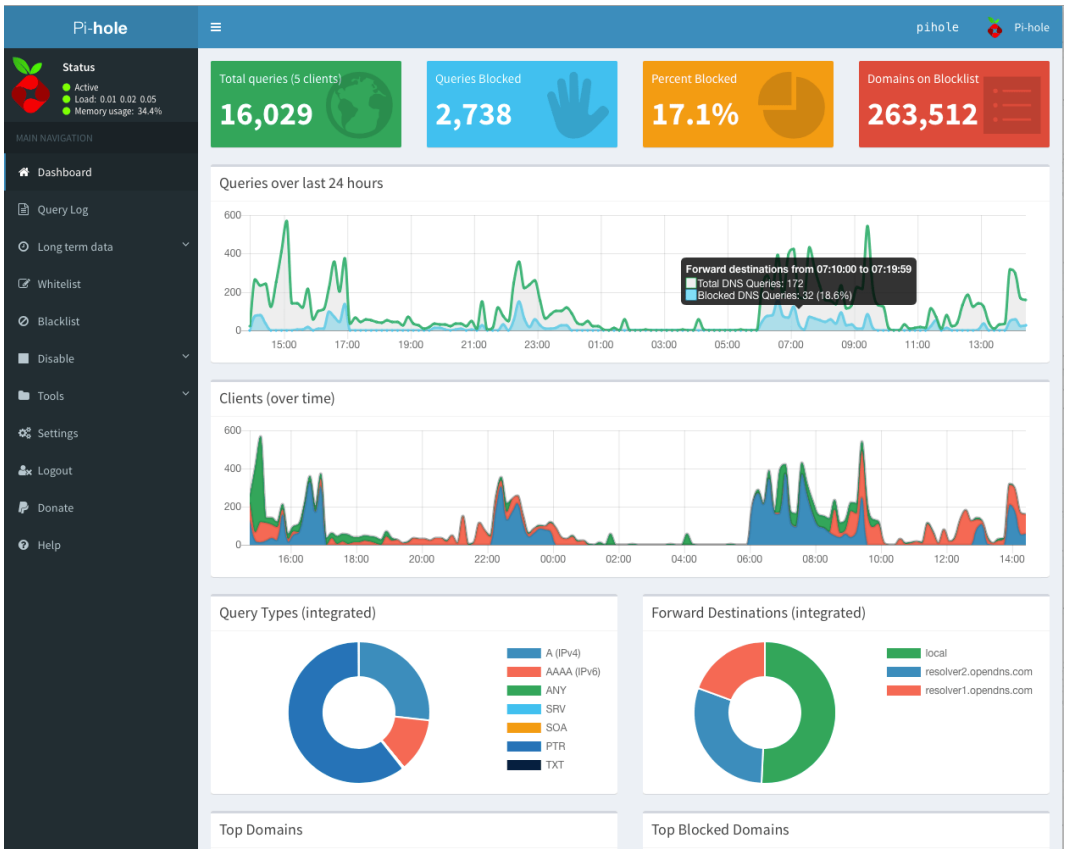
```
web-server:
  build:
    context: "./apache"
    dockerfile: "php.dockerfile"
  container_name: "web"
  restart: "always"
  volumes:
    - "./html/:/var/www/html/"
  ports:
    - "8080:80"

mariadb:
  image: "jsurf/rpi-mariadb"
  container_name: "mariadb"
  restart: "always"
  environment:
    MYSQL_ROOT_PASSWORD: "root"
  volumes:
    - "mysql-data:/var/lib/mysql"
  ports:
    - "3306:3306"

phpmyadmin:
  image: "phpmyadmin"
  container_name: "phpmyadmin"
  restart: "always"
  environment:
    PMA_HOST: "mariadb"
    PMA_USER: "root"
    PMA_PASSWORD: "root"
    PMA_ARBITRARY: "1"
  ports:
    - "5000:80"
```

SLIKA 8: DOCKER COMPOSE KODA ZA SPLETNI STREŽNIK.

3. 3. 3. PiHole



SLIKA 9: PI-HOLE. PRIDOBLJENO S (WIKIPEDIA CONTRIBUTORS, 2021).

`Pihole` je programska oprema, ki deluje preko DNS-ja in omogoča blokiranje oglasov, izbranih spletnih strani, neprimerne vsebine ipd. Tako ga lahko uporabimo, da blokiramo oglase po celotnem omrežju, bodisi na televizijah ali mobilnih aplikacijah. Dodatno, če si zaželimo ga lahko spremenimo v naš DHCP strežnik in tako zagotovimo, da so vse naše naprave avtomatsko zavarovane.

```
pihole:
  image: "pihole/pihole"
  container_name: "pihole"
  ports:
    - "53:53/tcp"
    - "53:53/udp"
    - "80:80/tcp"
    - "443:443/tcp"
  environment:
    TZ: "Europe/Ljubljana"
    WEBPASSWORD: "root"
  volumes:
    - "./etc-pihole/:/etc/pihole/"
    - "./etc-dnsmasq.d/:/etc/dnasmasq.d"
  restart: "unless-stopped"
```

SLIKA 10: DOCKER COMPOSE KODA ZA PIHOLE.

3. 4. Celotna koda

```yaml
version: "3.8" # verzija docker-compose
services:
  web-server:
    build:
      context: "./apache"
      dockerfile: "php.dockerfile"
    container_name: "web"
    restart: "always"
    volumes:
      - "./html/:/var/www/html/"
    ports:
      - "8080:80"
  mariadb:
    image: "jsurf/rpi-mariadb"
    container_name: "mariadb"
    restart: "always"
    environment:
      MYSQL_ROOT_PASSWORD: "root"
    volumes:
      - "mysql-data:/var/lib/mysql"
    ports:
      - "3306:3306"
  phpmyadmin:
    image: "phpmyadmin"
    container_name: "phpmyadmin"
    restart: "always"
    environment:
      PMA_HOST: "mariadb"
      PMA_USER: "root"
      PMA_PASSWORD: "root"
      PMA_ARBITRARY: "1"
    ports:
      - "5000:80"
  portainer:
    image: "portainer/portainer-ce"
    restart: "always"
    container_name: "portainer"
    ports:
      - "8000:8000"
      - "9000:9000"
    volumes:
      - "/var/run/docker.sock:/var/run/docker.sock"
      - "portainer_data:/data portainer/portainer-ce"
```

```
  pihole:
    image: "pihole/pihole"
    container_name: "pihole"
    ports:
      - "53:53/tcp"
      - "53:53/udp"
      - "80:80/tcp"
      - "443:443/tcp"
    environment:
      TZ: "Europe/Ljubljana"
      WEBPASSWORD: "root"
    volumes:
      - "./etc-pihole/:/etc/pihole/"
      - "./etc-dnsmasq.d/:/etc/dnasmasq.d"
    restart: "unless-stopped"

volumes:
  mysql-data:
  portainer_data:
```

SLIKA 11: CELOTNA KODA ZA DOCKER COMPOSE.

4 Zaključek

Z izvedbo naloge in doseženimi rezultati sem v celoti zelo zadovoljen.

Na mikro računalniku Raspberry PI sem izdelal delujoč spletni strežnik, DHCP strežnik, DNS strežnik in upravljanje s kontejnerji. To in več s samo uporabljenimi petimi kontejnerji dokazuje veliko uporabnost Dockerja ter zelo malo porabo sistemskih sredstev, kar daje tako postavljenemu sistemu velik potencial. Delovanje sistema sem še nadgradil z uporabo Docker-compose s katerim sem upravljal večjo množico kontejnerjev.

Mikroračunalnik Raspberry PI omogoča izdelavo odličnih uporabnih projektov in v naslednjem letu za zaključek izobraževanja na tehniku računalništva izdelam še nov projekt, ki bo strežniku dodal še več varnosti, saj je kibernetskih vdorov v omrežja vse več.

5 Literatura

[1] Đukić, M. (2016). Primerjava zmogljivosti virtualizacijskih tehnologij in tehnologij vsebnikov (Magistrsko delo, Univerza v Ljubljani).

[2] Hrastnik, D. (2019). Primerjava virtualnih strežnikov in Docker kontejnerjev za izvlečenje besedila dokumentov na primeru eArhiv Pošte Slovenije (Magistrsko delo, Univerza v Mariboru).

[3] Raspberry Pi Foundation (2020). What is a Raspberry Pi?

[4] Sawant, R. (2020). Installing Docker and Docker Compose on the Raspberry Pi in 5 Simple Steps. DEV Community.

[5] Schwarzmüller M. (2021) Docker & Kubernetes: The Practical Guide [MOOC]. Udemy.

[6] Shovon, B. S. (2020). Set up a LAMP server with Docker – Linux Hint. Linux Hint.

[7] Wikipedia contributors. (2021). Docker (software). In Wikipedia, The Free Encyclopedia.

[8] Wikipedia contributors. (2021). Pi-hole. In Wikipedia, The Free Encyclopedia.

[9] Wikipedia contributors. (2021). Raspberry Pi. In Wikipedia, The Free Encyclopedia.

6 Dodatek

A Kazalo slik

B Slovar Linux ukazov

Linux ukaz*	Pomen
`# raspi-config`	Ukaz odpre okno, kjer lahko nastavimo osnovne nastavitve za Raspberry Pi.
`# sudo apt update`	Pregleda »repositorije« za posodobitve.
`# sudo apt upgrade`	Posodobi vse kar se da.
`# sudo lsblk` `-o UUID,NAME,FSTYPE,SIZE,MOUNTPOINT,LABEL,MODEL`	Ukaz nam izpiše vse particije, ki so trenutno priključeno na Raspberry Pi

* V večini primerov se doda tudi ukaz `sudo`, torej da ukaze izvajamo kot »root«.

C Slovar Docker ukazov

Docker ukazi	Pomen
`# docker build .` `-t NAME:TAGE`	Zgradimo kontejner iz izbrane slike.

`# docker run IMAGE_NAME`
`--name NAME`
`-p LOCAL_PORT:INTERNAL_DOCKER_CONTAINER_PORT`
`-d`
`-it`
`--rm`
`-v /pot/v/kontejnerju`
`-v neko_ime:/pot/v/kontejnerju`
`-v /pot/v/gost/racunalnik:/pot/v/kontejnerju`
`-e IME_ENV=?`
`--env-file .env`
	Se ustvari »kontejner« na podlagi slike.

`# docker stop`	Ustvarimo izvajanje Dockerja.
`# docker ps` `-a`	Izpišemo kontejnerje, ki se trenutno izvajajo.
`# docker images`	Izpišemo slike, ki jih imam lokalno shranjene.
`# docker rm CONTAINER`	Izbrišemo izbrani kontejner.
`# docker rmi IMAGE`	Izbrišemo izbrano sliko.
`# docker pull IMAGE`	Pridobimo sliko na lokalni računalnik.
`# docker volume create VOL_NAME`	Ustvarimo volumen z izbranim imenom.
`# docker volume ls`	Izpišemo vse volumne.
`# docker volume prune`	Izbrišemo volumne, ki jih ne uporabljamo.
`# docker volume rm VOL_NAME`	Izbrišemo izbrani volumen.
`# docker network create IME_OMREZJA` `--driver bridge (ali drugi)`	Ustvarimo omrežje za kontejnerje.