Docker

Opis dockera

- Šta je Docker?
- Kako setovati Docker radno okruženje?
- Docker Cheat Sheet – sve bitne komande na jednom mestu
- Docker run – moj prvi kontejner
- Dockerfile – Moj prvi image
- docker-compose.yml – set instrukcija za stvaraje kontejnera

Šta je Docker?

NAPOMENA: Ovo i sva uputstva koja slede nisu oficijalna i moja su interpretacija originalne dokumentacije i uslova pod kojim se izvode. Tako da mogu i ne moraju da budu ispravna.

Pa da krenemo.

Docker je ako ste developer ili devops vaš najbolji prijatelj. Dosta ljudi ga izbegava zato što im izgleda teško i previše komplikovano za rad, ali istina je potpuno suprotna.

Koliko puta ste se setovali na isti projekat na različitim računarima? I uvek je milion stvari krenulo naopako. Nije bila dobra verzija baze, ili nije bio dobra verzija php-a, ili nešto u operativnom sistemu nije radilo kako treba a vi zato što ste developer možda niste znali šta je, ili ako ste devops niste znali šta je to developer tražio od dependencya. I tako prebacujete se sa računara na računar (sa čoveka na čoveka) i trošite svoje i firmino vreme na setovanje.

Tu docker uskače da vam pomogne. Docker je prostim rečima mikro operativni sistem (ali je u stvari servis). On koristi kernel i neke biblioteke host operativnog sistema, ali je u suštini potpuno nezavistan i osećaj kada radite unutar kontejnera je kao da ste na nekom remote serveru. Možete instalirati sve što vam treba, možete setovati sve što vam treba. I možete bez ikakve griže savesti sve unutra pokvariti i to neće imati nikakve posledice na host operativni sistem.

Docker image

Kako bi podigli **kontejner**, morate imati njegov **image**. On se sastoji od više slojeva koji su u suštini fajlovi koji su generisani ispaljivanjem neke od komandi tokom build procesa. U toku buildanja image-a, svaka komanda koja se unese npr. ” apt update” generiše novi sloj, tj novi fajl. Docker image slojevi koji su identični, koriste se između različitih imidža, npr. Imate image “ubuntu-git” i image “ubuntu-jenkins” oni imaju zajedničku stvar, a to je njihov operativni sistem. Tako da će uštedeti prostor i koristiti isti “sloj” -(eng. layer) . Ovo je primer slojeva za image alpine:latest.

```
IMAGE               CREATED             CREATED BY                                      SIZE
b7b28af77ffe        4 weeks ago         /bin/sh -c #(nop)  CMD ["/bin/sh"]              0B
<missing>           4 weeks ago         /bin/sh -c #(nop) ADD file:0eb5ea35741d23fe3…   5.58MB
```

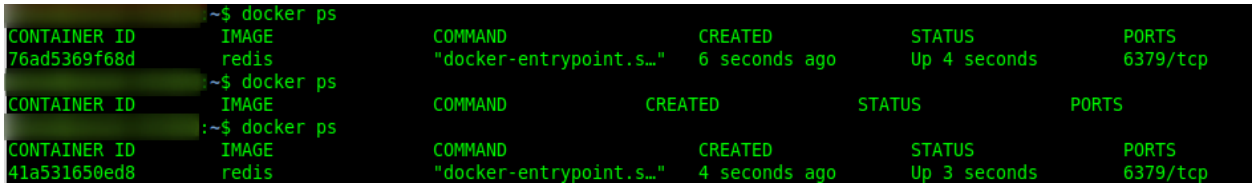
Generalno, vi ovo ne morate znati, ali mislio sam da će vredeti da se pomene kako to izgleda.

Docker container

Sada kada znate šta su imidži, da vam malkice približim i kontejnere. Kada bilo koji image želite da pokrenete, i od njega kreirate servis, stvoraćete kontejner. To je nešto poput virtualne mašine koja

će pokrenuti vaš image. Bitno je da kontejner posmatrate kao PRIVREMENU stvar. Nakon što prestane da bude potreban on će se zaustaviti. A u jednom zaustavljen kontejner više ne možete ući. Primer. Pokrenuo sam image redisa.

Vidite da je dobio ID 76ad5369f68d. Nakon zaustavljanja, proverio sam da li je neki kontejner aktivan. Video da nije, i opet ga pokrenuo.



Vidite novi ID: 41a531650ed8. Isti image. Ista komanda za pokretanje. Dva različita kontejnera. U tome je prava lepota dockera. Ne možete vi dovoljno nešto zeznuti koliko sve možete za 5 sekundi vratiti na početno stanje

Trebate znati i da svaki kontejner ima svoj networking, potpuno je izolovan od ostalih kontejnera, i sem ako mu se to ne naznači, on neće ni znati da nešto tu postoji osim njega.

Docker Registry

Sve imidže koje napravite negde morate čuvati. Za to se koriste registri. Većina koriste onaj glavni docker hub. Besplatan je. Kreirate tamo nalog. Ulogujete se na računaru koristeći komandu “docker login” . Ulogujete se na serveru. I isto kao na gitu, koristite komande push i pull . Prelepo jednostavno.

Sve početne imidže koje ćete ikada koristiti preuzećete sa docker huba. Pa vredelo bi da se upoznate sa sadržajem.

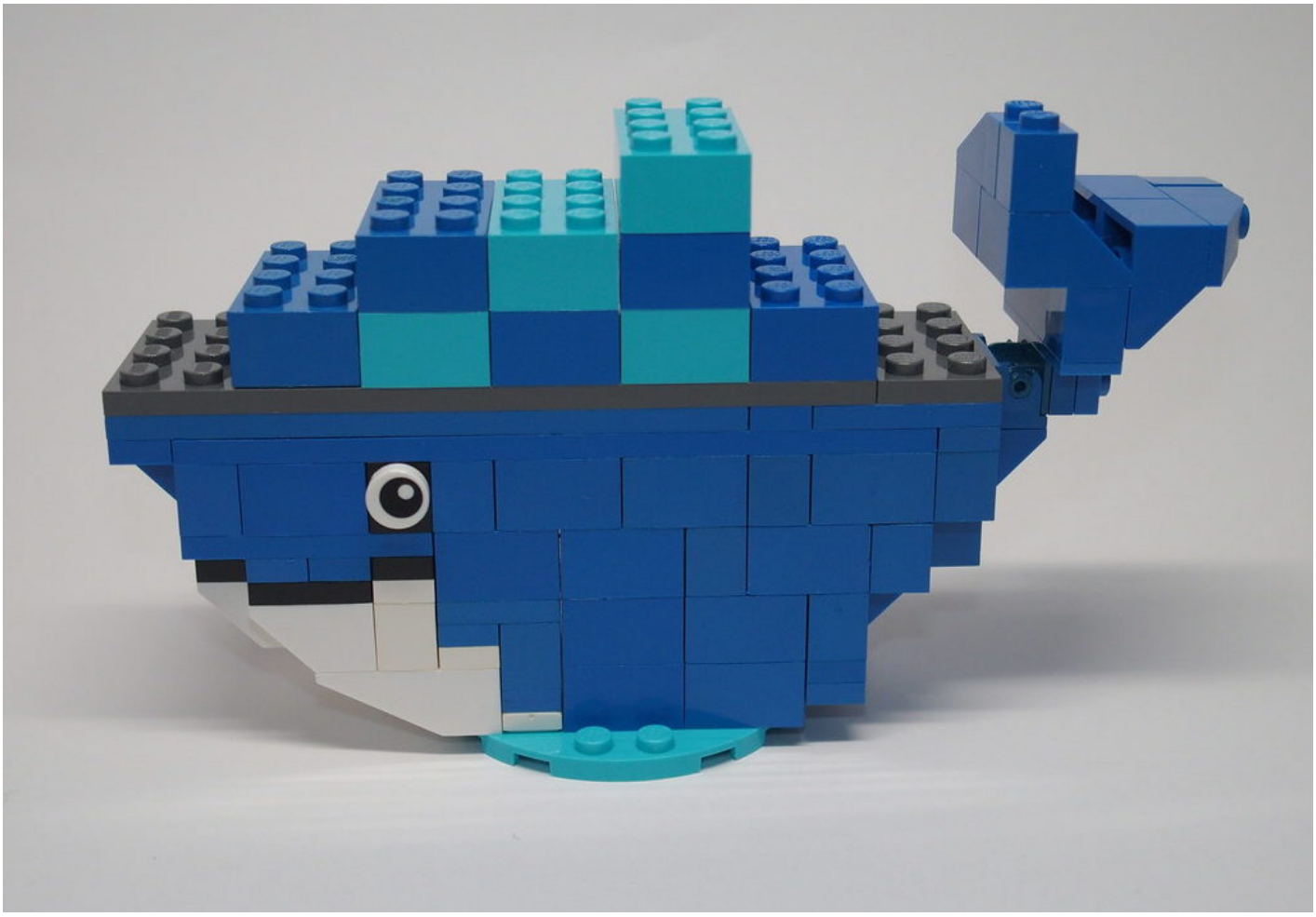
Znate koliko je teško nekada setovati neko parče softvera i koliko biblioteka može nedostajati da to nešto proradi. Znate ono, radi samo na tom operativnom sistemu, sa tim zastarelim bibliotekama kojih više nema nigde. Ali vi ste nakako uspeli sve to da pronađete, podignete i naterali da radi, ako ste to uradili unutar kontejnera možete sačuvati taj kontejner u image. I možda nekom olakšati život tako što ćete taj image deponovati na neki javni registar. Ujedno će i vama biti dostupno za sledeći put.

Možda vama treba nešto, pregledajte docker registar, možda se neko već sreo sa tim problemom i rešio ga za vas.

Za sada, od mene toliko. U sledećim tekstovima možete očekivati detaljnija objašnjenja i praktičnu primenu dockera.

Hvala na čitanju.

Kako setovati Docker radno okruženje?



NAPOMENA: Ovo i sva uputstva koja slede nisu oficijalna i moja su interpretacija originalne dokumentacije i uslova pod kojim se izvode. Tako da mogu i ne moraju da budu ispravna.

Za testne potrebe rada pod dockerom podigao sam mint 19.2 cinamon na virtualboxu. Tako da ako želite da vam sve izgleda identično thats the way to go.

Instalacija

Postoje nekoliko načina instalacije, biranje verzija blah blah. Mislim da je to previše za početnika. Idemo straight forward.

```
sudo apt update
```

```
apt install docker.io
apt install docker-compose
```

Posle uspešne instalacije oba paketa, ispalite

```
docker -v
```

```
Docker version 18.09.7, build 2d0083d
sistemas@osboxes:/home/osboxes$
```

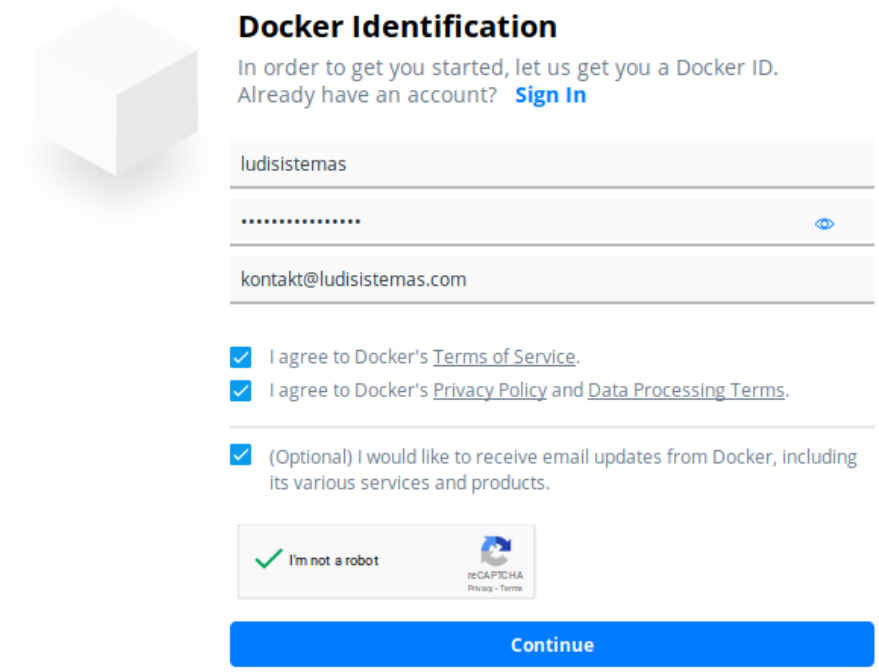
ovo je output kod mene

Prva komanda instalirala vam je Docker engine. Druga jedan jako koristan tool, koji će vam olakšati deployment kontejnera. Ali nećemo o tome sada.

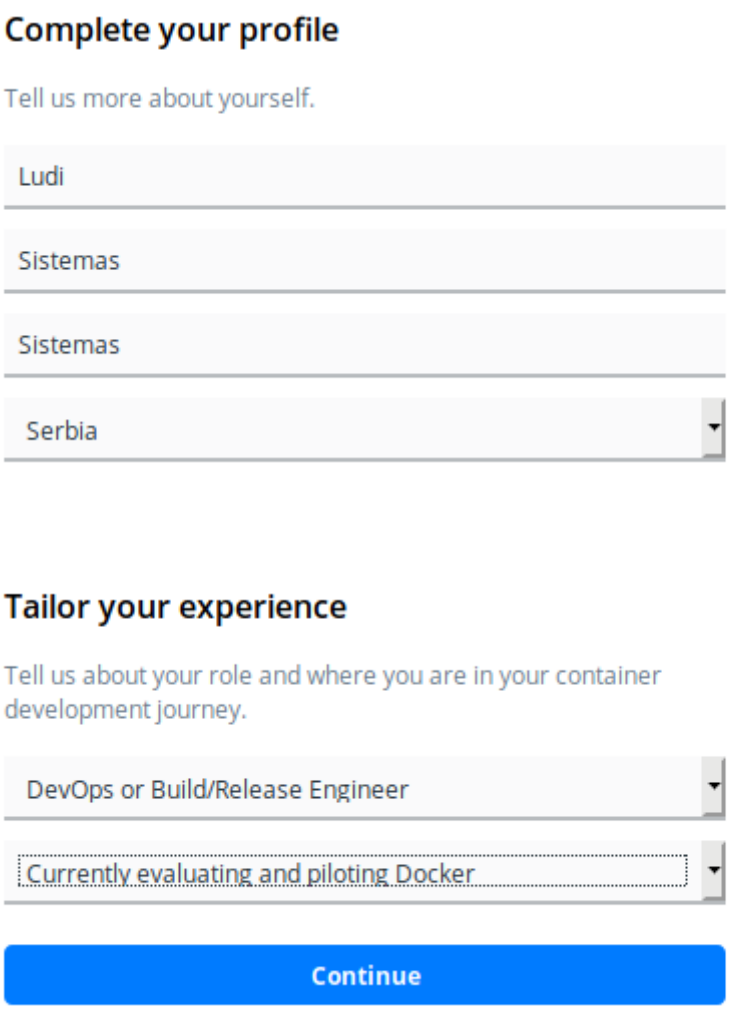
Registracija na docker hub

Sećate se onog cool mesta sa imidžima već spakovanih softvera, eee to je ovde.
https://hub.docker.com/signup

Za potrebe ovog tutorijala, napravih novi profil.



Na sledećoj stranici popunite kako god vam je volja. Nije za produkciju,

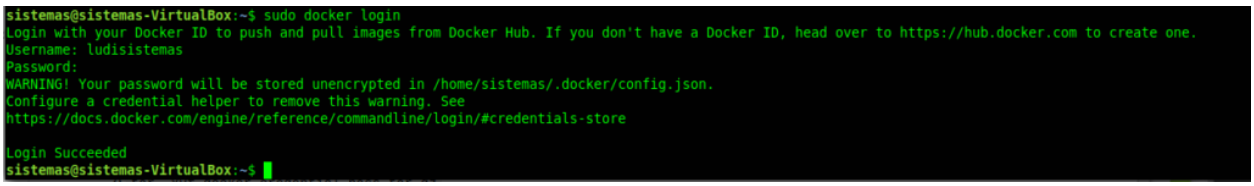


Nakon toga, stigao vam je verifikacioni mail sa linkom na koji trebate kliknuti.

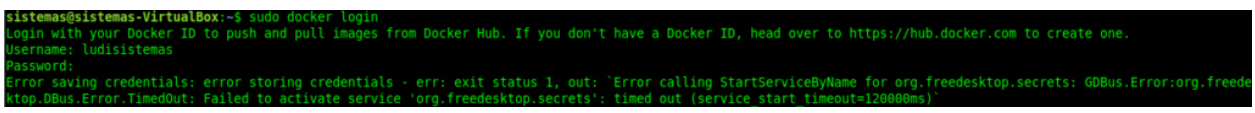
E sada se vraćamo nazad terminalu. I logujemo se na docker hub.

```
sudo docker login
```

Ulogujte sa sa kreiranim korisničkim imenom i lozinkom.



Potencijalni problem: – `Error saving credentials`



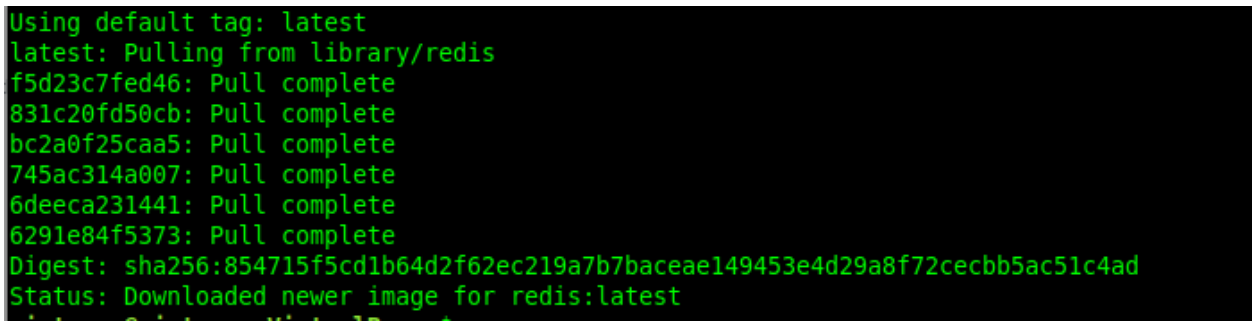
Rešenje;

```
sudo apt install gnupg2 pass
```

Kako bi testirali da li smo sve lepo setovali, hajde da svučemo jedan image.

```
sudo docker pull redis
```

Ako je sve prošlo kako treba, pokrenuće se download.



Sada na vašem računaru imate image iz repozitorijuma REDIS koji je tagovan kao latest tj poslednji.

Znači **repozitorijum:tag** je **redis:latest** .

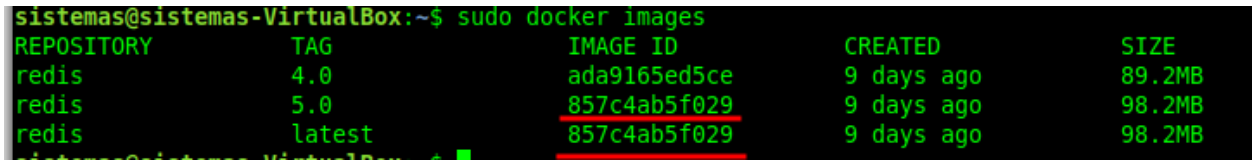
Na isti način možete pullovati neku stariju verziju redisa.

Probajte:

```
sudo docker pull redis:4.0
```

Da bi potvrdili koje image imamo u lokalu dovoljno je da ispalite:

```
sudo docker images
```



Ovo je moj listing. Ovde se vidi da imam 3 imidža. Ali pošto zadnje dve slike dele image ID. Možemo zaključiti da je verzija 5.0 poslednja.

Toliko u ovom tutorijalu. U sledećem ću se malo skoncentrisati na komande. I šta se i kada koristi.

Docker Cheat Sheet – sve bitne komande na jednom mestu

NAPOMENA: Ovo i sva uputstva koja slede nisu oficijalna i moja su interpretacija originalne dokumentacije i uslova pod kojim se izvode. Tako da mogu i ne moraju da budu ispravna.

Moj iskreni predlog je da bookmarkujete ovu stranicu. Ažuriraću je redovno i dodavati nove komande i objašnjenja.

Svlačenje imidža sa repozitorijuma

primer:

```
docker pull redis:latest
```

Dizanje imidža na repozitorijum

primer:

```
docker push redis:mojbuild
```

Koristi se za guranje imidža koje ste vi pravili na lokalu na neki od registra.

Listanje imidža

primer:

```
docker images
```

Izlistaće sve lokalne imidže koje ste preuzeli ili buildali.

Listanje kontejnera

```
docker ps
```

Gornja komanda izlistaće vam sve AKTIVNE kontejnere

```
docker ps -a
```

Izlistaće vam sve kontejnere, uključujući i exitovane. Ovo je korisno ako tražite id kontejnera koji se nije uspešno digao zbog daljeg debugovanja.

Buildanje image-a

```
docker build -t mojnoviimage .
```

Komanda traži Dockerfile u direktorijumu iz kog se izvršava (primećujete tačku na kraju) .
Komandom -t definište tag image-a.

Logovi

```
docker logs idkontejnera
```

Nakon što vam ne uspe dizanje kontejnera iz bilo kog razloga prvo mesto gde tražite problem su docker logovi. Prethodno komandom `docker ps -a` pokupite id exitovanog kontejnera i zatim ovom komandom iščitate šta nije u redu.

Čišćenje exitovanih kontejnera

```
docker container prune
```

Ovom komandom brišete sve exitovane kontejnere. Nakon nekog vremena nakupi se njih da zauzme pozamašni prostor. Sećate se da se svaki kontejner izvršava samo jednom.

Čišćenje neupotrebljenih imidža

```
docker image prune
```

Iako imidži native dele dosta prostora u lajerima, vremenom se nakupe razni updejti, pa neki mogu narasti i preko 1 gb. Puta broj raznih varijacija. Ode disk. Komanda će obrisati sve image koje u datom trenutku ne koristi najmanje jedan kontejner.

Čišćenje celog docker sistema

```
docker system prune
```

Ova iako na izgled bezopasna komanda bi trebalo bez nekih većih opasnosti ukloniti sve kontejnere, volume, imidže i ostatak crapa koji se nakupi radom dockera. Meni je slupao testno docker swarm okruženje. Tako da ga ne preporučujem. Nisam siguran da li je bio neki bug. Ali eto. Probajte pa vidite.

Docker compose

```
docker-compose up
```

izvršava se iz direktorijuma gde vam se nalazi docker-compose.yml fajl. Podiže ceo set zadatih kontejnera sa promenljivama koje ste uneli. (za ovo će trebati poseban post)

```
docker-compose up -d
```

Radi isto što i prethodna komanda ali vam se ne vezuje sa sesijom nego radi to detached, a vaš terminal ostaje slobodan za druge poslove

```
docker-compose -f moj-compose.yml up
```

Isto radi što i ove gore komande, ali ako imate par compose fajlova u istom direktorijumu, ovako možete izabrati onaj koji vam treba. Takođe radi sa -d

```
docker-compose down
```

Zaustavlja sve servise koji se pominju u compose fajlu direktorijuma iz kog se ispaljuje.

Docker swarm

```
docker stack deploy -c compose.yml --with-registry-auth imestaka
```

Komanda se ispaljuje na menadžeru iz direktorijuma gde je smešten compose fajl. `--with-registry-auth` kaže da za svlačenje imidža koji su naslovljeni u compose fajlu koristi kredencijale za registar kao i registar na koji je node na kome se deploy izvršava ulogovani. `imestaka` predstavalja ime kojim će se označavaju svi servisi (kontejneri) koje će ovaj compose fajl kreirati.

```
docker stack rm imestacka
```

Ovim uklanjate sve kontejnere koje ste podigli prethodnim compose fajlom.

```
docker stack ls
```

Ovim listate sve stackove. Pored imena svakog od njih, pisaće koliko servisa imaju podignutih (kontejnera)

```
docker service ls
```

Ovim listate sve servise na svim nodovima. Ova kao i sve prethodne komande mogu se izvršiti samo sa DOCKER MENADŽERA

Docker run – moj prvi kontejner

NAPOMENA: Ovo i sva uputstva koja slede nisu oficijalna i moja su interpretacija originalne dokumentacije i uslova pod kojim se izvode. Tako da mogu i ne moraju da budu ispravna.

docker run je komanda kojom ćete od imaga konačno napraviti kontejner. Što bi se reklo vaš image je porastao i vreme mu je da se uozbilji

Dakle. Prvo sa docker huba, preuzimamo neki image koji nam u ovom trenutku treba. Nekad to bude redis repozitorijum, poslednji image.

```
sudo docker pull redis:latest
```

```
sistemas@sistemas-VirtualBox:~$ sudo docker pull redis
Using default tag: latest
latest: Pulling from library/redis
1ab2bdfe9778: Already exists
966bc436cc8b: Pull complete
c1b01f4f76d9: Pull complete
8a9a85c968a2: Pull complete
8e4f9890211f: Pull complete
93e8c2071125: Pull complete
Digest: sha256:9755880356c4ced4ff7745bafe620f0b63dd17747caedba72504ef7bac882089
Status: Downloaded newer image for redis:latest
```

Kao što vidite u primeru, ako ne naznačite image, pullovaće se po defaultu latest iz tog repoa.

Hajde da pokrenemo direktno redis, bez ikakve pripreme. Čisto da vidimo šta će se desiti.

```
sudo docker run reds
```

```
sistemas@sistemas-VirtualBox:~$ sudo docker run redis
1:C 01 Sep 2019 12:12:24.117 # oO0OoO0OoO0Oo Redis is starting oO0OoO0OoO0Oo
1:C 01 Sep 2019 12:12:24.120 # Redis version=5.0.5, bits=64, commit=00000000, modified=0, pid=1, just started
1:C 01 Sep 2019 12:12:24.120 # Warning: no config file specified, using the default config. In order to specify a config file use redis-server /path/to/redis.conf
1:M 01 Sep 2019 12:12:24.121 * Running mode=standalone, port=6379.
1:M 01 Sep 2019 12:12:24.121 # WARNING: The TCP backlog setting of 511 cannot be enforced because /proc/sys/net/core/somaxconn is set to the lower value of 128.
1:M 01 Sep 2019 12:12:24.121 # Server initialized
1:M 01 Sep 2019 12:12:24.121 # WARNING overcommit_memory is set to 0! Background save may fail under low memory condition. To fix this issue add 'vm.overcommit_memory = 1
' to /etc/sysctl.conf and then reboot or run the command 'sysctl vm.overcommit_memory=1' for this to take effect.
1:M 01 Sep 2019 12:12:24.121 # WARNING you have Transparent Huge Pages (THP) support enabled in your kernel. This will create latency and memory usage issues with Redis.
To fix this issue run the command 'echo never > /sys/kernel/mm/transparent_hugepage/enabled' as root, and add it to your /etc/rc.local in order to retain the setting afte
r a reboot. Redis must be restarted after THP is disabled.
1:M 01 Sep 2019 12:12:24.121 * Ready to accept connections
```

Kao što vidite, redisov **proces** se pokrenuo. Da se podsetimo priča od ranije, u docker svetu proces=kontejner. Tako da nije pogrešno reći da smo pokrenuli docker kontejner redisa. Dobili smo log startupa. I on sada živi u ovoj sesiji koju sam inicijalizovao. Ako iz bilo kog razloga zatvorim ovaj terminal(ukinem sesiju). Redis će prestati da postoji.

Pošto je cilj eksperimenta postignut. Hajde da ukinemo sesiju kao bilo šta drugo ctl+c. I time ubijamo ovaj kontejner.

```
^C1:signal-handler (1567340427) Received SIGINT scheduling shutdown...
1:M 01 Sep 2019 12:20:27.365 # User requested shutdown...
1:M 01 Sep 2019 12:20:27.365 * Saving the final RDB snapshot before exiting.
1:M 01 Sep 2019 12:20:27.376 * DB saved on disk
1:M 01 Sep 2019 12:20:27.376 # Redis is now ready to exit, bye bye...
sistemas@sistemas-VirtualBox:~$
```

Hajde sada da pokrenemo redis u **detached** modu, tj da naša aktivna sesija više nema veze sa pokrenutim kontejnerima.

```
sudo docker run -d redis:latest
```

```
sistemas@sistemas-VirtualBox:~$ sudo docker run -d redis:latest
3fcef0c464c81edbc607dc0a6ae816c038660fa4b285724d7cdabea4e77a436a
sistemas@sistemas-VirtualBox:~$
```

Komandom `sudo docker ps` možemo izlistati startovane kontejnere.

```
sistemas@sistemas-VirtualBox:~$ sudo docker ps
[sudo] password for sistemas:
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS               NAMES
3fcef0c464c8        redis:latest        "docker-entrypoint.s…"   About an hour ago   Up About an hour    6379/tcp            amazing_lumiere
sistemas@sistemas-VirtualBox:~$
```

Ovo jeste zanimljivo, ali za bilo kakvo ozbiljnije korišćenje, predlažem rad sa compose fajlovima. O tome ću pisati u nekom drugom postu.

Za više informacija kako radi docker run, posetite ovaj link

https://docs.docker.com/engine/reference/run/

Dockerfile – Moj prvi image

Konačno smo dogurali do onog zanimljivog dela. Do pravljenja naše verzije nekog imidža. Ako vam je kojim slučajem izvetro šta je imidž, ne krivim vas, mnogo sam veliki razmak napravio između tekstova. Podsetite se klikom na OVAJ link

Pa da zaronimo. Dockerfile je srce svakog imidža. Svi do poslednjeg image koji ste preuzeli sa nekog registra napravljeni su na ovaj način.

Napomena: Dockerfile, mora baš tako i da se imenuje veliko D je obavezno. Bez ekstenzije je.

Udjite u test direktorijum, otvorite terminal. I kreirajte Dockerfile

```
touch Dockerfile
```

Editujte ga editorom koji vam odgovara. Ja ću koristiti nano.

```
```

```
nano Dockerfile
```

Hajde da napravimo imidž koji će biti zasnovan na php 7.2 fpm. I da mu ubacimo samo naj potrebnije stvari. Naravno i da ga ažuriramo.

```
FROM php:7.2-fpm

RUN  apt-get update && apt-get install -y

CMD ["php-fpm"]
EXPOSE 9000
```

Da raščlanimo:

`FROM php:7.2-fpm` – Ovim dockeru kažemo da preuzme imidž iz repozitorijuma php sa tagom 7.2-fpm. Ovako možete modifikovati bilo koji image koji imate u lokalnim repozitorijumima ili ste od negde preuzeli. Time će biti kreiran prvi sloj(layer) našeg imidža. A Dockerfile php:7.2-fpm možete pogledati ovde.

`RUN apt-get update && apt-get install -y` – Ovom komandom stvaramo novi sloj. Operativni sistem na kome je zasnovan php 7.2 fpm(debian:buster-slim) ažuriraće se i postaviti sloj broj 2 unutar ovog imidža.

RUN komandu, koristite isključivo za proces izvršavanja instalacija i modfikacija koji će se upisati u vašu verziju imidža.

`CMD ["php-fpm"]` – Po stvaranju kontejnera od ovog imidža, izvršiće se ova komanda. I tako će docker kontejner ostati da radi, neće pasti. U stvari mi samo pokrećemo php-fpm.

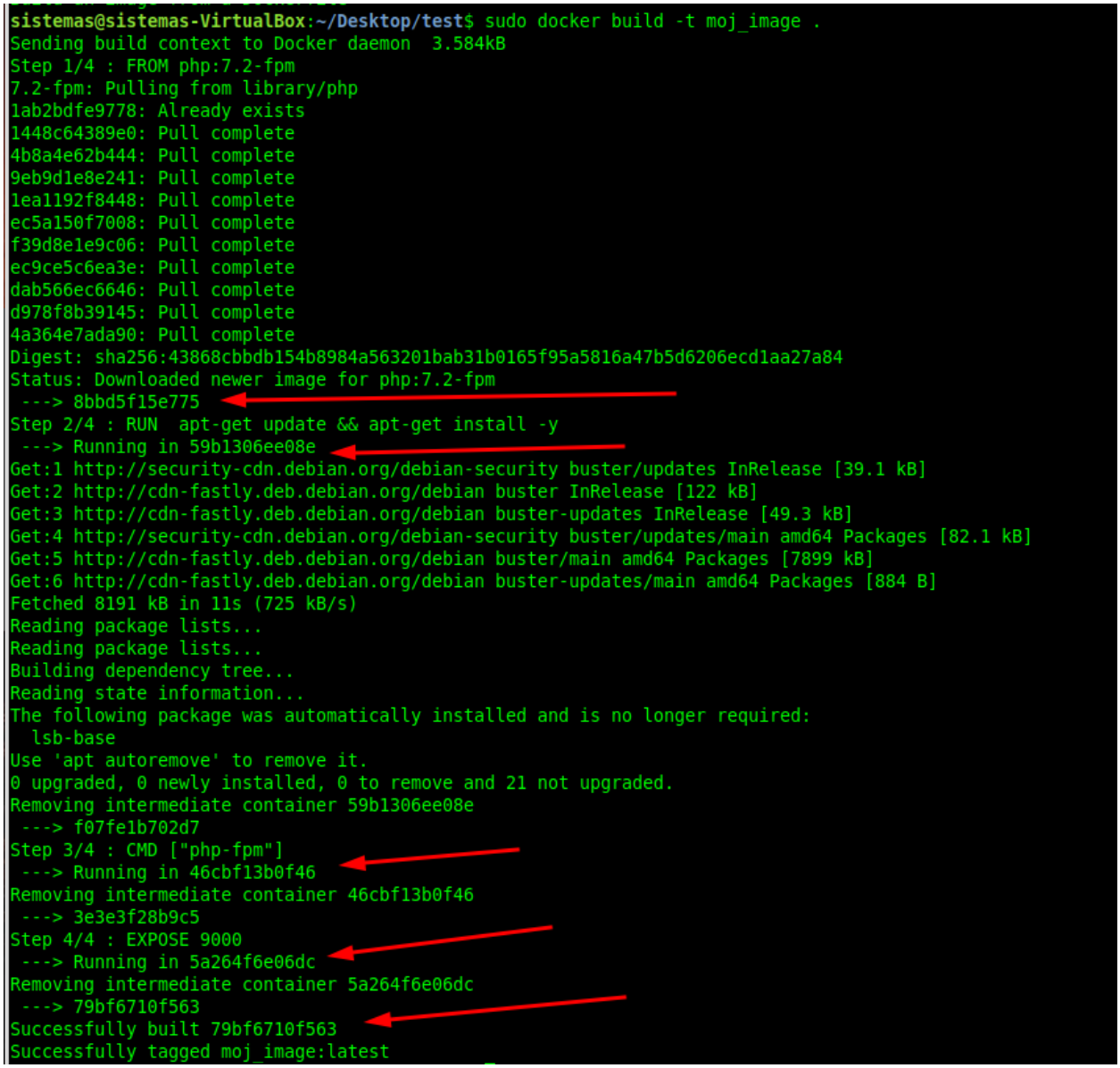
CMD – Ako imate više CMD direktiva u okviru jednog dockerfajla, izvršiće se sve u build procesu, sem poslednje. Ona će ostati da se izvrši pri podizanju kontejnera. Ovo je bitno znati, da se ne oslanjate na njih za više stvari koje treba uraditi na inicijalizaciji kontejnera. Za te stvari, koristimo ENTRYPOINT.

`EXPOSE 9000` – To znači da vam port 9000 neće biti otvoren kada pokrenete ovaj kontejner ka host mašini, već samo ka dockerovoj mreži.

Konačno buildanje

```
sudo docker build -t ime_imidza .
```

Komanda se pokreće iz direktorijuma gde je smešten Dockerfile. -t označava tag i obavezna je tačka na kraju, kako bi dockeru rekli da smo unutar direktorijuma gde nam je Dockerfile.



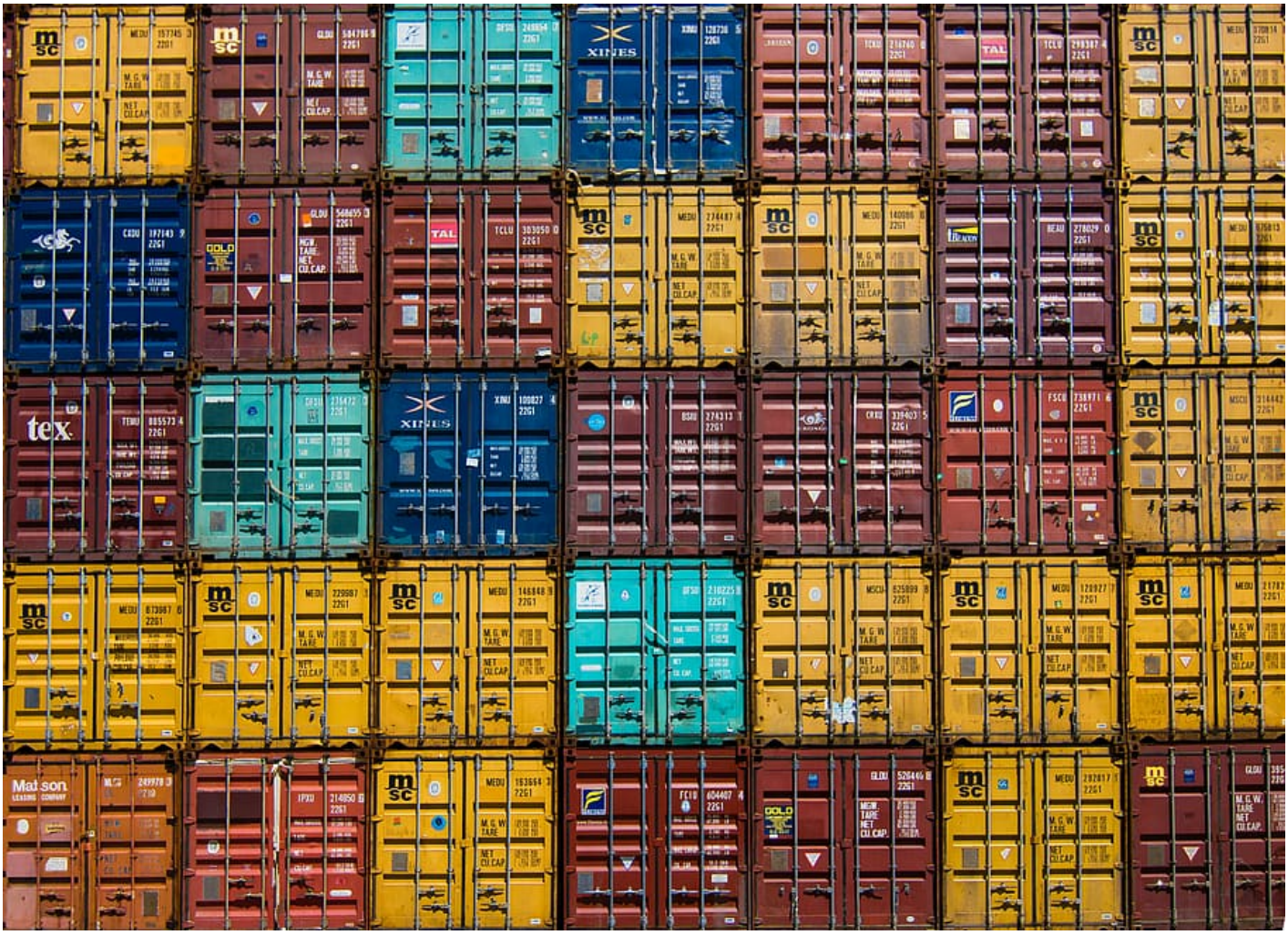
Kao što vidite, imali smo 4 komande u našem Dockerfajlu. Sve 4 su kreirale svoje kontejnere u okviru kojih su se izvršile i postale lajeri tj slojevi. I na kraju smo stvorili novi image. Sada pustite mašti na volju, vidite koliko se lako prave imidži. A jednom napravljeni imidž, možete na milion mesta koristiti.

I na kraju da vam napomenem. Prijavite se na moju mailing listu, ili na telegram kanal.

https://t.me/LudiSistemas . U slučaju da se desi neki problem sa fejsbuk stranicom, da ostanemo u kontaktu.

NAPOMENA: Ovo i sva uputstva koja slede nisu oficijalna i moja su interpretacija originalne dokumentacije i uslova pod kojim se izvode. Tako da mogu i ne moraju da budu ispravna.

docker-compose.yml – set instrukcija za stvaraje kontejnera



Docker compose možete posmatrati kao set instrukcija koji će dati vašem imidžuu neki dodatni set funkcionalnosti a koji ne mogu da se definišu unutar Dockerfajla. Uz njega možete izbeći sve one komplikacije gde morate u dugaćkim kobasicama navoditi šta želite da vaš kontejner ima otvoreno od portova, koje mount pointe da koristi, kako da mu bude ime definisano.

Neki od vas su sigurno bar jednom otkuali `docker-compose up`, možda baš u laradocku, i naveli šta sve žele da se startuje od usluga. E ovde ću pokušati da vam prikažem kroz objašnjenje jednog nasumičnog yaml fajla sa interneta, šta sve docker-compose može.

Podizanje wordpressa

```
Version: '3.3'

services:
   db:
     image: mysql:5.7
     volumes:
       - db_data:/var/lib/mysql
     restart: always
     environment:
       MYSQL_ROOT_PASSWORD: somewordpress
       MYSQL_DATABASE: wordpress
       MYSQL_USER: wordpress
       MYSQL_PASSWORD: wordpress

   wordpress:
     depends_on:
       - db
     image: wordpress:latest
     ports:
       - "8000:80"
     restart: always
     environment:
       WORDPRESS_DB_HOST: db:3306
       WORDPRESS_DB_USER: wordpress
       WORDPRESS_DB_PASSWORD: wordpress
       WORDPRESS_DB_NAME: wordpress
volumes:
    db_data: {}
```

NAPOMENA: Kod yaml kodiranja, jako su bitni indenti. Ako tu omašite za jedan razmak, yaml fajl neće raditi.

Komentar je isti kao u većini programskih jezika, počinje sa znakom # . Sada ću kopirati kod od iznad, i iskomentarisati vam u kodu i inline šta znači šta.

```
Version: '3.3'  #Verzija zavisi od toga koji docker engine koristite. Može se desiti da
sintaksa koju koristite ne bude podržana na starijim enginima.

services: #Ovde navodimo koje servise želimo da podignemo i definišemo imena
   db: #Definišemo ime servisa koji će gurati bazu podataka
     image: mysql:5.7 #Ovde navodimo od kog imidža sa docker huba se ovaj kontejner/servis
spawnuje
     volumes: #ovde definišemo persistentnu memoriju kontejnera.
       - db_data:/var/lib/mysql # lokalni_dir:/lokacija/u/kontejneru
     restart: always # U slučaju da nešto krene naopako pokrećemo ga opet
     environment: #Definišemo globalne promenljive ovog kontejnera
       MYSQL_ROOT_PASSWORD: somewordpress
       MYSQL_DATABASE: wordpress
       MYSQL_USER: wordpress
       MYSQL_PASSWORD: wordpress

   wordpress: #Ime jednog od servisa, obratite pažnju da je u istom indentu kao i db gore
     depends_on: #Ovde možemo da kažemo ovom kontejneru da čeka dok se ne podigne neki
kontejner
       - db  #U stvari čekamo bazu, ili php ima da kaže da ne vidi bazu, i srušiće se
kontejner
     image: wordpress:latest #Opet navodimo koji image želimo da podginemo
     ports: #Definišemo koje portove želimo da prosledimo lokalnoj mašini sa dockera
       - "8000:80"
     restart: always #Isti uslov kao i za bazu
     environment:
       WORDPRESS_DB_HOST: db:3306 #Obratite pažnju, ovde gađamo ime servisa:kroz port u
docker networku
       WORDPRESS_DB_USER: wordpress
       WORDPRESS_DB_PASSWORD: wordpress
       WORDPRESS_DB_NAME: wordpress
volumes:
    db_data: {} #Ovde mountujemo db_data iz meni nepoznatog razloga u okviru wordpressa :)
```

Ovaj fajl sačuvajte kao docker-compose.yml i napravie direktoriju db_data. Nakon što ispalite

dokcer-compose up. DIžu se se DVA kontejnera/servisa. Kontejner koji se zove db i kontejner koji se zoba wordpress. I oni komuniciraju bezbedno kroz docker mrežu. Ali to je već priča za sebe. O tome drugi put.

Ako budete imali pitanja, postavite ih u komentaru na fejsbuk stranici. A ja ću dopisati odgovore u okviru ovog članka.