Vodič kroz doker

Ovde ćemo se fokusirati na sledeće teme:

- **Šta je virtuelizacija?**
- **Šta je kontejnerizacija**
- **Prednosti kontejnerizacije u odnosu na virtuelizaciju**
- **Uvod u Dokera**
- **Prednosti Dokera**
- **Virtuelizacija vs kontejnerizacija**
- **Instalacija dokera**
- **Dokerfile, Doker slika i Doker kontejner**
- **Šta je Docker Hub?**
- **Doker arhitektura**
- **Docker Compose**

Doker dobija na popularnosti i njegova upotreba se širi kao veliki požar. Razlog sve veće popularnosti Dokera je do koje mere može da se koristi u IT organizaciji. Veoma mali broj alata ima funkcionalnost koja se može napraviti korisnom i programerima i administratorima sistema. Doker je jedan takav alat koji zaista ispuni svoja obećanja za "**Build**", **"Ship** and **Run"**.

Jednostavnim rečima, **Doker je softverska platforma za kontejnerizaciju, što znači da možete da napravite aplikaciju, spakujete ih zajedno sa njihovim**

zavisnostima u kontejner, a zatim se ti kontejneri mogu lako isporučiti da rade na drugim mašinama.

Na primer: Razmotrimo aplikaciju zasnovanu na Linuxu koja je napisana i na Rubi i na Pythonu. Ova aplikacija zahteva specifičnu verziju Linuxa, Rubi i Pythona. Da bi se izbegle bilo kakve neusaglašenosti verzija na kraju korisnika, može se kreirati linux doker kontejner sa potrebnim verzijama Ruby i Python instaliranim zajedno sa aplikacijom. Sada krajnji korisnici mogu lako da koriste aplikaciju pokretanjem ovog kontejnera bez brige o zavisnostima ili neusaglašenosti verzija.

Ovi kontejneri koriste kontejnerizaciju koja se može smatrati razvijenom verzijom virtuelizacije. Isti zadatak se može postići korišćenjem virtuelnih mašina, ali nije baš efikasan.

U ovom trenutku, uglavnom dobijam pitanje, pa. Koja je razlika između virtuelizacije i kontejnerizacije? Ova dva koncepta su veoma slična jedno drugom. Prvo da vam kažem šta je virtuelizacija?

Šta je virtuelizacija?

Virtuelizacija je tehnika uvoza gostujućeg operativnog sistema povrh sistema hostoperacije. Ova tehnika je u početku bila otkrovenje jer je omogućavala programerima da pokreću više operativnih sistema u različitim virtuelnim mašinama koje sve rade na istom domaćinu. Ovo je eliminisalo potrebu za dodatnim hardverskim resursima.

Prednosti virtuelnih mašina ili virtuelizacija su:

- Više operativnih sistema može da radi na istoj mašini
- Održavanje i oporavak su bili laki u slučaju kvarova
- Ukupni troškovi vlasništva takođe su bili manji zbog smanjenih infrastrukturnih potreba.

Kao što znate da ništa nije savršeno, **virtuelizacija takođe ima neke mane**. Pokretanje više virtuelnih mašina u istom operativnom sistemu domaćina dovodi do

pogoršanja performansi. To je posledica gostujućeg OS-a koji radi na glavnom OS-u, koji će imati sopstveni jezgro i skup biblioteka i zavisnosti. Ovo zauzima veliki deo sistemskih resursa, naime. Čvrsti disk, procesor i posebno RAM memorija.

Sledeći problem sa virtuelnim mašinama koje koriste virtuelizaciju je taj što je potrebno skoro minut da se pokrene. Ovo je veoma važno u slučaju aplikacija u realnom vremenu.

Slede mane virtuelizacije:

- **Pokretanje više virtuelnih mašina dovodi do nestabilnih performansi**
- **Hipervisori nisu efikasni kao operativni sistem domaćina**
- **Proces pokretanja je dugačak i zahteva vreme**

Ovi nedostaci doveli su do pojave nove tehnike zvane Kontejnerizacija.

Šta je kontejnerizacija?

Kontejnerizacija je tehnika dovođenja virtuelizacije na nivo operativnog sistema. Dok virtuelizacija donosi apstrakciju hardveru, kontejnerizacija donosi apstrakciju operativnom sistemu. Imajte na umu da je kontejnerizacija takođe vrsta virtuelizacije. Međutim, kontejnerizacija je efikasnija jer ovde nema gostujućeg OS-a i koristi operativni sistem domaćina, deleći relevantne biblioteke i resurse po potrebi i za razliku od virtuelnih mašina. Binarne datoteke specifične za aplikaciju i kontejnerske biblioteke pokreću se na jezgru domaćina, što čini obradu i izvršavanje veoma brzo. Čak i za pokretanje tenka treba samo delić sekunde. Pošto svi kontejneri dele, hostujte operativni sistem i sadrže samo binarne datoteke i biblioteke povezane sa aplikacijom. Lagane su i brže od virtuelnih mašina.

Prednosti kontejnerizacije u odnosu na virtuelizaciju:

- **Rezervoari na istom jezgru OS-a su lakši i manji**
- **Bolje korišćenje resursa u poređenju sa VMs**
- **Proces početka je kratak i traje nekoliko sekundi.**

Kontejneri sadrže samo biblioteke specifične za aplikaciju koje su izdvojene za svaki kontejner, brži su i ne troše resurse.

Sa svim ovim kontejnerima obrađuje se sloj kontejnerizacije koji nije urođen u operativni sistem domaćina. Zbog toga je potreban softver koji vam omogućava da kreirate i pokrenete kontejnere na operativnom sistemu domaćina.

Uvod u Doker

Doker je platforma za kontejnerizaciju koja pakuje vašu aplikaciju i sve zavisnosti zajedno u kontejnerskom obliku kako bi osigurala da vaša aplikacija neprimetno radi u bilo kom okruženju.

Svaka aplikacija će se pokrenuti na posebnom kontejneru i imaće svoj skup biblioteka i zavisnosti. Ovo takođe obezbeđuje izolaciju na nivou procesa, što znači da je svaka aplikacija nezavisna od drugih aplikacija, pružajući programerima garanciju da mogu da prave programe koji neće ometati jedni druge.

Kao programer, mogu da napravim kontejner na kome se instaliraju različite aplikacije i predam ga svom QA timu koji će morati da pokrene kontejner samo da bi replicirala razvojno okruženje.

Prednosti Dokera

Sada QA tim ne mora da instalira sve aplikacije za testiranje zavisnih softvera i kodova, a to im pomaže da uštede mnogo vremena i energije. Takođe se obezbeđuje da radno okruženje bude dosledno za sve pojedince koji učestvuju u procesu, počevši od razvoja do raspoređivanja. Broj sistema se može lako povećati i kôd im se može bez napora distribuirati.

Virtuelizacija vs kontejnerizacija

Virtuelizacija i kontejnerizacija vam omogućavaju da pokrenete više operativnih sistema unutar glavnog računarskog sistema.

Virtuelizacija se bavi kreiranjem mnogih operativnih sistema na jednom računaru domaćina. Sa druge strane, kontejnerizacija će po potrebi kreirati više kontejnera za svaku vrstu aplikacije.

Glavna razlika je u tome što u virtuelizaciji nedostaje više gostujućih operativnih sistema koji nedostaju u kontejnerizaciji. Najbolji deo u vezi sa kontejnerizacijom je da je veoma lagan u poređenju sa teškom virtuelizaciju.

Kako se instalira Docker

Instaliraj dokera:

Instaliraću Dockera na moju Ubuntu 17.10 mašinu. Slede koraci za instaliranje Dokera:

1. Instaliranje neophodnih paketa
2. Postavite Docker skladište
3. Instaliraj Dokera na Ubuntu

1. Instalirajte potrebne pakete:

Postoje određeni paketi koji su vam potrebni za instaliranje Dokera. Izvršite dole navedenu komandu da biste instalirali ove pakete.

```
sudo apt-get install curl apt-transport-https ca-bici software-properties-common
```

2. Postavljanje doker depoa:

Sada uvezite zvanični GPG ključ Dockersa da biste proverili potpis paketa pre nego što ih instalirate koristeći apt-get. Pokrenite dole navedenu komandu na terminalu:

```
curl -fssl https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add
```

Sada dodajte Docker skladište u vaš Ubuntu sistem koji sadrži Docker pakete, uključujući i njegove zavisnosti, izvršite komandu ispod:

```
sudo add-apt depo 'deb [arch =amd64]
https://download.docker.com/linux/ubuntu $ (lsb_release -cs) stabilan'
```

3. Instalirajte Dokera na Ubuntu:

Sada je potrebno da nadogradite apt indeks i instalirate Doker izdanje zajednice, da bi se na taj način izvršile dole navedene komande:

```
sudo apt-get update sudo apt-get installation docker-ce
```

Čestitam! Uspešno ste instalirali Dokera .

Da vidimo nekoliko važnijih Dokerovih koncepata.

Docker file, Docker image i Docker tank:

1. Dokerova slika je kreirana nizom komandi napisanih u datoteci "Dokerfile".
2. Kada se ova Docker datoteka izvrši pomoću komande dokera, ona rezultira Doker slikom sa imenom.
3. Kada se ova slika izvrši komandom "doker pokreni", ona će sama pokrenuti bilo koju aplikaciju ili uslugu koju mora da pokrene prilikom njenog izvršavanja.

Doker čvorište:

Docker Hub je kao GitHub za Docker Images. To je u osnovi registar u oblaku gde možete da pronađete Dokerove slike koje otpremaju različite zajednice, takođe možete da razvijete sopstvenu sliku i otpremite ih u Docker Hub, ali prvo morate da kreirate nalog na DockerHub-u.

Doker arhitektura:

Sastoji se od doker mašine koja je klijent-serverska aplikacija sa tri glavne komponente:

1. Server koji je tip dugoročnog programa koji se zove demonski program (doker komanda).
2. REST API koji navodi interfejse koje programi mogu da koriste za razgovor sa demonom i nalažu šta da rade.
3. Klijent interfejsa komandne linije (CLI) (komanda dokera).
4. CLI koristi Docker REST API za kontrolu ili interakciju sa Docker daemon putem skripti ili direktnih CLI komandi. Mnoge druge Doker aplikacije koriste osnovne API i CLI.

Kompozicije Doker:

Doker Kompon se u osnovi koristi za pokretanje više Doker kontejnera kao jedan server. Daću vam primer:

Pretpostavimo da imam aplikaciju koja zahteva WordPress, Maria DB i PHP MyAdmin. Oni mogu da kreiraju jednu datoteku koja bi pokrenula oba kontejnera kao uslugu, a da pri tom ne moraju da se pokreću svaki posebno. Zaista je korisno naročito ako imate mikroservis arhitekturu.

Install Docker - Docker installation on Ubuntu and CentOS

Naučimo kako da instalirate Docker na Ubuntu i CentOS. Takođe ćete naučiti kako da pokrenete Kontejner dokera pomoću komande

"doker run".

Dakle, počnimo tako što ćemo prvo instalirati Dokera na Ubuntu operativni sistem.

Docker instalacija na Ubuntu-u

Korak 1: Da biste instalirali doker na Ubuntu box, prvo nam dozvolite da ažuriramo pakete.

```
sudo apt-get update
```

2. korak: Pre instaliranja dokera, potrebno je da instaliram preporučene pakete. Za ovo samo otkucajte dole navedenu komandu:

```
sudo apt-get install linux-image-extra - $ (uname -r) linux-image-extra-virtual
```

Pritisnite "y" da biste nastavili.

Posle toga, završili smo sa preduslovima! Idemo sada da instaliramo Dokera.

3. korak: Otkucajte dole navedenu komandu da biste instalirali doker mašinu:

```
sudo apt-get install docker-engine
```

Ponekad će ponovo tražiti lozinku. Pritisnite taster Enter i instalacija će početi.

Ako to uradite, zadatak instalacije dokera će biti dovršen!

Korak 4: Pa, hajde da jednostavno pokrenemo doker servis. Za ovo samo otkucajte dole navedenu komandu:

```
sudo servis doker start
```

Doker je uspešno instaliran.

Korak 5: Sada samo da proverim da li doker uspešno radi, dozvolite mi da vam pokažem kako da izvučete CentOS sliku iz centra dokera i pokrenete CentOS kontejner. Za ovo samo otkucajte dole navedenu komandu:

```
sudo docker pull centos
```

Prvo će proveriti da li postoji slika u lokalnom registru u CentOS-u. Ako nije tamo, otići će na raskrsnicu i povući sliku. Pogledajte dole navedeni snimak ekrana radi boljeg razumevanja:

Zbog toga smo uspešno izvukli centOS sliku iz doker čvorišta. Sledeće, oslobodimo CentOS kontejner. Za ovo samo otkucajte dole navedenu komandu:

```
Sudo Docker Run - It Centos
```

Kao što vidite na snimku ekrana iznad, sada smo u CentOS kontejneru!

Da rezimiramo, prvo smo instalirali doker na Ubuntu, nakon čega smo izdvojili CentOS sliku iz dokerskog čvorišta i koristeći tu sliku, uspešno smo napravili CentOS kontejner.

Osnovne doker komande ,biće vam od koristi u svakodnevnoj upotrebi dokera.

Trend doker kontejnera neopozivo raste sa organizacijama koje aktivno traže profesionalce koji poseduju i dobro poznavanje ovih Dokerovih komandi daće vam stručnost koja vam je potrebna.

Slede komande koje su pokrivene:

1. **doker verzija**
2. **docker pull**
3. **doker trčanje**
4. **Doker PS**
5. **doker ps -a**
6. **doker izvršava**
7. **doker zaustavljanje**
8. **Doker da ubije**
9. **doker se obavezuje**
10. **prijavite se za dokera**
11. **potises to the dock**
12. **doker slike**
13. **Doker RM**
14. **Doker RMI**
15. **Izgradnja dokera**

Dokerova naređenja

1. **doker verzija**

Upotreba: docker -version

Ova komanda se koristi za dobijanje trenutno instalirane verzije dokera

2. **docker pull**

Upotreba: docker pull

Ova komanda se koristi za izdvajanje slika **iz prodavnice dokera** (hub.docker.com)

3. **Docker Run**

Upotreba: doker run -it -d

Ova komanda se koristi za kreiranje kontejnera sa slikama

Četiri. **doker ps**

Ova komanda se koristi za listu pokrenutih kontejnera

5. **doker ps -a**

Ova komanda se koristi za prikazivanje svih započetih kontejnera i izlaza

6. **Doker izvršava**

Ознаено a: docker exec -it bash

Ova komanda se koristi za pristup trenutnom kontejneru

7. **doker stop**

Upotreba: zaustavljanje dokera

Ova komanda sprečava pokretanje kontejnera

8. **docker kill**

Upotreba: docker kill

Ova naredba ubija kontejner tako što odmah zaustavlja njegovo pogubljenje. Razlika između 'doker kill' i 'docker stop' je u tome što 'doker stop' daje rezervoaru vremena da se graciozno isključi, u situacijama kada je potrebno previše vremena da kontejner prestane, može se odlučiti da ga ubije

9. Doker **se obavezuje**

Upotreba: doker commit

Ova komanda kreira novu sliku uređenog kontejnera u lokalnom sistemu

10. **prijava na dokere**

Ova komanda se koristi za prijavljivanje na skladište čvorišta dokera

Jedanaest. **doker potiska**

Upotreba: potisak dokera

Ova komanda se koristi za guranje slike u prodavnicu doker čvorišta

12. **Doker slike**

Ova komanda navodi sve lokalno uskladištene slike dokera

13. **docker rm**

Upotreba: docker rm

Ova komanda se koristi za brisanje zaustavljenog kontejnera

14. **Docker RMI**

Upotreba: docker rmi

Ova komanda se koristi za brisanje slike iz lokalnog skladišta

Petnaest. **Izgradnja dokera**

Upotreba: doker konstrukcija

Ova komanda se koristi za izradu slike iz određene datoteke dokera

Doker kontejner je lagano alternativno rešenje za virtuelnu mašinu za kreiranje, podešavanje i pokretanje aplikacija unutar kontejnera.

Šta će nam doker kontejneri?

Ideja mikroservisa je da određene vrste aplikacija postaju lakše građene i održavane kada se razbiju na manje komade koji se mogu sastaviti i koji rade zajedno. Svaka komponenta se razvija posebno, a aplikacija je zatim jednostavno zbir komponenti komponenti.

U ovoj arhitekturi smo koristili CentOS virtuelne mašine. Ove virtuelne mašine su konfigurisane pisanjem dugih skripti. Pa, konfigurisanje tih VM-a nije bio jedini problem.

Razvoj takvih aplikacija zahteva pokretanje nekoliko mikro usluga u jednoj mašini. Dakle, ako vodite pet tih usluga, treba vam pet VM-a na toj mašini. Uzmite u obzir dijagram ispod:

Drugi problem je prilično čest, znam da mnogi od vas mogu da se poistovete sa njim. Aplikacija radi na laptop računaru projektanta, ali ne i u testiranju ili proizvodnji. Do ovoga može doći zbog nepridržavanja usaglašenog okruženja za računare. Uzmite u obzir dijagram ispod:

Bilo je mnogo drugih problema osim ovoga, ali smatram da su ovi problemi dovoljni da vam objasnim potrebu za Doker tenkovima.

Pa zamislite da dam 8GB RAM memorije svim mojim VM-ovima i imam 5 mikroservisa koji rade na razlicitim virtuelnim aparatima. U ovom slučaju, ovim VM-ovima će biti potrebno 40GB RAM memorije. Eto, sada zahtevam da konfiguracije mog host računara budu veoma visoke, skoro 44GB RAM-a bi trebalo da bude tu na mom domaćinu. Očigledno je da ovo nije održivo rešenje za takvu arhitekturu, jer ovde trošim mnogo resursa.

Ok, imam mnogo resursa za gubljenje, ali još uvek imam problem nedoslednosti u životnom ciklusu isporuke softvera (SDLC). Moram da konfigurišem ove VM-ove na testu kao i u proizvodnom okruženju. Negde u toku procesa neki softver se ne ažurira na probnom serveru, a razvojni tim koristi ažuriranu verziju softvera. To dovodi do konflikta.

Šta ako koristim 100 VM-a, onda će konfiguracija svakog VM-a potrajati mnogo vremena, a takođe je i sklon greškama.

Hajde da otkrijemo šta je Doker Kontejner i kako funkcioniše i kako je rešio moj problem.

Šta je Dokerov tenk?

Doker je alat dizajniran da olakša kreiranje, razvoj i pokretanje aplikacija pomoću kontejnera.

Možete kreirati Dokerove kontejnere, ovi kontejneri će sadržati sve binarne datoteke i biblioteke potrebne u vašoj aplikaciji ili mikroservisu u mom slučaju. Dakle, vaša aplikacija je prisutna u kontejneru ili ste je kontejnerizovali. Sada se taj isti kontejner može koristiti u probnom i produženom okruženju.

Doker tenkovi su lagano rešenje za virtuelne mašine i koriste host OS. Najbolji deo, nije potrebno da ram memoriju unapred dodelite tenku Doker, biće vam potrebna po potrebi i kada je to potrebno. Sa Doker kontejnerom ne moram da brinem o traćenju resursa.

Hajde da otkrijemo kako radi Kontejner dokera.

Kako funkcioniše Doker kontejner?

- Projektant će prvo zapisati kôd projekta u Docker datoteku, a zatim će napraviti sliku iz te datoteke.
- Ova slika će sadržati ceo kôd projekta.
- Sada možete da pokrenete ovu Doker sliku da biste kreirali onoliko kontejnera koliko želite.
- Ova Dokerova slika se može otpremiti u Docker čvorište (u osnovi to je skladište u oblaku za vaše Doker slike, možete ga držati javnim ili privatnim).
- Ovu Doker sliku na Doker čvorištu mogu da povuku drugi timovi kao što su QA ili Prod.

To ne samo da sprečava otpad resursa, već obezbeđuje da računarsko okruženje koje postoji u laptop računaru projektanta bude kopirano i u druge timove. Sada osećam da ne moram da ti kažem zašto nam treba Doker.

Ovo je bio jedan od načina da ga upotrebim, pretpostavljam da ljude zanima kako sam koristio Dokera da rešim problem sa mikroservisima. Daću vam pregled istog.

- **Prvo smo napisali složene zahteve u Dockerfileu.**
- **Onda smo ga gurnuli u GitHub.**
- **Posle toga smo koristili CI server (Dženkins).**
- **Ovaj Dženkins server će ga izvući iz Gita i izgraditi tačno okruženje. Ovo će se koristiti na proizvodnim serverima, kao i na probnim serverima.**
- **Primenili smo ga u pripremno okruženje (odnosi se na podešavanje softvera na serverima u svrhe testiranja, pre nego što ga u potpunosti implementišemo u proizvodnju.) Okruženja za testere.**
- **U suštini, uložili smo u proizvodnju upravo ono što smo imali u razvoju, testiranju i podešavanju.**

Zapravo će biti pošteno reći da mi je Doker olakšao život.

Pogledajmo studiju slučaja Univerziteta Indijana. Kako je Doker rešio njihove probleme.

Studija slučaja sa Univerziteta Indijana:

Univerzitet Indijana je javni univerzitetski sistem sa više kampusa u državi Indijana, Sjedinjene Države.

Izjava o problemu

Koristili su prilagođene skripte za postaviti aplikacije u VM.

Njihovo okruženje je optimizovano za njihove nasleđene Java programe. Njihovo rastuće okruženje uključuje nove proizvode koji nisu samo zasnovani na javi. Da bi svojim studentima pružio najbolje moguće iskustvo, Univerzitetu je bilo potrebno da počne sa modernizacijom aplikacija.

Univerzitet je želeo da unapredi način na koji oblikuju aplikacije prelaskom na arhitekturu zasnovanu na mikroservisu za njihove aplikacije.

Bezbednost je bila potrebna za podatke studenata kao što su SSN i podaci o zdravlju studenata.

Rešenje:

Sve probleme je rešio Docker Data Center (DDC),

Docker trusted Registry - Skladišti Doker slike.

UCP (Univerzalni kontrolni avion) Veb korisnički interfejs - Pomaže u upravljanju celim klasterom sa jednog mesta. Usluge se koriste UCP Veb korisnički interfejs pomoću Doker slika uskladištenih u DTR (Docker trusted Registry).

IT operativni timovi koriste Univerzalni kontrolni avion da obezbede Doker softver instaliran na domaćinima, a zatim posagne svoje aplikacije bez potrebe za ručnim koracima za podešavanje sve njihove infrastrukture.

UCP i DTR se integrišu sa svojim LDAP serverom radi brzog pristupa aplikacijama.

Nadam se da ste čitali prethodne blogove da biste naučili osnove Dokera.

Dozvolite mi da objasnim kako moћemo da koristimo Docker Compose za aplikacije sa više kontejnera.

Pogodna upotreba Dokera:

Pretpostavljam da ste instalirali Docker.In ovaj post koristiću Docker Compose, ispod sam dao mali uvod u Docker Compose.

Sastavi doker: To je alatka za definisanje i pokretanje doker aplikacija sa više kontejnera. Pomoću funkcije Docker Compose možete da konfigurišete usluge aplikacije za pisanje datoteke. Zatim, koristeći jednu komandu, možete da kreirate i pokrenete sve usluge iz konfiguracije.

Recimo da imate više aplikacija u raznim kontejnerima i da su svi ti kontejneri povezani zajedno. Dakle, ne želite da pogubite svaki od ovih kontejnera jedan po jedan. Ali, želite da pokrenete te kontejnere jednom komandom. Tu na slici dolazi Doker Kompon. Pomoću nje možete da pokrenete više aplikacija u različitim kontejnerima pomoću jedne komande. i.e. doker-sklop.

Primer: Zamislite da imate različite kontejnere, jedan koji pokreće web aplikaciju, drugi pokreće postgrese i drugi koji pokreće ponovne prikaze u YAML datoteci. Ovo se zove doker datoteka za pisanje, odatle možete da pokrenete ove kontejnere jednom komandom.

Uzmimo još jedan primer:

Pretpostavimo da želite da objavite blog, za to ćete koristiti cms (content management system) a wordpress je najčešće korišćen CMS. U suštini, potrebna vam je jedna wordpress posuda, a za pozadinu vam je potreban još jedan kontejner kao što je MySQL, da MySQL kontejner treba da bude povezan sa wordpress kontejnerom. Potreban nam je i još jedan Php Myadmin kontejner koji će biti povezan sa MySQL bazom podataka, koja se u osnovi koristi za pristup MySQL bazi podataka.

Kako bi bilo da izvedem gore navedeni primer praktično.

Obuhvaćeni koraci:

1. **Instaliraj Docker Compose :**
2. **Install WordPress:** Koristićemo zvanično WordPress i MariaDB Docker slike.
3. **Instalirajte MariaDB:** To je jedan od najpopularnijih servera baze podataka na svetu. Napravili su ga originalni mysql developeri. MariaDB je razvijen kao open source softver, a kao relaciona baza podataka obezbeđuje SQL interfejs za pristup podacima.

4. **Install PhpMyAdmin:** To je alat za slobodan softver napisan na PHP-u, namenjen administriranju MySQL-a preko weba.
5. **Kreirajte WordPress sajt:**

Hajde da počnemo!

Instaliraj docker compose:

Prvo instaliraj Python Pip:

```
sudo apt-get install python-pip
```
Php Mysql_fetch_array

Sada možete da instalirate Docker Compose:

```
sudo pip install docker-compose
```
Instaliraj WordPress:

Kreirajte WordPress direktorijum:

```
mkdir wordpress
```
Unesite ovaj wordpress direktorijum:

```
cd wordpress /
```
U ovom direktorijumu kreirajte Docker Compose YAML datoteku, a zatim je uredite pomoću gedita:

```
sudo gedit doker-compositions.yml
```
Nalepite donje redove koda u tu yaml datoteku:

```
wordpress: slika: wordpress linkovi: - wordpress_db: mysql portovi: - 8080: 80 wordpress_db: slika: mariadb okruženje: MYSQL_ROOT_PASSWORD: edureka phpmyadmin: image: corbinu / docker-phpmyadmin linkovi: - wordpress_db: mysql portovi: - 8181: 80 MYSQL_USERNAME: root MYSQL_ROOT_PASSWORD:
```
Znam da želiš da ti objasnim ovaj kod, pa, šta ću da uradim, uzeti ću male delove ovog koda i objasniću ti šta se dešava.

```
wordpress_db: ... okolina: MYSQL_ROOT_PASSWORD: edureka ...
```
Ovo će postaviti promenljivu okruženja unutar kontejnera wordpress_db koji se MYSQL_ROOT_PASSWORD sa željenom lozinkom. Slika MariaDB Docker je konfigurisana da proverava ovu promenljivu okruženja kada se pokrene i vodiće

računa o podešavanju DB-a sa osnovnim nalogom sa lozinkom definisanom kao MYSQL_ROOT_PASSWORD.

```
wordpress: ... luke: - 8080: 80 ...
```

Prvi broj porta je broj porta na domaćinu, a drugi broj porta je port unutar kontejnera. Dakle, ova konfiguracija proslepuje zahteve na 8080 portu za host do podrazumevanih 80 Porta Web servera 80 u kontejneru.

```
phpmyadmin: image: corbinu / docker-phpmyadmin links: - wordpress_db: mysql portovi: - 8181: 80 okruženje: MYSQL_USERNAME: root MYSQL_ROOT_PASSWORD: edureka
```

Ovo grabi doker-phpmyadmin od strane člana zajednice Corbina, povezuje ga sa našim wordpress_db kontejnerom sa imenom mysql (što znači da će u okviru phpmyadmin-ovog kontejnera, reference na ime host mysql biti prosleđene našem wordpress_db kontejneru), izlaže svoj port 80 na portu 8181 iz sistema domaćina, i konačno postavlja nekoliko promenljivih okruženja sa našim mariaDB korisničkim imenom i lozinkom Ova slika ne preuzima automatski promenljivu životnu sredinu MYSQL_ROOT_PASSWORD iz okruženja wordpress_dbcontainer, kao što to radi WordPress slika. Mi zapravo moramo da kopiramo MYSQL_ROOT_PASSWORD: edureka iz wordpress_db kontejnera i da postavimo korisničko ime na root.

Sada pokrenite grupu aplikacija:

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

To je sve što treba da uradiš. Na ovaj način možete da dodate onoliko kontejnera koliko želite i da ih povežete na bilo koji način.

Sada u pregledaču idite na port 8080, koristeći javno IP ili ime glavnog računarskog programa, kao što je prikazano u nastavku:

```
localhost: 8080
```

Popunite ovaj formular i kliknite na install WordPress.

Po završetku, ponovo posetite IP adresu servera (ovog puta koristeći port 8181, npr. Localhost: 8181). Dočekaće vas phpMyAdmin ekran za prijavljivanje:

Unapred se prijavite koristeći osnovno korisničko ime i lozinku koje ste postavili u YAML datoteci i moći ćete da pregledate bazu podataka. Primetićete da server uključuje WordPress bazu podataka koja sadrži sve podatke iz vaše WordPress instalacije.

Doker Kompon je alatka za definisanje i pokretanje više kontejnera za pokretanje složenih aplikacija u Dokeru, na primer kontejnerizovanje MEAN aplikacije.

Videli smo šta su Dokerove slike, Doker tenkovi i kakva je potreba za njima.

Nakon istraživanja prilika koje Doker ima, svakako je veoma uzbudljivo saznati više.

Uvod u Docker Compose

Sadržaj jedne komunalne aplikacije je bio jednostavan za mene. Ali kada sam morao da obuzdam više službi u odvojenim kontejnerima, naleteo sam na prepreku. Moj zahtev je bio da obuzdam i ugostim aplikaciju MEAN steck.

Da, dobro ste pročitali. Aplikacija sa punim nizom. Prvo sam mislio da to nije moguće. Ali nakon što sam čuo za Dokera Kompona, znao sam da će svi moji problemi biti rešeni.

Doker Kompon može da se koristi za kreiranje zasebnih kontejnera (i hostovanje) za svaki niz u aplikaciji MEAN stek. MEAN označava MongoDB Express Angular & NodeJs.

Koristeći Docker Compose, svaku od ovih tehnologija možemo postaviti u zasebne kontejnere na istom domaćinu i naterati ih da međusobno komuniciraju. Svaki kontejner će izložiti vezu za komunikaciju sa drugim kontejnerima.

Komunikaciju i vreme održavanja ovih kontejnera održavaće Doker Kompon.

Dokerfile

Slično tome kako okrećemo bilo koji kontejner za jednu aplikaciju pisanjem doker fajla, moraćemo da napišemo zasebnu doker datoteku da bismo napravili svaku

aplikaciju sa jednim kontejnerom. Pored toga, moraćemo da napišemo i Docker Compose fajl koji će uraditi stvarni posao. Datoteka "Doker sastavi" će izvršiti različite doker datoteke da bi kreirala različite kontejnere i omogućila im međusobno interakciju.

U našem slučaju imamo punu stek aplikaciju koju čine MongoDB, ExpressJS, Angular i NodeJS. MongoDB vodi računa o bazi podataka u pozadini, NodeJS i ExpressJS su za generaciju na strani servera, a Angular je za front.

Pošto postoje tri komponente, za svaku komponentu moramo da zavrtimo rezervoare. Moramo da zavrtimo rezervoare na sledeći način:

1. **Rezervoar 1 - ugao**
2. **Tenk 2 - NodeJS i ExpressJS**
3. **Tenk 3 - MongoDB**

Pravljenje dokerskih tenkova

Kao prvi korak za dokerizaciju posredničke aplikacije, napišimo dokerfile da napravimo svaku od komponenti, počevši od Angulalnog rezervoara. Ova datoteka dokera mora biti prisutna u direktorijumu projekta zajedno sa datotekom "package.json". 'Packa.json' sadrži detalje o tome koju verziju zavisnosti treba da koristi 'NPM' za izradu ugaone aplikacije.

1. Dokerfile za prednji kraj

```
OD noda: 6 RUN mkdir -p /usr/src/app WORKDIR/usr/src/app COPY package.json/usr/app
RUN npm cache clean RUN npm install COPY. / usr / src / app EXPOSE 4200 CMD ['npm',
'start']
```

Kao i uvek, naša prva komanda je da prevučemo osnovnu sliku, a osnovnu sliku prevučemo 'čmikom: 6'.

Sledeće dve komande se odnose na kreiranje novog direktorijuma "/usr/src/app" unutar kontejnera "Doker" da bi se sačuvali ugaoni kodovi i ovaj pretvorio u radni direktorijum unutar kontejnera.

Zatim kopiramo 'package.json' fajl iz našeg projektnog direktorijuma u unutrašnjost kontejnera.

Zatim pokrenemo komandu "npm keš čista" koja čisti npm keš memoriju.

Nakon toga pokrećemo komandu 'npm install', koja počinje da preuzima kotlovske ploče potrebne za hostovanje Angular aplikacije. Počinje preuzimanjem kotlarskih ploča na osnovu verzija zavisnosti navedenih u 'package.json'.

Sledeće pokretanje komande "RUN" je kopiranje svih kodova, fascikli prisutnih iz direktorijuma projekta u kontejner.

Gorenavedena komanda traži od kontejnera da izloži port broj 4200 kako bi komunicirao sa backend serverom kako bi poslao zahteve korisnika koji pristupaju front klijentu putem Veb korisničkog interfejsa.

Konačno poslednja komanda: "RUN" je komanda za pokretanje "npm". Ovo počinje sa izvršavanjem kodova za izradu naše Angular aplikacije.

Angularna aplikacija je sada spremna, ali neće biti pravilno hostovana zbog zavisnosti od backend servera i baze podataka. Hajde da nastavimo dalje i napišemo doker fajl da kontejnerizujemo back-end server.

2. Dokerfile za zadnji deo

Čak će i ova datoteka dokera biti prisutna u direktorijumu projekta. Ovaj direktorijum će takođe sadržati datoteku 'package.json' za definisanje zavisnosti Express servera i drugih NodeJS zahteva. Ali najvažnije, sadrži kôd projekta koji podržava back-end server.

```
OD noda: 6 RUN mkdir -p /usr/src/app WORKDIR/usr/src/app COPY package.json/usr/app
RUN npm cache clean RUN npm install COPY. / usr / src / app EXPOSE 3000 CMD ['npm',
'start']
```

Kao što vidite, postoji mnogo sličnosti između dva dokerfila. Koristimo isti 'node: 6' kao osnovni sloj slike, kreiramo novi direktorijum unutar kontejnera, konvertujemo ga u radni direktorijum i između ostalih pokrenemo komandu 'npm install'. Međutim, jedina razlika je u broju luka koje su izložene komunikaciji. U ovom slučaju, port broj 3000 je definisan. Server će biti uslužen ovde i tražiće zahteve klijenata.

3. Baza podataka

Možda se pitate zašto nisam pomenuo "dokerfile za bazu podataka" u naslovu. Razlog je taj što nam ne trebaju nikakva prilagođavanja. Možemo odmah da povučemo osnovnu sliku 'MongoDB' za skladištenje podataka i samo izložimo broj porta gde mu se može pristupiti.

Sad bi imao pitanje na umu gde bih to mogao da uradim? Pa, možemo to da uradimo u dosijeu Docker Compose.

Doker novi fajl

Docker Compose Datoteka je YAML datoteka koja sadrži detalje o uslugama, mrežama i volumenima za podešavanje Docker aplikacije.

Pokrenite dole navedenu komandu da biste pronašli verziju doker mašine.

```
doker -v
```

Izvršavanje komande će vratiti verziju pokrenutu na domaćinu. Na osnovu verzije Docker Engine-a na vašem domaćinu, preuzmite odgovarajuću verziju Docker Compose-a. Možete potražiti odgovarajuću verziju za preuzimanje*Dokerova zvanična dokumentacija* .

Od kada pokrenem Docker Engine verziju 17.05.0-ce, koristio sam Docker Compose verziju 3.

Instaliranje pisanja dokera

Da biste preuzeli Komandu, pokrenite dole navedeni skup komandi.

```
sudo curl -L https://github.com/docker/compose/releases/download/1.16.1/docker-
compose-'uname -'uname -m' -o/usr/local/bin/docker- compose sudo chmod + x / usr /
local / bin / docker-compose
```

Imajte na kraju da će se broj verzije u komandi promeniti na osnovu verzije Doker mašine koju koristite.

Ispod su komande koje su deo mog Dosijea Doker Kompon.

```
version: '3.0' # specify docker assembly version # Define services / containers to
run services: angular: # first service build name: angular-app # specify dockerfile
port directory: - '4200: 4200 ' # navedite port mapping express: # ime druge
konstrukcije usluge: express-server # navedite dockerfile port direktorijum: - '3000:
3000' # navedite veze mapiranja portova: - baza podataka # povežite ovu uslugu sa
bazom podataka usluge baze podataka:  # naziv treće slike usluge: mongo # navedite
sliku za izgradnju kontejnera iz luka: - '27017: 27017' # navedite prosleđivanje
porta
```

Prilično sam siguran da vam komande u gorenavedenom dosijeu nemaju smisla. Hajde da rešimo taj problem.

U prvom redu koda definisao sam verziju Docker Compose koju koristim. Ovo je veoma važan korak ako želite da Funkcija "Sastavi" funkcioniše ispravno bez greške. Obavezno preuzmite Verziju Docker Compose u skladu sa verzijom Docker Engine- a.

Nakon toga, definisao sam tri kontejnera koristeći ključnu reč 'usluge'. Ove usluge se primenjuju na tri komponente mog otuda, prednji kraj, zadnji deo i bazu podataka. Dakle, u ovom slučaju, ime mojih kontejnera će biti ime mojih usluga, tačnije. "Ugaona", "ekspres" i "baza podataka".

Ključna reč "build" se koristi da označi da je doker fajl za okretanje tog kontejnera prisutan u tom direktorijumu.

Jednostavno je. Putanja treba da bude navedena nakon "build:". U našem slučaju, 'angular-app' i 'express-server' su putanje do dva direktorijuma do kojih se može doći iz direktorijuma gde se nalazi Docker Compose fajl. Za naš kontejner baze podataka sam jednostavno rekao da koristite osnovnu 'sliku: mongo' umesto putanje do dokerfile.

Za svaku od ovih usluga naveo sam i brojeve portova koji se mogu koristiti za prijem/slanje zahteva iz drugih kontejnera (usluga). 4200 u slučaju angular, 3000 u slučaju ekspresa i 27017 u slučaju mongo.

Pored toga, ekspresni kontejner ima 'vezu:' sa kontejnerom baze podataka, što znači da će svi podaci primljeni na strani servera biti poslati u bazu podataka gde će biti uskladišteni.

Sada smo konačno na kraju postavke Kompone. Da bismo pokrenuli Docker Compose i zavrteli tri kontejnera sa tri servisa, moramo jednostavno izvršiti dve dole navedene komande iz direktorijuma gde se nalazi Docker Compose fajl (YAML fajl)

```
docker-compose build docker-compose up
```

Komanda "doker-sastavi građu" se koristi za izradu/vraćanje usluga u prethodno stanje, dok se komanda "doker-sastavi" koristi za kreiranje/pokretanje kontejnera. Samo napred! Probajte sami.

Ispod su snimci ekrana Dokerovih slika koje su napravljene, a zatim izvršene. Možda ćete primetiti da se kreira ugaona slika koja je zatim označena kao "ugaona: najnovija".

Takođe, slika za Express je izgrađena sa imenom i nalepnicom kao 'express: latest'.

Sada kada je slika izgrađena, hajde da pokušamo da je pokrenemo i tako zavrtimo kontejner u tom procesu. Ispod je taj snimak ekrana.

Ispod je snimak ekrana 'webpack: uspešno sastavljen', što znači da Doker uspešno kontejnerizuje tri servisa.

Sada kada su kontejneri hostovani, usluge koje su aktivne možete videti u njihovim portovima. Otkucajte sledeće brojeve portova u Veb pregledač da biste imali interakciju sa GUI aplikacijom MEAN.

localhost: 4200 - *Angular application (front)*
localhost: 3000 - *Express Server & NodeJS (background/server side)*
localhost: 27017 - MongoDB (database)

Doker još nije završio! Pomoću komande 'doker-sastavi skalu = 'x" možemo lako da povećamo / smanjimo broj podešavanja. Drugim rečima, možemo da napravimo toliko uslužnih kontejnera. Ispod je kompletna komanda za razmeru određene usluge na kontejnere '5':

```
docker-compose skala = 5
```

Tako da su usluge lakog podešavanja, pakovanja i kontejnerizacije na tako isplativ način ono što Dokera čini jednim od najboljih instalacionih alata i mojim ličnim favoritom.

Docker umrežavanje - istražite kako kontejneri međusobno komuniciraju

U današnjem svetu, preduzeća su se zainteresovala za kontejnerizaciju koja zahteva jake mrežne veštine za pravilno konfigurisanje kontejnerske arhitekture, što stoga uvodi koncept **Docker Networkinga**.

Šta je Doker?

Da biste razumeli Dokera, potrebno je da znate o istoriji načina na koji su aplikacije ranije bile raspoređene, a zatim kako su aplikacije sada raspoređene pomoću kontejnera.

Na stari način su se videle aplikacije na domaćinu. Tako n broj aplikacija deli biblioteke prisutne u tom operativnom sistemu. Ali, uz kontejnerizaciju, operativni sistem će imati jezgro koje će biti jedino što će biti zajedničko svim aplikacijama. Tako da aplikacije ne mogu da pristupe bibliotekama jedna drugoj.

TakoDoker Jednostavnim rečima, otvorena platforma za razvoj, isporuku i pokretanje aplikacija, koja omogućava korisniku da odvoji aplikacije od infrastrukture uz pomoćKontejnera za brzu isporuku softvera.

Kako ti kontejneri komuniciraju jedni sa drugima u raznim situacijama?

Pa, dolazi preko Docker Networkinga.

Docker umrežavanje

Pre nego što zaronim u Docker Networking, da vam pokažem Doker tok posla.

Projektant upisuje šifru koja propisuje zahteve aplikacije ili zavisnosti u doker fajl koji je lako napisati, a ovaj Doker fajl daje doker slike. Dakle, kakva god da je zavisnost potrebna za određenu aplikaciju, prisutna je na ovoj slici.

Dokerovi tenkovi nisu ništa drugo do instanca Dokera tokom bekstva. Ove slike su okačene u Docker Hub (Git Docker skladište slika) koje sadrži javna/privatna skladišta.

Dakle, takođe možete da povučete sliku iz javnih depoa, a slike možete da otpremite u Doker čvorište. Zatim, iz Docker Hub-a, razni timovi kao što su garancija kvaliteta ili produkcijski timovi će izdvojiti tu sliku i pripremiti sopstvene kontejnere. Ovi pojedinačni kontejneri međusobno komuniciraju preko mreže da bi izvršili neophodne radnje, što nije ništa drugo nego **Docker Networking**.

Stoga možete definisati **Docker Networking** kao komunikacioni prolaz kroz koji svi izolovani kontejneri međusobno komuniciraju u različitim situacijama kako bi izvršili neophodne radnje.

Šta mislite, koji su ciljevi Docker Networkinga?

Ciljevi docker umrežavanja

Fleksibilnost - Doker pruža fleksibilnost tako što omogućava bilo kom broju aplikacija da međusobno komuniciraju preko različitih platformi.

Cross-platforma Docker se lako može koristiti na unakrsnoj platformi koja radi na različitim serverima uz pomoć Docker Swarm Klastera.

Skalabilnost - Doker je potpuno distribuirana mreža koja omogućava individualni rast i skaliranje aplikacija uz istovremeno obezbeđivanje performansi.

Decentralizovano - Doker koristi decentralizovanu mrežu koja omogućava širenje i visoku dostupnost aplikacija. U slučaju da skladištu resursa iznenada nedostaje kontejner ili domaćin, možete da otvorite dodatni resurs ili da se prebacite na usluge koje su još uvek dostupne.

Korisnik - friendly - Doker olakšava automatizaciju uvođenja usluga, čineći ih jednostavnim za korišćenje u svakodnevnom životu.

Podrška - Doker nudi gotovu pomoć. Dakle, mogućnost korišćenja Docker Enterprise Edition i da sve funkcije dobiju veoma lako i jednostavno, čini Docker platformu veoma lakom za korišćenje.

Da biste omogućili gorena vedene ciljeve, potrebno vam je nešto što se zove "**Kontejnerski mrežni model**".

Kontejnerski mrežni model (CNM)

Pre nego što vam kažem šta je tačno model kontejnerske mreže, dozvolite mi da vas ukratko obavestim o **Libnetworku** koji je potreban pre nego što shvatite CNM.

Libnetwork je open source Docker biblioteka koja primenjuje sve ključne koncepte koji čine CNM.

Stoga **kontejnerski mrežni model (CNM) standardizuje** korake potrebne za obezbeđivanje kontejnera za umrežavanje koji koriste više mrežnih upravljačkih programa. CNM zahteva distribuirani ključ/vrednost skladišta kao konzola da bi sačuvao mrežnu konfiguraciju.

CNM ima interfejse za IPAM dodatne komponente i mrežne dodatke.

IPAM API dodatne komponente se koriste za kreiranje/brisanje depoa adrese i dodeljivanje/uklanjanje IP adresa kontejnera, dok se API-je mrežnih dodatnih komponenti koriste za kreiranje/brisanje mreža i dodavanje/uklanjanje kontejnera sa mreža.

CNM je uglavnom izgrađen na 5 objekata: **mrežni kontroler, upravljački program, mreža, endpoint i sandbox.**

Objekti kontejnerskog mrežnog modela

Mrežni kontroler: Obezbeđuje ulaznu tačku za Libnetwork koja izlaže jednostavne DOcker Engine API-je za dodeljivanje i upravljanje mrežama. Budući da Libnetwork podržava više ugrađenih i udaljenih upravljačkih programa, mrežni kontroler omogućava korisnicima da prilože određeni upravljački program datoj mreži.

Vozač: On je vlasnik mreže i odgovoran je za upravljanje mrežom učešćem više lansera koji zadovoljavaju različite slučajeve korišćenja i scenarije primene.

Mreža: Obezbeđuje povezivanje između grupe krajnjih tačaka koje pripadaju istoj mreži i izolaciju od drugih. Dakle, svaki put kada se mreža kreira ili ažurira, odgovarajući upravljački program će biti obavešten o događaju.

Endpoint: Omogućava vam da povežete usluge izložene kontejnerima u mreži sa drugim uslugama koje pružaju drugi kontejneri u mreži. Endpoint predstavlja uslugu, a ne obavezno određeni kontejner, endpoint takođe ima globalni opseg unutar klastera.

Sandbox: Kreiran kada korisnici zahtevaju kreiranje krajnje tačke na mreži. Sandbox može imati više krajnjih tačaka povezanih sa različitim mrežama koje predstavljaju mrežnu konfiguraciju kontejnera, kao što su IP adresa, MAC adresa, rute, DNS.

Dakle, to je bilo 5 glavnih subjekata CNM-a.

Sada ću vam reći različite programe za pokretanje mreže koji su uključeni u Docker umrežavanje.

Mrežni upravljački programi

Tu je uglavnom 5 mrežnih drajvera: Bridge, Host, None, Overlay, Macwlan

Bridge: Mrežna mreža je privatna podrazumevana interna mreža koju je kreirao doker na domaćinu. Dakle, svi kontejneri dobijaju internu IP adresu i ti kontejneri mogu da pristupe jedni drugima, koristeći tu internu IP adresu. Mreže mostova se obično koriste kada se aplikacije pokreću u samostalnim kontejnerima kojima je potrebna komunikacija.

Host : Ovaj program za pokretanje sistema uklanja mrežnu izolaciju između dokera domaćina i doker kontejnera za direktno korišćenje mreže domaćina. Sa ovim nećete moći da pokrenete više Veb kontejnera na istom domaćinu, na istom portu kao što je port sada uobičajen za sve kontejnere u mreži domaćina.

None : U ovoj vrsti mreže, rezervoari nisu povezani ni sa jednom mrežom i nemaju pristup spoljnoj mreži ili drugim kontejnerima. Dakle, ova mreža se koristi kada želite da potpuno onemogućite mrežni zrak na kontejneru i kreirate samo uređaj sa petljom povratnih informacija.

Overlay : Kreira unutrašnju privatnu mrežu koja se proteže kroz sve čvorove koji učestvuju u jatu rojeva. Tako Preklapanje mreža olakšava komunikaciju između rojenja službe i samostalnog tenka ili između dva samostalna tenka na različitim Doker demonima.

Macvlan: Omogućava vam da kontejneru dodelite MAC adresu, čineći je prikazanom kao fizički uređaj na mreži. Zatim, Docker demon usmerava saobraćaj prema kontejnerima prema njihovim MAC adresama. Macvlan upravljački program je najbolji izbor kada se od vas očekuje da budete direktno povezani sa fizičkom mrežom, a ne usmereni preko steka Docker host mreže.

Ok, to je bila sve što je teorija trebala da bi se razumelo Docker umrežavanje. Dozvolite mi da vam pokažem kako se mreže kreiraju i kako kontejneri međusobno komuniciraju.

Ruke dalje

Pod pretpostavkom da ste svi instalirali Dokera na svoje sisteme, imam scenario prezentacije.

Pretpostavimo da želite da uskladištite ime i ID kursa, za šta će vam biti potrebna web aplikacija. U suštini, potreban vam je jedan kontejner za web aplikaciju, a za pozadinu vam je potreban drugi kontejner kao što je MySQL, da MySQL kontejner treba da bude povezan sa kontejnerom veb aplikacije.

Kako bi bilo da izvedem gore navedeni primer praktično.

Obuhvaćeni koraci:

- **Inicijalizujte Docker Swarm da biste formirali roj grupe.**
- **Kreiranje preklopne mreže**
- **Kreiranje usluga za veb aplikacije i MySQL**
- **Povezivanje programa preko mreže**

Hajde da počnemo!

1. korak: Pokretanje roja dokera na mašini.

```
doker roj init --reklama-addr 192.168.56.101
```

Tag –reklamira-addr konfiguriše menadžerski node da objavi svoju adresu kao 192.168.56.101. Drugi čvorovi u roju moraju biti u mogućnosti da pristupe menadžeru na IP adresi.

2. korak: Sada, ako želite da se pridružite ovom menadžeru, iskopirajte vezu koju dobijate kada pokrenete roj na radnom čmuku.
3. korak: Kreiranje koordinatne mreže koja se preklapa.

```
docker network create -d overlay myoverlaynetwork
```

Gde je Myoverlay ime mreže, i -d dozvoljava Dokeru Dejmonu da radi u pozadini.

4.1. korak: Kreirajte webapp1 uslugu i koristite mrežu koju ste kreirali da biste podesili ovu uslugu preko klastera rojeva.

```
docker service create --name webapp1 -d --net myoverlaynetwork
-p 8001: 80 hshar / webapp
```

Gde -p je za prosleđivanje portova,hsharje ime naloga na Docker Hub-u, a webapp je ime web aplikacije koja je već prisutna na Docker Hub-u.

4.2. korak: Sada proverite da li je usluga kreirana ili ne.

```
doker servis ls
```

Korak 5.1: Sada kreirajte MySQL uslugu i koristite mrežu koju ste kreirali da biste podesili uslugu preko gomile rojeva.

```
docker service create --name mysql -d --net myoverlaynetwork
-p 3306: 3306 hshar / mysql: 5.5
```

Korak 5.2: Sada proverite da li je usluga kreirana ili ne.

```
doker servis ls
```

Korak 6.1: Nakon toga proverite koji kontejner radi na vašem glavnom čmiču i unesite kontejner hshar/webapp.

```
doker ps
```

Korak 6.2: Tako da možete da vidite da je samo webapp usluga na menadžeru. Dakle, uđite u kontejner za vebapp.

```
docker exec -it container_id bash nano var / www / html / index.php
```
Komanda docker ps će navesti oba kontejnera sa povezanim ID-om kontejnera. Druga komanda će dozvoliti ovaj kontejner u interaktivnom režimu.

7. korak: Sada promenite ime servera $ iz localhost u mysql i $ lozinku iz "u'edureka" i takođe promenite sve neophodne podatke baze podataka i sačuvajte indeksnu datoteku.php koristeći tastersku prečicu Ctrl + x i nakon toga y da biste sačuvali i pritisnuli enter.

Korak 8: Sada idite u mysql kontejner koji radi na drugom čmiču.

```
docker exec -it container_id bash
```
9. korak: Kada uđete u kontejner mysql, unesite dole navedene komande da biste koristili bazu podataka u MySQL-u.

Korak 9.1: Pristupite korišćenju mysql kontejnera.

```
mysql -u root -pedureka
```
Gde -u predstavlja korisnika, a -p je lozinka računara.

Korak 9.2: Kreiranje baze podataka u mysql-u koja će se koristiti za dobijanje podataka sa webapp1.

```
KREIRANJE HANDSON BAZE PODATAKA
```
9.3. korak: Korišćenje kreirane baze podataka.

```
KORISTI HandsOn
```
9.4. korak: Kreirajte tabelu u ovoj bazi podataka koja će se koristiti za pribavljanje podataka iz webapp1.

```
KREIRAJ TABELU detalji_tečaja (naziv_tečaja VARCHAR (10), id_kursa VARCHAR (11))
```
Korak 9.5: Sada izađite iz MySQL-a i kontejnerizujući komandu "**Izlaz**".

10. korak: Idite na pregledač i unesite adresu **kao localhost: 8001/index.php** . Ovo će otvoriti vašu Web aplikaciju. Sada unesite detalje kurseva i kliknite na Pošalji **upit** .

11. korak: Nakon što kliknete na dugme "Pošalji upit", idite do oglasa gde je pokrenuta MySQL usluga, a zatim unesite kontejner.

```
docker exec -it container_id bash mysql -u root -pedureka USE
HandsOn SHOW tableselect * from predmeta_detalji
```

To će vam pokazati rezultate svih kurseva, o kojima ste popunili detalje.