Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.9
DOI: https://doi.org/10.24867/12BE25Deket

POREĐENJE DOCKER SWARM I KUBERNETES

COMPARISON OF DOCKER SWARM AND KUBERNETES

Milan Deket, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *U radu su opisani osnovni pojmovi i načini korišćenja Docker, Docker Swarm i Kubernetes alata. Objašnjeno je šta je Docker i šta su kontejneri, kao i kako se koriste. U poglavljima Docker Swarm i kubernetes se opisuje kako upravljati kontejnerima u kompleksnijim rešenjima koja mogu da se nalaze na više servera kao i kako pratiti stanje celog sistema i time se braniti od neočekivanih grešaka, odnosno neočekivanog gašenja sistema.*

Ključne reči: *Virtuelizacija, kontejneri, skaliranje aplikacija, docker, kubernetes, docker swarm*

Abstract – *The basic terms and ways of using Docker, Docker Swarm and Kubernetes tools are described in this paper. It explains what the Docker is and what the containers are, and how they are used. The Docker Swarm and Kubernetes chapters describe how to manage containers in more complex solutions that can be located on multiple servers, as well as how to monitor the state of the entire system and thus to defend themselves from unexpected errors, or unexpected crashing of the system.*

Keywords: *Virtualization, containers, application scaling, docker, docker swarm, kubernetes*

1. UVOD

Docker je nastao 2013. godine kao o*pen source* projekat od strane kompanije *dotCloud*. U toku prve godine razvoja projekat je porastao do veličine da je kompanije zatvorena i otvorena nova pod nazivom *Docker, Inc*. Pola decenije kasnije *Docker* je postao standard za serviranje *web* aplikacija.

Sve je počelo sa virtuelizacijom, jer su serveri postajali sve jači hardverski i veliku količinu vremena su bili u stanju mirovanja odnosno bez zadataka. Tako su programeri počeli da postavljaju više operativnih sistema na jedan računar i nastala je virtuelizacija i virtuelne mašine.

Sledeći veliki talas je bio takozvani *cloud*. Kompanija Amazon je predstavila svoje *cloud* rešenje pod nazivom *Amazon Web Services* ili skraćeno *AWS*. To je bilo jeftino i jednostavno rešenje za ljude koji su hteli da brzo dobiju veliku kompjutersku moć i mogućnost skaliranja aplikacija, ali da je isto tako brzo i ugase ukoliko im nije potrebna koristeći konzolu na *AWS*-u.

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Aleksandar Kupusinac, vanr. prof.

Nakon *cloud*-a stižu *Docker* kontejneri. To je bila još jedna velika promena u razvoja i serviranja aplikacija. Ova promena je kreirana da pomogne i programerima, dok su prethodne najviše uticale na sistem administratore. Ovu promenu su sa istim naporom mogli da nauče i programeri i administratori, a pomagala je i jednima i drugima u njihovim poslovima.

2. DOCKER

U današnje vreme se sve vrti oko brzine. Brzina razvoja aplikacija, brzina razvoja biznisa i svih ostalih stvari koje jednoj kompaniji mogu omogućiti veći profit za što kraće vreme. Koristeći *Docker* mogu da se osete benefiti, ali isto tako ako niste upoznati sa *Docker*-om i ostalim alatima neophodnim za održavanje kontejnera, oni mogu da vaš proizvod, odnosno aplikaciju odvedu u propast ili da je ugroze.

Docker pokriva ceo životni ciklus razvoja aplikacija. Aplikacija koja sadrži *frontend*, *backend*, *worker*-e i druge slojeve ili pod-aplikacije ima veliki broj zavisnosti, tj. oslanja se na funkcionalnost nekih drugih manjih aplikacija, biblioteka ili tehnologija, a onda može da se desi da te manje aplikacije ili biblioteke takođe imaju svoje zavisnosti što sve zajedno stvara jedan začarani krug. Ukoliko delovi glavne aplikacije rade na različitim operativnim sistemima, to značajno otežava razvoj i održavanje. Postoje rešenja za ovaj problem, kao što su na primer korišćenje virtuelnih mašina, instaliranje više operativnih sistema na jedan računar ili korišćenje više računara za razvoj jedne aplikacije, što je najgore rešenje. Nakon što uspešno postavite sve što vam je neophodno za razvoj aplikacije, potrebno je isto postaviti i na test, kao i na produkcionom okruženju, a zatim i na mašinu svakog novog programera koji se pridruži na razvoju aplikacije. Tu dolazimo do gore pomenute stvari, tj. brzine. Vreme potrebno da se sve ovo namesti će koštati kompaniju. Tu svoju primenu pronalaze Docker i kontejneri.

Docker će omogućiti da na isti način pokrenete i testirate aplikacije nezavisno od operativnog sistema. U današnje vreme 80 odsto vremena programera se svodi na održavanje aplikacije. To znači da malo vremena ostaje za dodavanje novih funkcionalnosti. Korišćenjem *Docker*-a možemo smanjiti procenat koji se odnosi na održavanje i pretvoriti to vreme u vreme za dodavanje novih mogućnosti na postojećoj aplikaciji ili kreiranju nove aplikacije.

2.1. *Docker image* i kontejneri

Image predstavlja sve zavisnosti, biblioteke, kod i operativni sistem koji su neophodni da pokrenu aplikaciju. Kontejner je *image* koji je pokrenut, od koje je napravljen proces. Da bismo počeli da koristimo *image* i

kontejnere najlakše je uzeti *image* koji je spreman za pokretanje. Najveća kolekcija za *Docker* se nalazi na *Docker hub*-u. *Web* adresa za *Docker hub* je hub.docker.com.

"*Docker image* je datoteka koja se sastoji iz više slojeva. Ona se koristi da bi se pokrenuo kontejner koji je opisan u datoteci. Datoteka se zapravo sastoji iz setova komandi i definisanih verzija aplikacija koje treba da se pokrenu, a zavisi od računara na kojem se pokreću. Takođe može da sadrži razne sistemske biblioteke koje će biti neophodne za pokretanje kontejnera, kao i alate i druge datoteke. Jedan *Docker image* može da se pokrene u obliku više kontejnera." [1].

Svaki *Docker image* počinje sa praznim slojem poznatijim kao *scratch*. Nakon toga svaka komanda koja se izvrši predstavlja novi sloj. *Docker image* može da ima jedan ili više slojeva. Kada pravimo novi *docker image*, počinjemo sa praznim slojem, tj *scratch*-om. Svaki novi sloj ima svoju jedinstvenu *hash* vrednost koja je kreirana *SHA* algoritmom, odnosno funkcijom. Jedinstvena *hash* vrednost pomaže sistemu da uporedi dva sloja i proveri da li su dva sloja ista ili različita.

U trenutku kada u *dockerfile*-u imamo definisane druge *image*-e i možda dva ili tri druga *image*-a imaju definisan Ubuntu kao prvi sloj, tad *hash* vrednost i keširani *image*-i pomažu. *Docker* neće dva puta skinuti Ubuntu, nego će preko *hash* vrednosti prepoznati da je to isti sloj i samo ga iskoristiti onoliko puta koliko je neophodno, što znači da ukoliko odlučimo da Ubuntu bude početni *docker image* za više slojeva, tada čuvamo samo jednu kopiju Ubuntu *image*-a. Od ove funckionalnosti najviše benefitujemo tako što čuvamo prostor na *hard* disku, smanjujemo protok saobraćaja koji treba da ide preko mreže na internet i povećavamo brzinu pokretanja kontejnera.

Docker image se identifikuje preko *image id*-a, repozitorijuma i taga, odnosno oznake. Tagovi su slični kao i *git* tagovi. Sa *docker* tagom zapravo možemo da skinemo *docker image* sa odredjenog *commit*-a.

2.2. Poređenje kontejnera i virtuelne mašine

Često na internetu možemo naići na poređenja kontejnera i virtuelnih mašina. Sličnosti postoje, ali u suštini to su dve različite stvari. Zapravo ih je teško i porediti, jer se poprilično razlikuju. Kontejner je samo proces. Proces koji radi na postojećem operativnom sistemu. To je proces koji ima ograničene resurse unutar operativnog sistema i samim tim se značajno razlikuje od virtuelne mašine.

2.3. *Docker* mreža

Docker svojim unapred podešenim opcijama omogućava da se kontejneri lako startuju za ljude koji tek počinju sa učenjem i upoznavanjem *Docker* kontejnera, ali postoji veliki broj opcija koji može dodatno da se konfiguriše, kao što je na primer mreža unutar kontejnera.

Kada se pokrene kontejner, čak i bez navođenja kojoj mreži da pripada, on će biti uključen u mrežu koju je *Docker* za korisnika unapred kreirao pod nazivom *bridge* mreža. *Bridge* je privatna virtuelna mreža kreirana od strane *Docker*-a. Svaka mreža, koju je *Docker* unapred kreirao ili je korisnik naknadno napravio za sopstvene potrebe, a koristeći *Docker*, je priključena na *NAT Firewall* - operativnom sistemu računara na kojem je *Docker* pokrenut. *Docker* će uraditi svu potrebnu konfiguraciju da bi kreirani kontejneri mogli da se povežu na internet.

U prethodnih nekoliko primera sa *Nginx*-om prilikom pokretanja smo koristili opciju *-p* da bismo otvorili *port*, odnosno saobraćaj sa operativnog sistema na određenom portu preusmerili na određen *port* unutar kontejnera. Ukoliko želimo da kontejneri međusobno komuniciraju, nema potrebe da otvaramo *port*-ove i time stvorimo bezbednosne probleme. Dovoljno je samo da dva kontejnera koja treba da komuniciraju budu unutar iste mreže. Broj mreža koje korisnik može da kreira nije ograničen, tako da može da ima jednu mrežu po aplikaciji odnosno kontejneru, a isto tako kontejner ne mora biti priključen ni na jednu mrežu.

2.4. *DNS* u *Docker*-u

Veoma je važno znati da se korisnik ne sme osloniti na *IP* adrese kontejnera u mreži, obzirom da su one dinamičke tako da je krucijalno da se koriste nazivi kontejnera, jer se kontejneri konstantno pokreću, uništavaju, premeštaju i slično. To znači da se i *IP* adrese kontejnera menjaju. Ne možemo se čak ni osloniti da će iz minuta u minut *IP* adresa biti ista, jer može da se desi da je kontejner uništen ili da je greška izazvala kontejner da se stopira i onda će *Docker* da se pobrine da pokrene novi kontejner, ali ne možemo znati na kojoj *IP* adresi.

Zato *Docker* koristi *DNS* (*Domain Name System*). "*DNS* je, u osnovi, sistem koji pretvara imena računara (*hostname*) u *IP* adrese. DNS takodje obezbeđuje podatke i o serverima elektronske pošte na domenu (MX), početnom *DNS* serveru (*SOA*) i druge. *DNS* je zasnovan na hijerarhijskom principu i jedna je od osnovnih komponenti interneta." [2]. U unapred kreiranoj *bridge* mreži kontejneri mogu da pronađu jedni druge samo preko *IP* adresa, dok u mreži koju korisnici kreiraju mreža ima automatski podržanu *DNS* opciju, tako da se kontejneri mogu pronaći i preko imena, a ne samo preko *IP* adresa. Ukoliko korisnik ima dva kontejnera, jedan pod nazivom *mysql* i drugi pod nazivom *web-app*, *web-app* kontejner će moći da pristupi *mysql*-u pozivajući *mysql* sa *port*-om 3306, tačnije *mysql*:3306, ako je *mysql* pokrenut na svom podrazumevanom *port*-u.

2.5. *Dockerfile*

Dockerfile je datoteka koja u sebi sadrži navedene korake za kreiranje određenog *docker image*-a. *Dockerfile* podseća na *shell* skriptu, ali nije, to je potpuno drugačiji jezik koji je jedinstven za *docker*. Kada se kreira *dockerfile* podrazumeva se da tako i nazovemo datoteku, ali nije pogrešno ni da je nazovemo drugačije. Prilikom pokretanja se mora eksplicitno naglasiti naziv datoteke.

Prva komanda koju treba navesti je *FROM* komanda i ona se nalazi u svakom *dockerfile*-u i obavezna je. Koristi da se opiše od kojeg *docker image*-a će početi da se gradi *Docker* kontejner.

ENV komanda se koristi za podešavanje *environment* varijabli. Ova komanda je veoma bitna za kreiranje kontejnera jer je ona glavni način da se proslede *key-value* vrednosti za kreiranje kontejnera ili za već pokrenute kontejnere. Jedan od razloga zašto se *environment*

varijable preferiraju je zato što su podržane na svakom sistemu.

Red navođenja komandi u *dockerfile*-u je bitan s obzirom da ih *Docker* izvršava *top-down* pristupom, što znači da kreće od prve navedene komande i redom prelazi na sledeću navedenu komandu. U pozadini *Docker* izvršava *shell* komande unutar kontejnera.

RUN komanda može da pokreće *shell* komande čije izvršenje nam je neophodno unutar kontejnera. To može biti *updatepackage manager-a* operativnog sistema, instaliranje neke aplikacije ili otpakivanje određene datoteke.

2.6. Perzistencija podataka

Kontejneri su *immutable* i privremeni što znači da oni nikad ne menjaju stanje, nego kreiraju novo i nisu namenjeni da dugo rade dok ne budu uništeni. Ideja je da se kontejneri mogu u bilo kom trenutku odbaciti ili uništiti, dok se uvek može kreirati novi iz *docker image*-a. Ovaj koncept kontejnera da se ne menjaju i da budu privremeni nas tera da ne menjamo stvari kada se pokrenu.

Ako treba da se desi promena u konfiguraciji ili hoćemo da pokrenemo noviju verziju kontejnera onda ćemo stopirati i uništiti stare kontejnere i pokrenuti nove. Ovakav koncept ima svoje prednosti i mane. Šta se dešava ukoliko kontejner sadrži bazu podataka koju treba da sačuvamo? Nju ne možemo uništiti jer ćemo time izgubiti podatke važne za korisnike aplikacije.

U idealnom scenariju kontejneri ne bi trebali da sadrže te osetljive podatke koje bismo snimili u bazu u kombinaciji sa datotekama koje pokreću samu aplikaciju. Ovakav koncept je poznat kao *separation of concerns*.

Kontejneri kao što su *nginx* i *mysql* su već bili konfigurisani da čuvaju podatke sve dok kontejner nije uništen i samo restartovanje ili stopiranje ove dve vrste kontejnera ne bi uništilo podatke koji su kreirani za vreme njihovog rada. Pre 10 godina ovaj problem nije postojao jer su svi podaci bili sačuvani direktno na disk računara i nije bilo kontejnera, tako da sistem administratori nisu morali da brinu o ovom problemu.

Sve je samo po sebi bilo perzistentno. U doba kontejnera i automatskog skaliranja aplikacija ovo predstavlja jedinstven i velik problem. *Docker* nudi dva rešenja za ovaj problem:

1. data volumes
2. bind mounts

Prva opcija, *data volumes*, kreira specijalnu lokaciju izvan kontejnera na kojoj će se čuvati podaci koji zahtevaju perzistenciju. Ovo će sačuvati podatke iako obrišemo kontejner i omogućiti nam da posle tu lokaciju dodamo bilo kom drugom kontejneru. Kontejner to vidi kao putanju do datoteka.

Druga opcija nam omogućava da delimo direktorijum između *host*-a i *Docker*-a, odnosno između računara na kojem je pokrenut *Docker* kontejner i samog kontejnera. Za kontejner ova putanja će biti kao i bilo koja druga lokalna putanja, jer neće biti svestan da se zapravo taj direktorijum ili datoteka nalazi na *host*-u.

3. *DOCKER SWARM*

Velika prednost kontejnera je ta što aplikacije možemo da pokrenemo na bilo kojoj platformi i na bilo kojem *hardware*-u. Aplikacije se pokreću i na različitim *cloud* provajderima - od *AWS*-a, preko *Azure*-a, *Digital Ocean*-a, do *Google Cloud*-a. Problem nastaje kada imamo na stotine kontejnera koji su rasprostranjeni po raznim serverima. To stvara veliki problem za administratore. Ukoliko ste velika organizacija kao što je *Netflix* koji imaju jako veliki broj inženjera koji održavaju na hiljade kontejnera onda problem i nije toliko velik jer imate veliki broj ljudi koji će da prate stanja tih kontejnera i servera, ali ukoliko ste mala organizacija i imate samo jednog čoveka zaduženog za sve to, onda je problem znatno veći i odgovori na pitanja kako pokrenuti sve kontejnere, kako ugasiti samo neke, kako ih pokrenuti ponovo, kako ih obrisati i update-ovati postaju znatno komplikovaniji. Upravo kao odgovor na ovo pitanje 2016. godine kreiran je *Swarm* režim. To je *Swarm* koji danas poznajemo. Swarm je postojao i pre, za verzije koje su bile starije od 1.12, ali nije obavljao sav potreban posao. 2016. godine na konferenciji u vezi sa *Docker*-om koja se zove *Dockercon* objavljen je *SwarmKit*. *SwarmKit* je set biblioteka kojim je omogućen veliki broj novih funkcionalnosti za *Swarm* režim.

Swarm je zadužen za upravljanje grupom kontejnera (*cluster management*). Sastoji se od više *host*-ova koji koriste *Docker*, koji je pokrenut u *Swarm* režimu.

Nakon instalacije *Docker*-a, *Swarm* režim nije uključen. Potrebno je pozovati dodatnu komandu da bi se *Swarm* uključio. To je urađeno kako se ne bi uticalo na postojeći *Docker* sistem ili na alate koji su se oslanjali na *Docker*.

3.1. Vrste *node*-ova u *Swarm*-u

Node je instanca *Docker*-a koja je uključena u *Swarm*. Jedan ili više *node*-ova može biti pokrenuto na jednom *host*-u, odnosno računaru ili *cloud*-u. Za produkciju je uglavnom preporučljivo da se *node*-ovi nalaze na što više fizički odvojenih mašina. Za pokretanje *Swarm*-a zadužen je menadžer *node*. Menadžeri imaju tri zadatka:

1. Da održavaju stanje klastera
2. Da planiraju pokretanje servisa
3. Serviraju *HTTP API endpoint*-e za *Swarm*

Koristeći Raft, menadžeri uspevaju da održe konzistentno stanje celog *swarm*-a i svih servisa koji su pokrenuti u njemu. Za testiranje je u redu da imamo samo jednog menadžera, ali za produkciju je ipak potrebno više u slučaju da taj jedan menadžer prestane da radi, potrebno je imati druge koji će da preuzmu posao. Menadžeri mogu da rade posao kao *worker*-i. Menadžeri su zapravo *worker*-i sa dozvolom da kontrolišu *Swarm*. Menadžere možemo učiniti *worker*-ima, ali isto tako i *worker*-e možemo promovisati u menadžere.

3.2. Arhitektura menadžera *node*-a

Menadžer *node* se sastoji iz 5 glavnih delova, a to su: *API*, *Orchestrator*, *Allocator*, *Scheduler*, *Dispatcher*.

API je zadužen za prihvatanje komandi od klijenta, u našem slučaju terminala.

Orchestrator služi za evaluaciju stanja klastera. On upoređuje stanje koje je korisnik deklarisao da želi u odnosu na ono stanje u kojem je klaster trenutno. Ukoliko korisnik želi da doda jednu instancu nekog kontejnera, *orchestrator* će uporediti željeno stanje klastera, a to je da treba da ima jednu instancu pokrenutu, sa trenutnim stanjem u kojem na primer nema nijednu pokrenutu instancu. Njegov odgovor će biti da treba kreirati jednu instancu željenog kontejnera, tj. kreiraće novi zadatak koji će biti neophodno izvršiti. Zadatak u ovom trenutku još uvek nije dodeljen niti jednom *node*-u.

Allocator služi da dodeli resurse novom zadatku. On će uzeti komandu koju je korisnik uneo kroz *API*, na primer kreiranje novog servisa i novi zadatak koji je kreirao *orchestrator* i za njih će alocirati *IP* adresu.

Scheduler je zadužen za dodeljivanje zadataka *worker node*-ovima. *Scheduler* konstantno prati da li je kreiran neki novi zadatak. On će preuzeti zadatak, proveriti koji su *node*-ovi dostupni i imaju mogućnost da izvrše taj zadatak. Mogućnost izvršavanja nekog zadatka može da zavisi od resursa određene mašine, tj. da li ima dovoljno memorije, zauzetosti procesora, količine već pokrenutih kontejnera, itd. Nakon toga, ukoliko ima više *node*-ova koji mogu da izvrše zadatak, on će ih sortirati i uzeti onaj za koji misli da je najbolji za izvršenje tog zadatka.

Dispacther je komponenta na koju se svi *worker node*-ovi povežu i odgovaraju. Svaki *worker* kada želi da se poveže na *dispatcher* će prijaviti koliko resursa ima, koliko kontejnera je pokrenuto, itd. Takođe, konstantna komunikacija se dešava izmedju *worker*-a i *dispatcher*-a na menadžer *node*-u, takozvani *heartbeat*, da bi menadžer znao da li je *worker node* još uvek tu ili se ugasio. Njegova glavna uloga je prosleđivanje zadataka *worker node*-ovima. On će da proveri da li postoji neki novi zadatak koji je kreiran i za koji je određen koji *worker* treba da ga izvrši, ukoliko ima on će proslediti taj zadatak određenom *worker*-u.

4. KUBERNETES

Kubernetes je *open-source orchestrator* sistem za *Docker* kontejnere, što znači da kontroliše životni ciklus kontejnera na klasteru koji može da ima više fizičkih mašina. Može da pokrene više kontejnera na jednoj mašini, dok više mašina onda može da čini klaster. *Kubernetes* kao i *Docker Swarm* može da pokrene razne vrste aplikacija, ali postoje razlike u odnosu na S*warm*. Kontejnere možemo pokretati na određenom *node*-u zadajući parametre koje *node* treba da ispuni da bi se na njemu pokrenuo kontejner. Isto tako, kontejnere lako možemo prebacivati sa *node*-a na *node* ukoliko koristeći *Kubernetes*, na primer, treba da uradimo neku vrstu održavanja na jednoj od mašina.

Kubernetes projekat je podržan od strane *Google*-a i projekat se može naći na *Github*-u. *Google* koristi kontejnere duže od jedne decenije i svo iskustvo korišćenja i održavanja kontejnerizovanih aplikacija je iskorišćeno prilikom kreiranja *Kubernetes* projekta.

Kubernetes kao i *Docker Swarm* se može instalirati i pokrenuti na mnogim platformama. Na nekim platformama je podržan veći broj funckionalnosti. Platforme sa najboljom podrškom su *AWS* i *Google Cloud*. Na primer, na nekim platformama nisu podržani *volume*-i i eksterni *load balancer*-i, ali to ne znači da se *Kubernetes* ne može koristiti, nego da samo ove dve navedene funckionalnosti ne mogu.

5. ZAKLJUČAK

Kubernetes i *Swarm* nisu konkurentski alati. Oni omogućavaju isto krajnje rešenje, ali u zavisnosti od raznih faktora korisnik bi trebao da zna koji alat da koristi. *Kubernetes* je bolje koristiti u sledećim slučajevima:

1. Starije i stabilnije rešenje sa boljim *monitoring* opcijama
2. Za razvoj kompleksnijih aplikacija
3. Veliki broj sistema u klasteru

Sa druge strane, S*warm* je bolji za:

1. Korisnike koji imaju samo osnovni nivo znanja o *Docker*-u i ne žele previše vremena da potroše na instalaciju i konfiguraciju klastera
2. Razvoj manje kompleksnih aplikacija

Swarm pruža jednostavno rešenje koje omogućava korisniku da brzo kreira i pokrene svoj klaster, dok *Kubernetes* podržava veće i kompleksnije zahteve. *Swarm* je popularniji među programerima koji preferiraju brzo startovanje aplikacije na klasteru, bez puno konfiguracije, dok je *Kubernetes* korišćeniji u produkcionim rešenjima od strane visokoprofilisanih interenet kompanija koje su napravile popularne servise koje koriste veliki broj ljudi.

6. LITERATURA

[1] Docker image, https://searchitoperations.techtarget-.com/definition/Docker-image.

[2] Wikipedia, https://sr.wikipedia.org/wiki/DNS.

Kratka biografija:

Milan Deket rođen je u Subotici 1992. god. Master rad na Fakultetu tehničkih nauka iz oblasti Elektrotehnike i računarstva – Veštačka inteligencija odbranio je 2020.godine.
Kontakt: milandeket@gmail.com