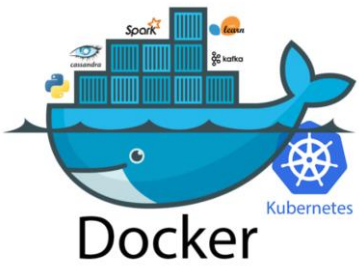


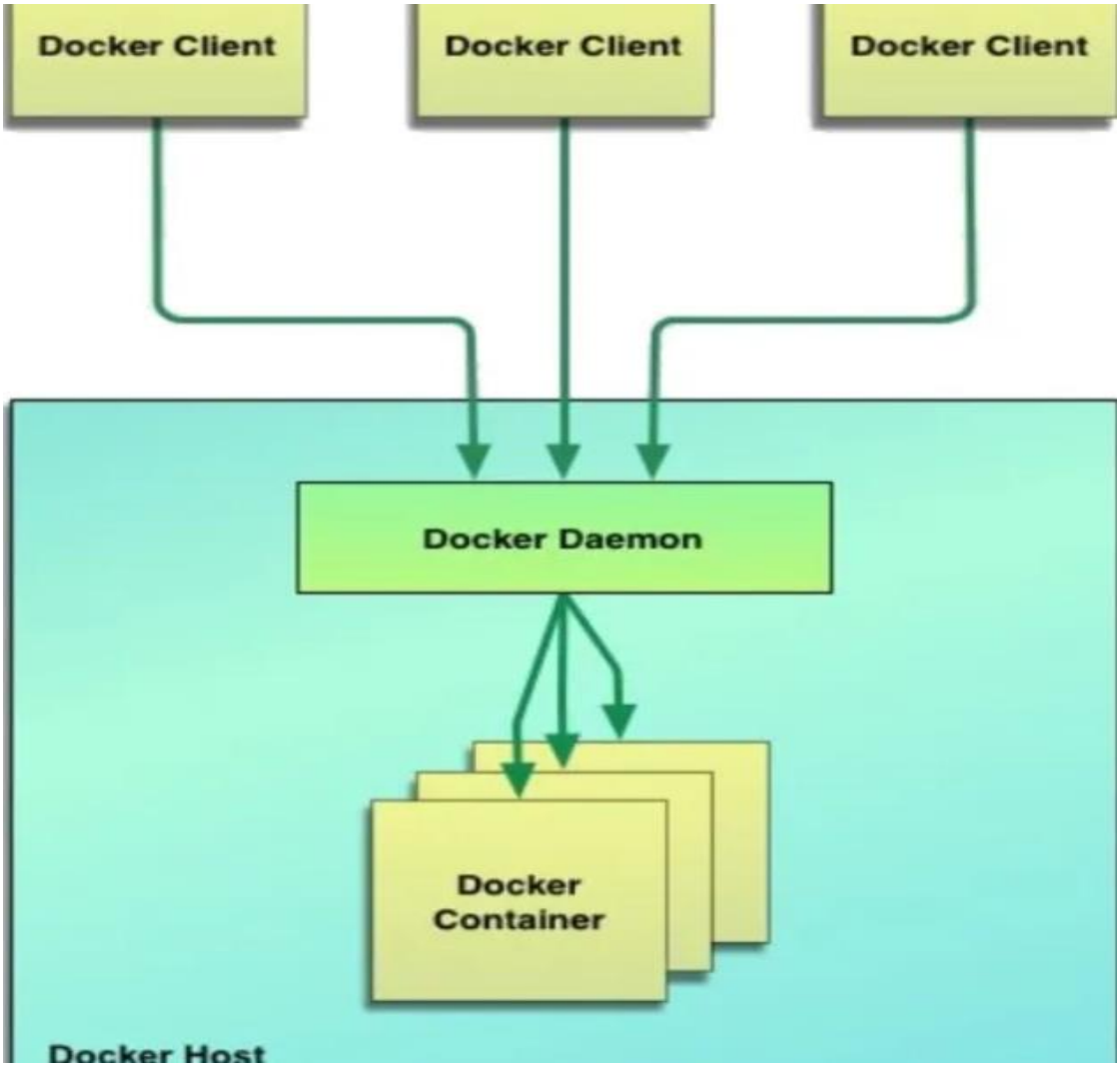
Docker i Kubernetes

Kontejneri se već duže vreme upotrebljavaju u računarskim naukama i za razliku od ostalih tipova virtuelizacije kontejneri se pokreću na vrhu kernela operativnog sistema. Zbog toga se virtuelizacija pomoću kontejnera često naziva i virtuelizacija na nivou operativnog sistema. Ovakav način virtuelizacije omogućava da više izolovanih instanca budu pokrenute na jednoj mašini.

Docker je aplikacija otvorenog koda koja automatizuje razvijanje aplikacija u kontejneru. Sa Dockerom, programeri brinu samo o aplikacijama koje su pokrenute unutar kontejnera, ne i o upravljanju kontejnerima.
Kontejneri se pokreću iz slika koje su fleksibilne, mogu da se dele, menjaju i one predstavljaju izvorni kod kontejnera. Docker pakuje slike u registre koji mogu da budu privatni i javni. Javne slike se nalaze na *Docker Hub-u* koji trenutno sadrži više od 10000 slika kreiranih i podeljenih od strane drugih ljudi.
Jedna od najvećih prednosti Docker-a je to što je veoma brz, većina kontejnera može da se startuje za manje od sekunde i jednostavne aplikacije mogu da se dokerizuju vrlo brzo. Takođe, Docker koristi *Copy on Write (COW)* model koji omogućava brzu izmenu aplikacija pokrenutih unutar kontejnera.

Copy on Write je tehnika koja je vrlo povoljna za optimizaciju. Zamisao je da se svima koji zahtevaju neki resurs dodeli samo pokazivač na njega sve dok neko pokuša da ga izmeni, tada se pravi kopija traženog resursa i dodeljuje onom koji je zahtevao promenu, na taj način ostali ne vide promene. Glavna prednost ove tehnike je to što se ne prave kopije dok ne dođe do izmene. Neki od fajl sistema koji koriste ovu tehniku su *Btrfs* i *ZFS*, takođe, ova strategija se koristi u *Microsoft SQL* serveru. Više o metodi COW može se naći na https://en.wikipedia.org/wiki/Copy-on-write.

Komponente Docker-a



Docker se sastoji od:

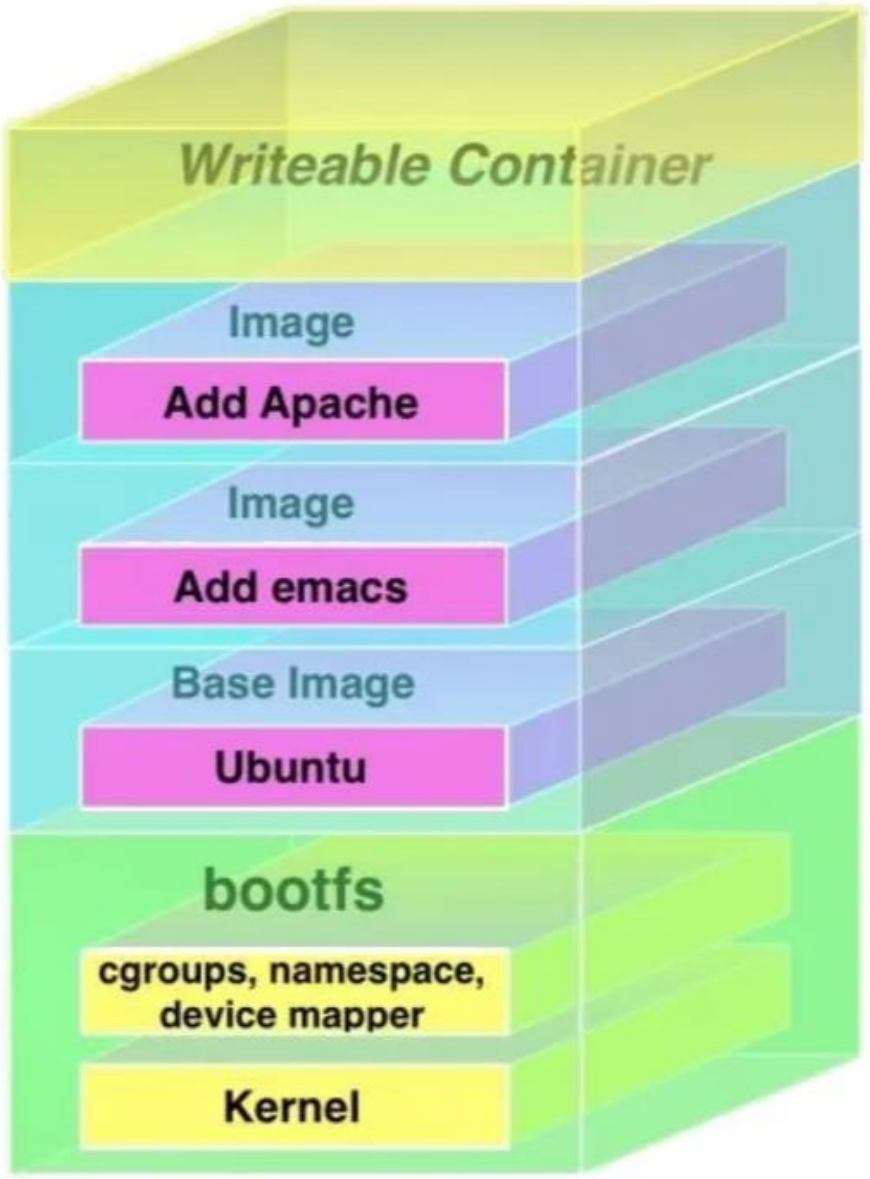
- **Klijenta i servera** – Docker je klijent-server aplikacija. Docker klijent komunicira sa serverom ili demonom koji radi sav posao u pozadini. Docker-om može da se upravlja preko komandne linije ili RESTful API-ja.
- **Slika** – gradivni elementi Docker-a. Kontejneri se pokreću na osnovu slika. Slike imaju slojevit format, koriste *Union* fajl sistem i nastaju izvršavanjem određenog skupa komandi. Slike se smatraju izvornim kodom kontejnera, portabilne su i mogu da se čuvaju, menjaju i dele.
- **Registara** – slike se čuvaju u registima. Postoje dva tipa registara: **privatni** i **javni**. Javni registar za slike odžava Docker i on se zove **Docker Hub**. Docker Hub podržava i privatne registre, svaki korisnik može privatno da sačuva svoju sliku i kasnije je preuzme uz upotrebu korisničkog imena i šifre.
- **Kontejnera** – se pokreću iz slika i mogu da sadže jedan ili više procesa u sebi. Docker kontejneri imaju isti koncept kao i pravi kontejneri, samo što umesto materijalnih stvari sadrže softver. Docker ne pravi razliku između kontejnera, nije bitno odakle je kontejner došao i šta se u njemu nalazi, svi kontejneri će biti pokrenuti na isti način.

Docker slike

Docker slike napravljene su od naslaganih fajl sistema. Na dnu se nalazi startni fajl sistem (*bootfs*), koji je tipični startni sistem Linux/Unix sistema. Korisnik Dockera nema nikakve dodirne tačke sa startnim fajl sistemom koji se kada se kontejner pokrene izbacuje iz memorije kako bi oslobodio mesto koje zauzima initrd. Iznad startnog fajl sistema nalazi se root fajl sistem (*rootfs*) koji može da da bude jedan ili više operativnih sistema. Root fajl sistem se postavlja (*mount*) tako da dozvoljava samo čitanje, ne i pisanje. *Union mount* dozvoljava da u istom trenutku bude postavljeno više fajl sistema i da se svi vide kao jedan fajl sistem, tako da krajnji fajl sistem može da sadrži i dodate datoteke i direktorijume. Više o fajl sistemima koje Docker koristi može se naći u James Turnbull, The Docker Book, 2014. i https://docs.docker.com .

Docker sve ove fajl sisteme naziva slike. Slike mogu biti postavljene jedna na drugu, slika ispod neke druge slike se naziva roditeljska slika. Moguć je prolazak kroz sve slike, na dnu se nalazi poslednja, bazna slika.

Kada se kontejner startuje na osnovu slike, Docker postavlja još jedan fajl sistem na vrhu i omogućava čitanje iz njega i pisanje u njega, ovo je mesto gde se svi zahtevani procesi pokreću.



Prilikom startovanja kontejnera, fajl sistem na vrhu je prazan. Kada se dese neke promene one se postavljaju u ovaj sloj. Na primer, ako želimo da promenimo neki fajl, on će prvo biti iskopiran iz fajl sistema iz koga je dozvoljeno samo čitanje u fajl sistem na vrhu u kome je dozvoljeno i čitanje i pisanje, na ovaj način slika uvek ostaje ista.

Izlistavanje i preuzimanje Docker slika

Slike mogu da se izlistaju komandom **`docker images`.**

```
$ sudo docker images REPOSITORY  TAG IMAGE ID
CREATED VIRTUAL SIZE ubuntu  latest  c4ff7513909d    6
days ago  255.4 MB
```

U primeru se vidi jedna slika sa registra ubuntu. Slike mogu da se preuzmu iz registra komandom **`docker pull`.**

```
$ sudo docker pull ubuntu:16:04 16.04: Pulling from
library/ubuntu Digest: sha256:
c6674c44c6439673bf56536c1a15916639c47ea04c39d6296c5df93
8add67b54b Status: Downloaded newer image for
ubuntu:16.04
```

Ovom komandom povukli smo sa registra ubuntu sliku Ubuntu 16.04. Ako bi nakon ovoga pokrenuli **`docker images`** dobili bismo:

```
$ sudo docker images REPOSITORY   TAG        IMAGE ID
CREATED         VIRTUAL SIZE ubuntu       latest
5506de2b643b    3 weeks ago     199.3 MB Ubuntu
16.04      0b310e6bf058    5 months ago    127.9 MB
```

Kreiranje Docker slika

Docker slike mogu da se kreiraju na dva načina:

- Komandom **`docker commit`**
- Komandom **`docker build`**, uz korišćenje *Dockerfile-a*

Prvi način se koristi ukoliko postoji neki pokrenut kontejner, čije trenutno stanje želimo da zapamtimo kao sliku.

Drugi način je mnogo fleksibilniji zato što se u njemu nalazi set instrukcija na osnovu kojih se kreira Docker slika.

Dockerfile sadrži listu instrukcija sa argumentima, svaka instrukcija treba da bude napisana velikim slovima i praćena jednim argumentom.

Dockerfile komande:

- FROM – Postavlja baznu sliku. Svaki Dockerfile mora imati FROM kao prvu instrukciju.
- MAINTAINER – Omogućava postavljanje autora slike.
- RUN – Izvršava bilo koju komandu nad trenutnom slikom i čuva rezultat.
- CMD – Dockerfile može da sadrži samo jednu CMD instrukciju i njen zadatak je da postavi osnovno ponašanje kontejnera pri pokretanju.
- LABEL – Dodaje metapodatke slici
- EXPOSE – Otvara zadati port
- ENV – Postavlja promenljive okruženja
- ADD – Dodaje fajlove i direktorijume

Primer korišćenja Dockerfile-a:

```
FROM ubuntu:14.04 MAINTAINER Korisnik RUN apt-get
update RUN apt-get install -y nginx #prethodna komanda
se moze napisati i kao #RUN [ "apt-get", " install", "-
y", "nginx" ] RUN echo ’Text’ >
/usr/share/nginx/html/index.html EXPOSE 80
```

Slika na osnovu Dockerfile-a može da se napravi komandom:

```
$ docker build –t=”NazivSlike”
```

Docker kontejneri

Koncept kontejnera potekao je od Linux kontejnera, potnatijih kao *LXC*. *LXC* je metod koji omogućava pokretanje više izolovanih Linux sistema na jednom domaćinu. Neke od tehnologija koje omogućavaju LXC su *Namespaces* i *Kontrole grupe (cgroups)*.

Kontrolne grupe su deo Linux kernela koji omogućava ograničavanje upotrebe resursa grupi procesa. Resursi mogu da podrazumevaju procesor, memoriju, disk, mrežu itd. Namespaces je, takođe, deo Linux kernela koji omogućava izolaciju i virtuelizaciju resursa nad grupom procesa.

Za kreiranje Docker kontejnera koristi se **`docker run`** komanda.

```
docker run -i -t ubuntu /bin/bash
```

Pokrećemo komandu ***run*** sa dva parametra: ***-i*** i ***-t***. Parametar -i otvara *STDIN* kontejnera čak iako korisnik nije povezan na kontejner, dok parametar

-t povezuje *pseudo-ttt* sa kontejnerom koji korisnik kreira. To omogućava korisniku da ima interaktivnu školjku u kontejneru koji kreira.
U prethodnoj komandi rekli smo Docker-u da želimo da koristimo ubuntu sliku. Docker pokušava da nađe ovu sliku lokalno i ako ne uspe pokušaće da je povuče sa *Docker Hub-a*. Na kraju, Docker očekuje komandu koju treba da izvši prilikom startovanja kontejnera, mi smo zadali bin/bash, što bi trebalo da pokrene Bash školjku koja izgleda ovako:

```
root@containerID:/#
```

Za izlaz iz kontejnera koristimo **exit** komandu.
Posle izlaska iz kontejnera kontejner je zaustavljen ali i dalje postoji, možemo videti listu svih kontejnera komandom **docker ps -a.** Komanda docker ps prikazuje samo kontejnere koji su trenutno pokrenuti dok nam parametar -a omogućava da vidimo sve kontejnere. Pokretanje zaustavljenog kontejnera možemo da uradimo komandom **docker start** naziv_kontejnera.

U nastavku ovog teksta biće reči o **Kubernetes**-ima, njihovoj arhitekturi na visokom i niskom nivou. Takođe, dat je i primer korišćenja Docker-a i Kubernetes-a.

Kubernetes

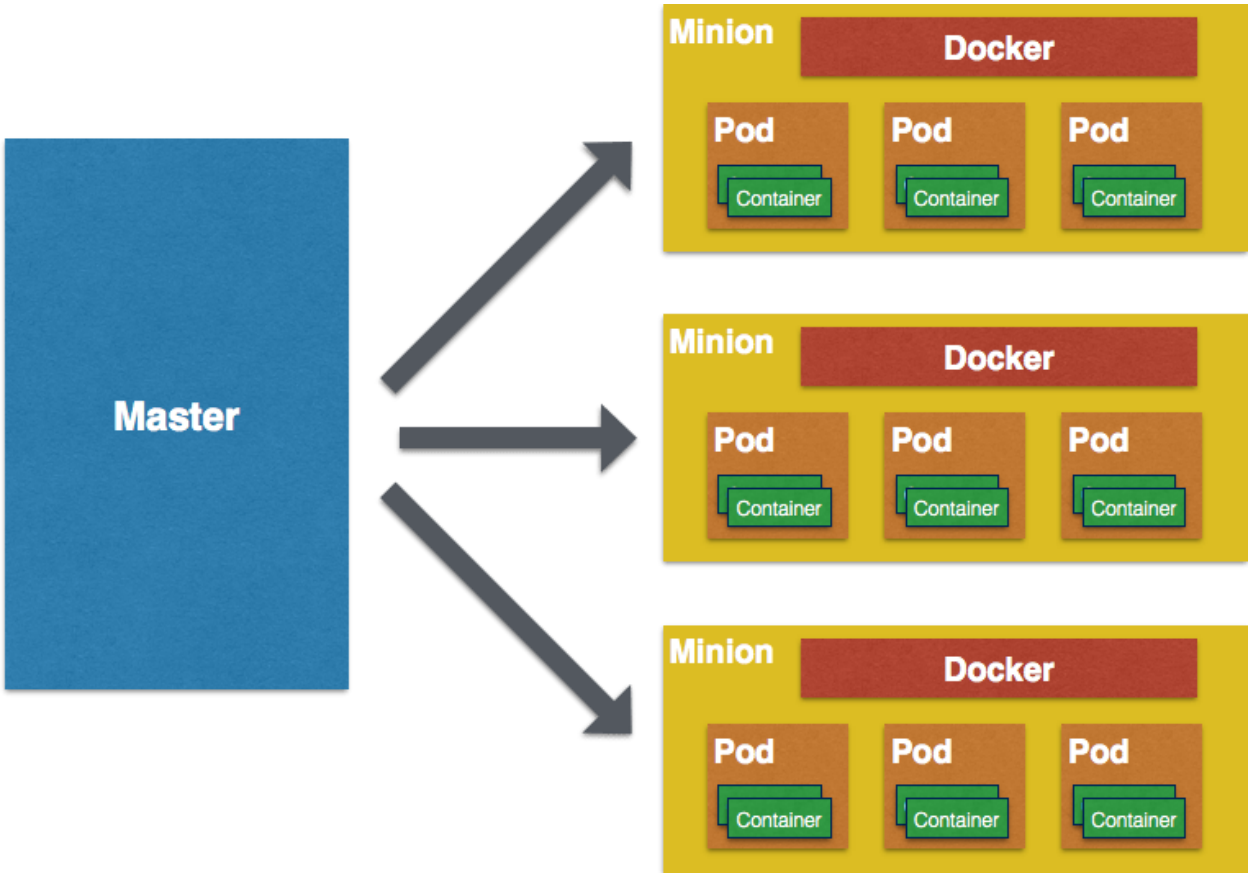
Kubernetes je platforma otvorenog koda razvijena od strane Google-a koja služi za automatizaciju operacija nad kontejnerima, kao što su razvoj, raspoređivanje i skalabilnost u klasteru.

Komponente Kubernetes-a su:

- **Podovi**
- **Lebele i selektori**
- **Replikacioni kontroleri**
- **Servisi**

Kubernetes Arhitektura

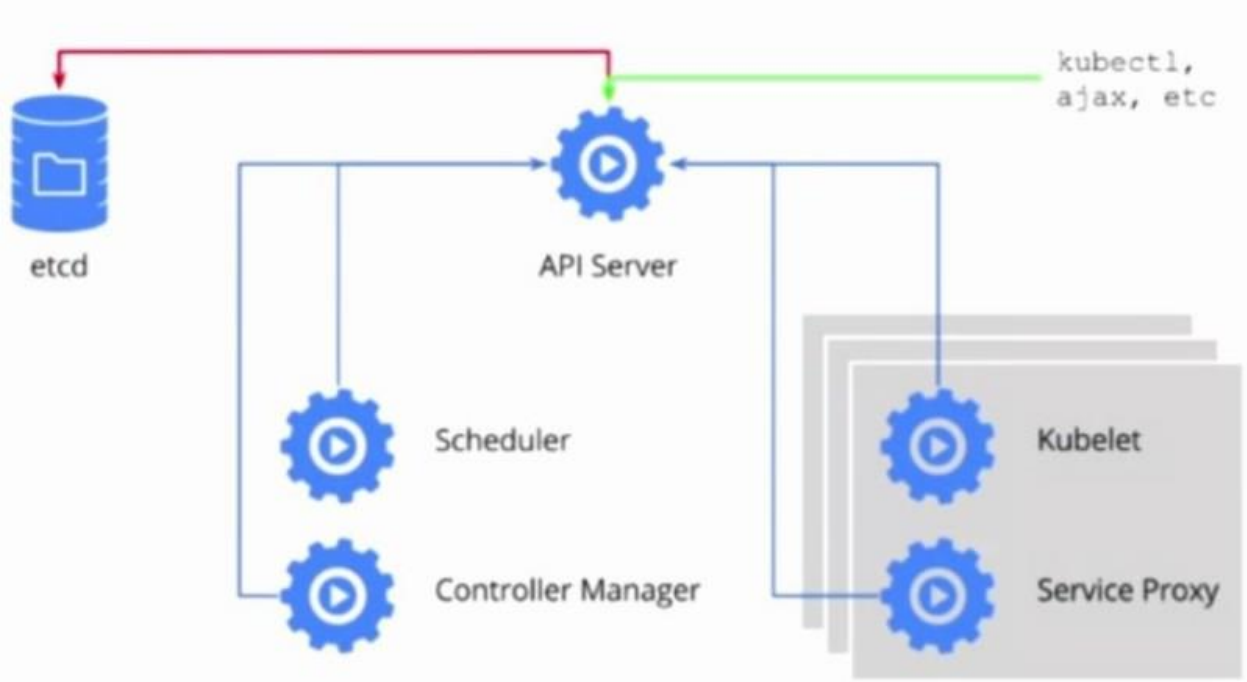
Na visokom nivou Kubernetes prati sledeću arhitekturu:



Na najvišem nivou Kubernetes se sastoji od:

- **Podova** – najmanja jedinica u Kubernetes-u.
- **Mastera** – centralna jedinica u klasteru.
- **Minioni** – radnici (slave) koji rade zadatke nametnute od strane Master-a.

Na niskom nivou Kubernetes prati sledeću arhitekturu:



- **API server** je server bez stanja (*stateless*) koji dozvoljava ostalim komponentama da koriste prikazane servise. API-ju se pristupa pomoću kubectl komande iz komandne linije, ili preko web interfejsa baziranom na AJAX tehnologiji.
- ***Kubelet*** je proces koji se pokreće na svakoj fizičkoj mašini i on upravlja Pod-ovima.
- ***Scheduler*** ili raspoređivač nalazi se samo na glavnoj mašini I on služi za raspoređivanje podova na ostale mašine koje se nazivaju radnici (slaves).
- ***Controller Manager*** je glavni servis koji pokreće Kubernetes. On održava jednu beskonačnu petlju koja nadgleda stanje celog sistema.

Više o arhitekturi Kubernetes-a može se naći na http://kubernetes.io/docs/.

Podovi su najmanje razvojne jedinice u Kubernetes-u. Svaki Pod može da sadrži jedan ili više kontejnera. Svi kontejneri koji se nalaze u istom Pod-u dele *host*, što

znači da dele IP adresu, portove i međusobno mogu da komuniciraju putem *localhost*-a.

Labele su parovi ključ/vrednost koji mogu da se postave na bilo koji objekat u Kubernetes-u(Pod, mašina u klasteru). Labele bi trebale da sadrže informacije bitne za korisnike. Mogu da se kreiraju pri pokretanju ili da se dodaju kasnije. Labele ne moraju da budu jedinstvene, čak se i očekuje da veliki broj objekata ima iste labele. Preko **selektora**, korisnik može da izabere grupu objekata. Trenutno postoje dva tipa selektora, selektor baziran na jednakosti (*Equality-based*) i selektor zasnovan na grupama (*set-based*).

Selektori bazirani na jednakosti služe za filtriranje po ključevima ili vrednostima. Podržavaju samo dva operatora, == i !=.

Selektori bazirani na grupama podržavaju tri operatora:

- *in* – filtrira labele tako da se ključ ili vrednost nalati u zadatoj grupi vrednosti
- *notin* – filtrira labele tako da se ključ ili vrednost ne nalati u zadatoj grupi vrednosti
- *exists* – samo jedna vrednos po kojoj se pretražuje.

Replikacioni kontroler je zadužen za to da određeni, zadati broj Pod-ova uvek bude pokrenut. Ako postoji više Pod-ova od zadatog broja Kubernetes će ubiti neke Podove, u suprotnom će ih kreirati. Za razliku od Pod-ova koji su manuelno pokrenuti, Pod-ovi pokrenuti od strane kontrolera će automatski biti zamenjeni ukoliko u nekom od njih dođe do greške, ili neki od njih bude obrisan.

Problem sa pod-ovima je to što kad se napravi nova replika, ne možemo da znamo koju će IP adresu dobiti i kako da joj pristupimo.

Ovaj problem rešavaju **Servisi**. Servisi određuju način na koji pristupamo pod-ovima. Servis može da opisuje jedan ili više podova i vezuje se na njih na osnovu selektora.

Primer korišćenja

Za pokretanje Kubernetes-a koristićemo:

- **VirtualBox**
- **Vagrant**

VirtualBox je softver otvorenog koda koji služi za virtuelizaciju operativnih sistema. *Vagrant* je softver koji kreira i konfiguriše virtuelnu mašinu. Predstavlja omotač oko VirtualBox-a i omogućava bržu i lakšu konfiguraciju virtuelne mašine.

Pokretanje Kubernetes-a vrši se na sledeći način:

- Kloniramo repozitorijum koji sadrži Kubernetes. To možemo da uradimo komandom:

```
git clone https://github.com/coreos/coreos-kubernetes.git
```

- Za pokretanje Kubernetes-a koristimo komandu vagrant up koja povlači sliku koja sadrži operativni sistem CentOS, pokreće je i pokreće Kubernetes u njoj.

Nakon uspešnog izvršavanja prethodnih komandi Kubernetes je startovan i spreman za upotrebu.

Kubernetes zadajemo kao argumente skripti kubectl.sh koja se nalazi u cluster folderu. Ovo možemo olakšati instalacijom *kubectl*-a. *Kubectl* binarni fajl treba da povučemo sa interneta i smestima u direktorijum **/usr/local/bin.** To možemo da uradimo komandom:

```
wget https://storage.googleapis.com/kubernetes-release/release/v0.17.0/bin
/darwin/amd64/kubectl -O /usr/local/bin/kubectl
```

Nakon uspešnog izvršavanja ove komande, binarni fajl koji sadrži kubectl nalazi se na zadatoj lokaciji i sve što treba da uradimo je da damo prava izvršavanja za njega. To možemo da uradimo komandom:

```
chmod +x /usr/local/bin/kubectl
```

Nakon ove komande možemo da upravljamo Kubernetes-om preko **kubectl-a.**

Pripremanje Docker slike

Za osvaj primer napravićemo Docker sliku koja sadrži jednostavan node.js server.

Primer Dockerfile-a:

```
FROM node:4-onbuild EXPOSE 8888
```

Za pravljenje slike na osnovu datog Dockerfile-a koristimo komandu:

```
docker build -t simple-server
```

Pre nego sto pokrenemo ovu komandu u direktorijumu u kom se nalazi Dockerfile mora da postoji package.json, koji sadrži start skriptu.

Primer jednostavnog `package.json` fajla:

```
{ "name": "test-server", "scripts": { "start": "node server.js" } }
```

U ovom fajlu kazemo da na start pokrene server.js. U server.js treba da se nalazi kod servera. Primer jednostavnog `server.js` fajla:

```
var http = require('http'); const PORT = 8888; function
handleRequest(request, response){ response.end('It works!'); } var server
= http.createServer(handleRequest); server.listen(PORT, function(){
console.log("Server listening on: http://localhost:%s", PORT); });
```

Dokumenova komande korišćene za pravljenje ovog jednostavnog servera mogu se naći na https://nodejs.org/api/.

Kad imamo sve potrebne fajlove i izvršimo komandu:

```
docker build -t dusanz/simple-server
```

Nakon toga sliku postavljamo na javni registar komandom:

```
docker push dusanz/simple-server
```

Docker će da napravi sliku sa zadatim serverom i nju možemo da pokrenemo komandom:

```
docker run -it --rm dusanz/simple-server
```

Nakon uspešnog izvršenja ove komande, možemo pristupiti serveru na adresi: `IP_KONTEJNERA:8888`.

Pripremanje fajlova za Kubernetes

Replikacioni kontroleri imaju ***yaml*** format.
Primer `node-server-replica.yaml` fajla, koji definiše jednu repliku:

```
apiVersion: v1 kind: ReplicationController metadata: labels: name: simple-
server name: simple-server spec: replicas: 1 template: metadata: labels:
```

```
name: simple-server spec: containers: - image: dusanz/simple-server name:
simple-server ports: - name: simple-server containerPort: 8888 hostPort:
8888
```

Ovaj replikacioni kontroler možemo da primenimo komandom:

```
kubectl create -f node-server-replica.yaml
```

Nakon ove komande Kubernetes će obezbediti da u svakom trenutku u klasteru postoji tačno jedna instanca pokrenutog servera.

Servisi takođe imaju *yaml* format. Definisaćemo servis koji za pod-ove sa labelom `simple-server` preusmerava saobraćaj koji dolazi na port 80 na port 8888.

```
apiVersion: v1 kind: Service metadata: name: sipmple-server labels: name:
simple-server spec: type: NodePort ports: - port: 80 targetPort: 8888
nodePort: 31400 protocol: TCP selector: name: simple-server
```

Servis kreiramo komandom:

```
kubectl create -f node-server-service.yaml
```

Zaključak

Docker i Kubernetes našli su veliku primenu kod serverskih aplikacija od kojih se zahteva da imaju dobar odziv i rade na velikom broju mašina. Kubernetes, koji je razvijen od strane Google-a, je u početku imao velike probleme sa produkcijom, najviše zbog toga što je najveći broj mašina kojima je mogao da upravlja bio 500. Nakon verzije 1.2 ovo ograničenje je ukinuto i Kubernetes počinje sve češće da se koristi u produkciji.

Korisni linkovi

- James Turnbull, The Docker Book, 2014.
- https://en.wikipedia.org/wiki/Copy-on-write
- https://docs.docker.com
- http://kubernetes.io/docs/
- https://nodejs.org/api/