Alati za razvoj softvera

Docker

Univerzitet u Novom Sadu
Fakultet Tehničkih Nauka

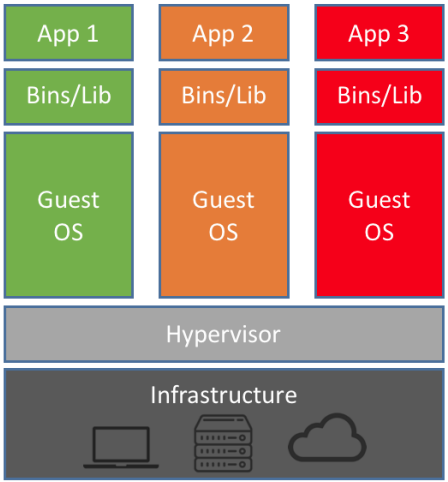
Uvod

- Sa pojavom virtuelnih mašina (VM), omogućeno je izbegavanje situacija gde se fizički serveri koriste na takav način da je iskoristivost resursa vrlo mala, što je u prošlosti često bio slučaj (iskoristivost resursa često bude od 10-20%)
- Svaka VM-a uključuje punu kopiju operativnog sistema, aplikacije, biblioteke
- Ispod njih nalazi hypervisor, odnosno softver koji omogućuje kreiranje, pokretanje i izvršavanje više VM-a na jednom fizičkom računaru — deljenje fizičkih resursa (memoriju, procesor)
- Dakle, virtuelne mašine (virtuelni serveri) su jeftiniji od fizičkih servera, s obzirom da troše deo resursa istog
- Pored manje cene, omogućuju lakše upravljanje, bolje skaliranje, konzistentno okruženje za izvršavanje aplikacija što ih čini odličnom podlogom za pružanje usluga web servisa (as a service model — cloud)

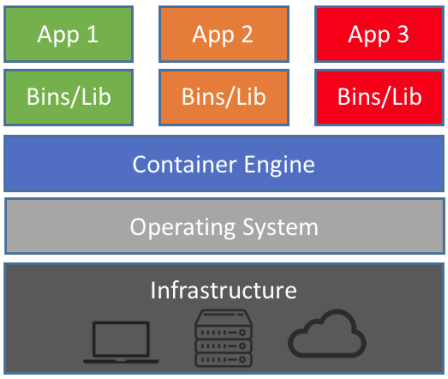
- Negativna strana VM-a je potpun underlying OS što znači da će se deo resursa koristiti za podizanje i izvršavanje OS-a
- VM sa kompletnim OS-om donose dodatan overhead u vidu performansi, liceni i administracije
- Nije redak slučaj da će više VM-a koristiti isti underlying OS i mi nemamo mogućno st da *ponovo iskoristimo* postojeće resurse
- Pored toga pošto VM sadrži potpun OS, vreme startovanja je u nekim situacijama je dosta visok
- Vremenom je dovelo do pitanja da li možemo *spakovati* više servisa na isti hardware

Kontejneri

- Za razliku od VM-a, kontejneri se oslanjaju na jedan host OS (najčešće Llinux) i dele njegov kernel
- Samim tim su dosta lakši i brži za pokretanje, nego VM
- Kontejnerske tehnologije **nisu** nova tehnologija i u UNIX/Linux svetu su prisutne dosta dugo
- Osim u velikim firmama (Google, Facebook, ...) nisu bile previše korišćene jer je kreiranje i upravljanje kontejnerima bilo realtivno kompleksno
- Docker je promenio ovu situaciju
- Docker se oslanja na nekoliko specifičnih stvari: namespaces, cgroups, unionFS
- Docker nije jedni engine za rad sa kontejnerima ali je najpoznatiji



Machine Virtualization



Containers

(VMs vs Containers)

Uvod

- Docker je open-source projekat koji nam omogućava da relativno rednostavno upravljamo kontejnerima
- Docker omogućava realtivno laganu komuniakciju sa container mehanizmima operativnog sistema (Linux-a)
- Njegove osnovne komponente su:
  - Docker engine
  - Docker images
  - Docker containers
  - Docker hub
  - Docker compose
  - Docker swarm

Docker slike

- Docker slike predstavljaju skup read-only layer-a, gde svaki sloj predstavlja različitosti u fajlsistemu u odnosu na prethodni sloj pri čemu uvek postoji jedan bazni (base) sloj
- Upotrebom storage driver-a, skup svih slojeva čini root filesystem kontejnera, odnosno svi slojevi izgledaju kao jedan unificirani fajlsistem
- Svi ovi read-only slojevi predstavljaju osnovu za svaki kontejner koji se pokrece i i ne moge se menjati
- Ukoliko zelimo da menjamo neki fajl koji se nalazi u nekom read-only sloju, taj fajl ce biti kopiran u read-write sloj, bice izmenjen i kao takav dalje koriscen
- Originalna verzija ce i dalje postojati (nepromenjena), ali nalazice se ,,skrivena" ispod nove verzije
- Na ovaj nacin se stedi jako puno prostora na disku, zato što već jednom skinuti layer-i mogu da se ponovo iskoriste

Docker hub

- Docker slike možemo da skinemo sa različitih mesta: (1) javni, (2) privatni
- Docker hub je privatni registar slika
- Docker hub liči malo na Git — ali samo liči
- Mi možemo sami da napravimo privatni registar za naše potrebe
- Postoje dva tipa slika: (1) oficijalne, (2) korisničke

Docker kontejner

- Kako slike predstavljaju build-time konstrukt, tako su kontejneri run-time konstrukt
- Gruba analogija odnosa izmedu slike i kontejnera se moze posmatrati kao klasa i instanca te klase
- Kontejneri predstavljaju lightweight execution environment koji omogucuju izolovanje aplikacije i njenih zavisnosti koristeci kernel *namespaces* i *cgroups* mehanizme
- namespaces nam rade izolaciju i svaki proces ima utisak kao da se on sam vrti na hardware-u
- cgroups nam omogućava da ograničimo koliko resursa svaki proces koristi
- Sa oba ova mehanizma dobili smo izolaciju i mogućnost da jedan proces ne *pojede* sve resurse sistema

Napomene

- Kontejneri su **read-only** strukture
- Sve izmene koje u njemu načinite **nestaju** onog momenta kada ugasite kontejner
- Ako treba da sačuvamo nekakve podatke ili resurse koje su nastale za vreme rada konrejnera, onda to moramo uraditi **van njega**
- To znači da moramo **mount-ovati** mesto na host OS-u za kontejner, i onda sve će biti **izbačeno** van kontejnera, a kontejner ima iluziju kao da su podaci kod njega
- Kontejneri su po svojoj prirodi **zatvoreni** za spoljnu komunikaciju
- Da bi nekakv saobraćaj ušao/izašao iz njega, moramo otvoriti port za komunikaciju
- Ista stvar kao i kod podataka, **povežaćemo** jedan port host OS-a i port kontejnera, i onda će sav saobraćaj koji stigne na taj port biti preusmeren na kontejner

Uvod

- Pre svega potrebno je da imate instaliran Docker na vašoj mašini
- Instalaijca je jednostavna, i trebate da instalirate *Docker community edition*
- Kompletno uputstvo možete naći u zvaničnoj dokumentaciji
- Nakon instalacije možete da proverite da li je sve prošlo kako treba koristeći komandu:

```
docker info
```

Komande

- Ukoliko zelimo da pokrenemo neki kontejner, kucamo komandu:

```
docker run naziv_slike
```

- Ako je slika prisutna biće kreiran kontejner od nje, ako slika nije prisutna ona će biti skinuta sa dockerhub-a
- Slika se sastoji od *n* layer-a i docker kreće da skida layer po layer, dok neki od layer-a mogu već biti prisutni na vašoj mašini i oni neće ponovo biti skunit
- Možemo zatrazili izlistavanje svih pokrenutih kontejnera sa komandom:

```
docker ps
```

- Ova komanda će prikazati samo kontejnere koji su trenutno aktivni, ako želimo da prikažemo sve kontejnere možmeo to uraditi dodavajući *flag* **-a** na kraj komande

- Možemo da zlistavamo informacije o svim preuzetim i kreiranim slikama sa komandom

```
docker images
```

- Mozemo izvrsiti i samo preuzimanje slike bez naknadnog pokretanja putem komande

```
docker pull naziv_slike:tag
```

- Možemo obrisati kontejner sa naše mašine komandom

```
docker rm naziv_kontejnera
```

- Možemo obrisati sliku sa naše mašine komandom

```
docker rmi naziv_slike
```

- Možemo zaustaviti kontejner komandom

```
docker stop naziv_kontejnera
```

- Možemo startovati ugašeni kontejner komandom

```
docker start naziv_kontejnera
```

Kreiranje naših slika

- Za potrebe kreiranja nase slike, neophodno je da kreiramo *Dockerfile* (sa tim nazivom), odnosno tekstualnu datoteku (najbolja praksa je da se ona nalazi u root direktorijumu projekta)
- Ova datoteka koristi bazicne instrukcije za kreiranje slika
- Kada kreiramo taj fajl, iz nje cemo kreirati nasu sliku izvrsavanjem instrukcija koje smo napisali komadom

```
docker image build
```

- Format je relativno jednostavan

```
# Comment
INSTRUCTION arguments
```

Podržane instrukcije

- FROM - Pomocu ove instrukcije definišemo koja je bazna slika za predstojece instrukcije koje ce biti izvrsene — svaki Dockerfile mora da ima FROM, i za osvnou najbolje birati neku od oficijalnih slika
- ADD - Ova instrukcija kopira fajlove sa zadate destinacije u fajlsistem slike na odredišnu destinaciju
- RUN - omogućava izvršavanje komande, pri cemu ce rezultat biti novi sloj (ayer) u samoj slici
- COPY - Slično kao i ADD instrukcija, s tim sto ADD omogućava da source bude i URL, dok COPY zahteva fizicku putanju na disku (bice dodat novi sloj u slici)
- WORKDIR - Postavlja putanju odakle će pojedine komande biti izvršene

- EXPOSE - Defini'v semo port kako bi mogli da odradimo mapiranje portova da bi kontejneri mogli da komuniciraju sa spoljašnjim svetom
- ENTRYPOINT - Postavljamo executable koji će biti pokrenut sa pokretanjem kontejnera
- ENV - Podešavanje environment varijabli.
- LABEL - Dodaje metapodakte slike, poput verzije, maintainer itd.
- Pored ovih postoji još komandi koje možete pročitati u dokuementaciji

Napomene

- Trebamo voditi računa da se prilikom izvršavanja pokrene onaj artefakt koji jezik koristi
- Za jezike koji su kompajlirani kao što je Go, ovde može doći do problema
- Nije zapravo problem, ali ne možemo prebacti binary u kontejner zato što možda koristimo drugi operativni sistem od onog koji je u kontejneru
- Ovo je ujedno i prednost nad drugim jezicima, zato što možemo da izbildamo binary u jednom kontejneru i da u novi kontejner prebacimo samo binary
- Ovo rezultuje **znatno** manjim slikama i kontejnerima, od nekih drugih jezika
- Ovo se radi jednom malo naprednijom tehnikom **multi stage build**

Multi stage build prvi deo

```
# Start from the latest golang base image
FROM golang:latest as builder

# Add Maintainer Info
LABEL maintainer="Milos Simic <milossimicsimo@gmail.com>"

# Set the Current Working Directory inside the container
WORKDIR /app

# Copy go mod and sum files
COPY go.mod go.sum ./


# Download all dependencies.
RUN go mod download

# Copy everything from the current directory to the container
COPY . .

# Build the Go app
RUN CGO_ENABLED=0 GOOS=linux go build -a -installsuffix cgo -o main .
```

Multi stage build drugi deo

```
######## Start a new stage from scratch #######
FROM alpine:latest

RUN apk --no-cache add ca-certificates

WORKDIR /root/

# Copy the Pre-built binary file from the previous stage
COPY --from=builder /app/main .

EXPOSE 8000

# Command to run the executable
CMD ["./main"] .
```

- Potrebno je da kreiramo našu sliku i opciono (ali preporučeno) da je tagujemo

```
docker image build -t first-app .
```

- Možemo da pokrenemo naš kontejner i da otvorim port ka spoljnem svetu (tag -p):

```
docker run -p 8000:8000 first-app
```

- Možemo pokrenuti kontjner *otkačen od terminala* tj u pozadini dodavajući tag -d

```
docker run -d -p 8000:8000 first-app
```

Dodatni materijali

- Graceful shutdown
- Graceful shutdown in go
- REST services Red Hat
- JSON
- Context package
- HTTP Request And Response

Kraj predavanja

Pitanja? :)