SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

SEMINAR

Programska i poslužiteljska infrastruktura skalabilnog sustava

Ivan Jeržabek
Voditelj: *Vedran Mornar*

Zagreb, svibanj 2023.

SADRŽAJ

1. Uvod

Rad opisuje nužne karakteristike poslužiteljske infrastrukture potrebne za pokretanje programske podrške skalabilnog sustava. Opisana je motivacija i postupak kontejnerizacije procesa te kako stvarati i pokretati Docker slike. Primjeri obuhvaćeni ovim radom dočarani su jednostavnim implementacijama poslužiteljske i klijentske aplikacije.

Softver kao servis (SaaS - Software as a Service) je popularan model razvoja programskih rješenja gdje korisnik pristupa usluzi putem klijentske aplikacije koja se spaja na servis u računalnom oblaku. Prednost za korisnika jest jednostavnost usluge, korisnik ne mora razmišljati o tehničkim izazovima koji se ukazuju pri održavanju programske podrške. Prednost za nuditelja usluge jest sveobuhvatna kontrola nad servisom, korisnik ne može nikako utjecati na samu uslugu. Nedostatak za korisnika jest naravno manjak kontrole nad samom uslugom, dok je nedostatak za nuditelja usluge obveza da se brine o dostupnosti samog servisa. Ako se nuditelj usluge odluči za ovaj model postoje određeni zahtjevi koje on mora ispuniti kako bi ponuđena usluga mogla raditi ispravno i kvalitetno. Ovaj rad obuhvaća neke od zahtjeva vezane za same poslužitelje te za programsku podršku potrebnu na tim poslužiteljima kako bi se pokrenuo nekakav konkretni servis.

2. Poslužitelj

Za izvedbu programskog rješenja potrebno je osigurati hardverske resurse na kojima će se softver izvršavati. Postoje razni načini izvedbe s različitim prednostima i manama ovisno o namjeni. Klasične opcije su kupovina vlastitog hardvera, unajmljivanje udaljenog hardvera te unajmljivanje virtualiziranih računalnih resursa. Glavna razlika između ovih opcija jest razina upravljanja hardverom te nužna količina održavanja hardvera. Kupovina vlastitog hardvera nudi najveću razinu fleksibilnosti i upravljanja jer vlasnik u potpunosti posjeduje čitavu infrastrukturu. Svaki aspekt sustava je moguće prilagoditi specifičnoj programskoj podršci koju vlasnik namjerava pokretati. Naravno ovakav pristup zahtjeva održavanje računalnog hardvera, što uključuje stvari poput:

- Osiguravanja prostora od neautoriziranih osoba
- Osiguravanje konstantnog dovoda električne energije i pristupa internetskoj mreži
- Redovito održavanje hardvera kako bi se osigurala kvalitetna razina rada
- Osiguravanje sigurnosnih kopija sustava u slučaju katastrofalnog ispada
- Zapošljavanje radnika koji će obavljati navedene dužnosti

Unajmljivanje udaljenog hardvera nudi izvrsnu ravnotežu pogodnosti i nedostataka navedenih pristupa. Korisnik može odabrati hardversku konfiguraciju prilikom kupovine te ju može naknadno promijeniti ovisno o zahtjevima sustava. Korisnik se ne mora brinuti o održavanju računalnog hardvera niti o prostoru u kojem se on nalazi. Korisnik ima potpunu kontrolu nad operacijskim sustavom koji mu je dostupan što još uvijek nudi više nego dovoljnu razinu fleksibilnosti. Nažalost ovo nije nužno savršeno rješenje - ovisno o nuditelju usluge cijena ovakvog pristupa može biti visoka u dugoročnom pogledu. Ovaj pristup također zahtjeva stručni kadar koji će znati upravljati sustavom. Postavljanje kvalitetnog web poslužitelja, vatrozida i slično nije nimalo jednostavan zadatak. Najmanju razinu brige ali i fleksibilnosti nose virtualizirani računalni resursi. Ovo nije jednoznačan pojam s obzirom na to da se odnosi primjerice na usluge gdje virtualna računala međusobno dijele hardverske resurse te

takozvane serverless usluge - platforme gdje korisnik učita posebno oblikovan kod koji prati određene standarde kako bi ga nuditelj usluge na vlastitu ruku pokrenuo u vlastitom okruženju. Korisnik se u drugom navedenom slučaju uopće ne mora brinuti o podršci potrebnoj za izvođenje servisa, brine se isključivo o vlastitom softveru. Ovaj pristup nudi vrlo fleksibilne modele plaćanja gdje korisnik plaća samo onoliko resursa koliko je zaista iskoristio. Ovo je vrlo isplativo dok sustav nema prevelike potrebe za resursima. Dodatne pogodnosti su brojne, od brzine podizanja sustava do globalne dostupnosti koju nuditelj sustava može osigurati.

2.1. Operacijski sustav

Najpopularniji izbor operacijskog sustava za poslužiteljsko okruženje jest Linux, specifično neke od njegovih distribucija poput Ubuntu. Podrška za distribucije operacijskih sustava linux je vrlo velika, dokaz tomu jest opširna dokumentacija te brojni online resursi za pomoć.

Većina distribucija se na poslužiteljima postavlja u načinu rada gdje se koristi isključivo ljuska, bez vizualnog korisničkog sučelja. Razlog tomu je što se svaka akcija na poslužitelju može izvršiti kroz kod u ljusci operacijskog sustava - to je idealno jer se mnoštvo akcija može automatizirati.

Napredno poznavanje operacijskog sustava omogućava zaista visoku razinu prilagodbe sustava te omogućava postavljanje različitih funkcionalnosti poput okruženja za razvoj, automatiziranog pokretanja testova, automatiziranu isporuku proizvoda i slično.

Alternativa linux operacijskom sustavu jesu naravno distribucije Windows operacijskih sustava za poslužitelje. Glavna razlika jest dostupnost grafičkog korisničkog sučelja što olakšava administraciju zbog manjeg potrebnog znanja.

2.2. Sigurnost

Pokretanju korisničkog softvera prethode naravno mjere sigurnosti kako bi infrastruktura bila otporna na napade i ranjivosti.

Pristup poslužitelju bi trebao pratiti načelo najmanjih privilegija. Svaki korisnik na sustavu ne smije imati pristup ničemu osim onoga što mu je eksplicitno dodijeljeno. Primjerice svaki razvojni inženjer u firmi bi trebao imati pristup nekom vlastitom direktoriju gdje može pokrenuti aplikacije u testnom okruženju, ali ne smije imati pristup

direktorijima drugih korisnika. Privilegije u ovom kontekstu se odnose na čitanje, pisanje te izvršavanje programa. Dodatno dobra praksa bi bila odvajanje produkcijskog okruženja od razvojnog okruženja tako da se one nalaze na potpuno odvojenim poslužiteljima.

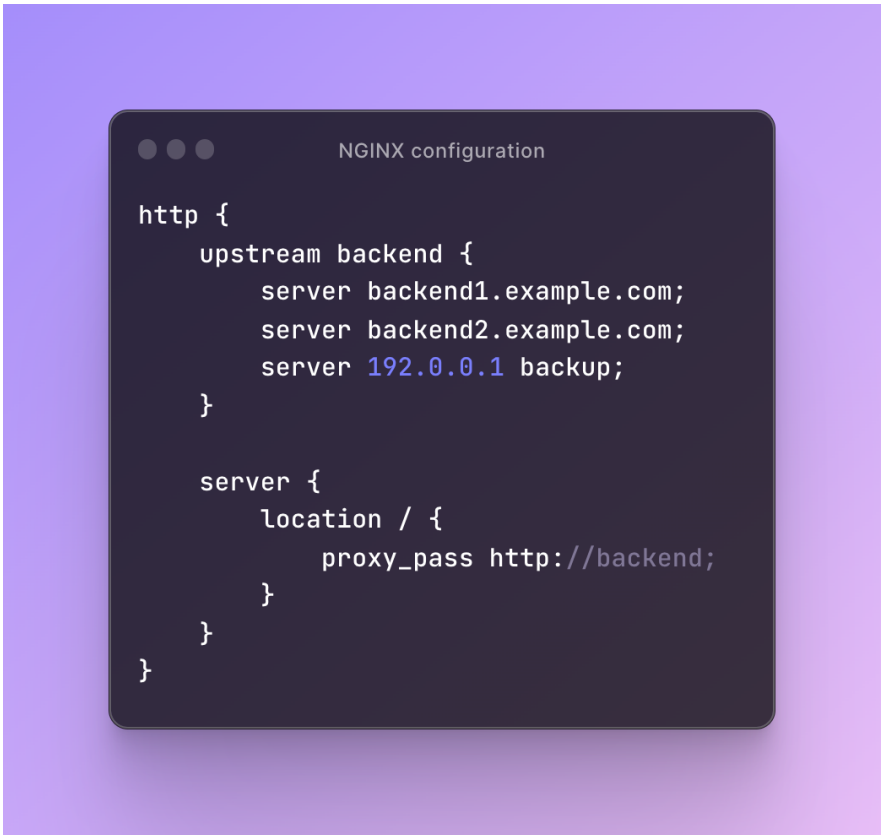
Prijava korisnika na poslužitelj bi morala biti vrlo sigurna. Popularan način sigurne autentifikacije na poslužitelje jest korištenje SSH protokola zaštićenog enkripcijskim ključevima. Na ovaj način korisničko ime i zaporka nisu dovoljni za prijavu u sustav jer je potreban i enkripcijski ključ koji se nalazi samo na računalu samog korisnika.

Vrlo bitna mjera zaštite jest vatrozid na poslužitelju. Bitno je blokirati mrežna vrata (port) koja koriste aplikacije na sustavu ali nisu namijenjene za pristup krajnjih korisnika. Jednostavan primjer ovoga jest blokiranje vrata 5432 na poslužitelju koji ima pokrenutu bazu podataka PostgreSQL. Bazi podataka trebao bi pristupati isključivo backend servis, u slučaju da vanjski aktor dobije pristup bazi podataka to može nositi ozbiljne pravne posljedice ako se razotkriju osobni podaci korisnika.

2.3. Web poslužitelj

Postoje mnogi protokoli komunikacije između servisa i poslužitelja. Java RMI sučelje omogućava nam primjerice da koristimo objekte u JVM memoriji koji se nalaze na drugom fizički udaljenom računalu povezanom putem mreže. Način povezivanja servisa dakle ovisi uvelike o arhitekturi aplikacije no u popularnoj mikroservisnoj arhitekturi najčešće susrećemo komunikaciju putem HTTP protokola. Takav način komunikacije zahtijeva otvorenu TCP konekciju na mrežnim vratima 80 ili 443 (ovisno o tome koristimo li sigurnu inačicu HTTP protokola). Osnovna HTTP komunikacija je trivijalna međutim osiguravanje stabilnog poslužitelja zahtjeva opsežno i kvalitetno implementiranu podršku HTTP protokola, ta zadaća leži na web poslužiteljima. Popularne opcije web poslužitelja su Apache te NGINX. Funkcionalnosti ovih web poslužitelja zapravo daleko nadilaze očekivanja jer sadržavaju usluge poput uravnotežavanja prometa (load balancing).

U primjeru vidimo kako se navode adrese instanci poslužitelja te kako se preusmjeruje promet na te instance. NGINX po zadanim postavkama koristi algoritam “round robin” gdje jednostavno ciklički prosljeđuje zahtjeve svakoj od instanci u grupi ravnomjerno.



Slika 2.1: NGINX Konfiguracija sa više instanci poslužiteljskog servisa

2.4. Optimizacije

Postoji mnogo načina optimiziranje softvera međutim to nije jedini način poboljšavanja performansi aplikacije. Smanjenje opterećenja moguće je postići smanjenjem poziva koji dođu do naših servisa ili smanjenjem zahtjeva za određenim resursima. Postoje resursi poput slika ili određenih dijelova web aplikacija koji se vrlo rijetko mijenjaju - takve resurse možemo privremeno pohraniti na drugim poslužiteljima kako bi naš glavni poslužitelj imao manje opterećenje. Opisana tehnologija naziva se cache - privremena pohrana. Općenito cache servisi rade tako da statičke resurse ili resurse koji se rijetko mijenjaju pohranjuju na čvorovima između krajnjeg korisnika i poslužitelja. Takvi čvorovi mogu biti replicirani ali i geografski raspoređeni diljem svijeta kako bi se dodatno smanjilo vrijeme odaziva.

Moguće je naravno implementirati vlastite cache servise no to zahtjeva opsežnu poslužiteljsku infrastrukturu. Popularna alternativa implementaciji vlastitog servisa jest korištenje komercijalnih rješenja poput Cloudflare. Pogodnosti korištenja takvih rješenja su brojne, bitno je napomenuti manjak troškova održavanja hardvera i softvera te brojne druge usluge koje Cloudflare nudi. Zaštita od DDoS napada je jedna od funkcionalnosti koje Cloudflare nudi i vrlo je pogodna, takvi napadi iskorištavaju ogromnu količinu poslužiteljskih resursa te ih je potrebno brzo i pravilno prepoznati i baratati njima.

3. Kontejnerizacija

Odvajanje ključnih segmenata u kodu i dobra organizacija su bitne značajke kvalitetnih programskih sustava, međutim to nisu jedini aspekti koda koje je bitno dobro organizirati. Izrada izvršivih datoteka te pokretanje istih su zadaci koje je bitno obavljati konzistentno i jednostavno, na sreću te procese je vrlo lako automatizirati uz pomoć kontejnerizacije.

Alati poput Docker ili Podman nam omogućuju definiranje, izgradnju i pokretanje kontejneriziranih aplikacija radi lakšeg i efikasnijeg upravljanja. Kontejnerizacija je jedna vrsta virtualizacije koja ima nekoliko prednosti nad uobičajenim virtualnim strojevima; manjak potrebnih resursa i brzina su glavne prednosti. Nadalje, velika prednost kontejnerizacije jest međusobna izolacija programa. Svaki program može individualno obaviti: instalaciju, izradu izvršivih datoteka, pokretanje. Zamjena jednog kontejnera ne utječe nikako na druge.

Konfiguracije kontejnera se nazivaju slike, njih je moguće dijeliti pomoću standardiziranih distributera (Dockerhub) ili vlastitih lokalnih distributera. Ovi koncepti omogućavaju vrlo jednostavan i efikasan način upravljanja isporučenom programskom podrškom. Slike se izrađuju tako da se spajaju slojevi gdje svaki sloj predstavlja modifikaciju datotečnog sustava. Prvi sloj može primjerice biti aplikacijsko okruženje node s programskim jezikom Javascript. Idući slojevi mogu imati razne svrhe, kopiranje programskog koda u kontejner, izgradnja izvršnih datoteka danog programskog koda te na posljetku to mogu biti naredbe koje će pokrenuti izvršne datoteke.

4. Primjer implementacije

U nastavku opisan je primjer kontejneriziranog sustava koji se sastoji od baze podataka, backend servisa te frontend aplikacije. Aplikacija je jednostavno sučelje za praćenje zadataka, uz mogućnost dodavanja novih i brisanja postojećih zadataka.

Implementacija [2] je bazirana na sljedećim tehnologijama

- PostgreSQL
- Java - Sprint Boot
- Typescript - Next.js

Backend servis ima dvije svrhe, on definira podatkovni model koji se pohranjuje u bazi podataka te nudi API sučelje putem kojeg je moguće pristupiti tim podacima. Frontend aplikacija pristupa API-u putem URI-a zapisanog u datoteci varijabli okruženja, zatim omogućava korisniku manipuliranje dohvaćenim podacima.

Oba servisa imaju definirane datoteke "Dockerfile" unutar kojih se nalazi konfiguracija slika kontejnera.

```
# First stage: Install dependencies
FROM node:16.18-alpine as deps

WORKDIR /app

COPY package.json yarn.lock ./
RUN yarn install

# Second stage: Build the application
FROM node:16.18-alpine as builder

WORKDIR /app

COPY . .
COPY --from=deps /app/node_modules ./node_modules

RUN yarn build && yarn install --production --ignore-scripts --prefer-offline

# Production image, copy all the files and run next
FROM node:16.18-alpine AS runner
WORKDIR /app

ENV NODE_ENV production

COPY --from=builder /app/next.config.js ./
COPY --from=builder /app/public ./public
COPY --from=builder /app/.next ./.next
COPY --from=builder /app/node_modules ./node_modules
COPY --from=builder /app/package.json ./package.json

CMD ["yarn", "start"]
```

Slika 4.1: Konfiguracija slike kontejnera frontend aplikacije

Slika prikazuje sadržaj Dockerfile datoteke frontend aplikacije gdje vidimo tri faze izrade slike. Prva faza jest instalacija paketa navedenih kao ovisnosti naše aplikacije. Nakon uspješne instalacije u novom okruženju se izgrađuju izvršne i minificirane datoteke pomoću kojih će se konačna aplikacija pokrenuti. Na posljetku u nanovo čistom okruženju donose se izgrađene izvršne datoteke te se navodi naredba pokretanja aplikacije.

Dobra je praksa pri izradi odvajati ove faze isporuke kako bi konačna slika kontejnera imala manju veličinu jer ne sadrži nepotrebne datoteke.

Kako bi se navedene slike mogle povezati i pokrenuti potrebno ih je povezati pomoću `docker-compose.yml` datoteke u slučaju Docker sustava. Unutar navedene datoteke definiraju se servisi koji se pokreću, na kojim slikama se temelje te dodatne opcije bitne prilikom pokretanja poput dodatnih varijabli okruženja ili mapiranje mrežnih vrata.

```yaml
version: "3.9"

services:

  sem2-db:
    image: postgres
    container_name: sem2-db
    restart: always
    environment:
      POSTGRES_DB: sem2
      POSTGRES_PASSWORD: 123
    ports:
      - "5550:5432"
    volumes:
      - sem2-db-data:/var/lib/postgresql/data

  sem2-backend:
    build:
      context: ./backend
      dockerfile: Dockerfile
    image: sem2-backend:1.0.0
    container_name: sem2-backend
    depends_on:
      - sem2-db
    ports:
      - "5555:8080"

  sem2-frontend:
    build:
      context: ./frontend
      dockerfile: Dockerfile
    image: sem2-frontend:1.0.0
    container_name: sem2-frontend
    depends_on:
      - sem2-backend
    ports:
      - "5560:3000"

volumes:
  sem2-db-data:
```

Slika 4.2: Kompozicija nekoliko kontejneriziranih servisa unutar docker-compose.yml datoteke

Pokretanje komponiranih slika je vrlo jednostavno s dvije naredbe koje se moraju izvesti. Prva je `docker compose build` kako bi se izgradile ili preuzele potrebne slike, zatim je potrebno izvesti naredbu `docker compose up` kako bi se pokrenuli kontejneri.

5. Zaključak

Ovaj rad proučava tehnike za postizanje skalabilnosti u programiranju i poslužiteljskim infrastrukturama, fokusirajući se na Docker, SaaS i sigurnost. Kontejnerizacija i Docker slike istaknute su kao ključni elementi u potpori skalabilnosti. Različiti pristupi u poslužiteljskim infrastrukturama, uključujući vlastite resurse, unajmljene udaljene hardvere i virtualizirane resurse, analiziraju se zajedno s njihovim prednostima i izazovima. U radu se dodatno istražuju sigurnosni aspekti, poput pristupa poslužitelju, autentifikacije, zaštite vatrozidom i optimizacije aplikacija.

6. Literatura

[1] Ian Gorton. *Foundations of Scalable Systems*. " O'Reilly Media, Inc.", 2022.

[2] Jeržabek Ivan. Repozitorij primjera kontejnerizirane aplikacije. `https://github.com/jerzabek/seminar-2`.

7. Sažetak

Rad opisuje nužne karakteristike poslužiteljske infrastrukture potrebne za pokretanje programske podrške skalabilnog sustava. Opisana je motivacija i postupak kontejnerizacije procesa te kako stvarati i pokretati Docker slike. Primjeri obuhvaćeni ovim radom dočarani su jednostavnim implementacijama poslužiteljske i klijentske aplikacije.