



Admin assignment

Lovro Rački

20. Listopada 2023.

Zadatak - web servis i docker	1
Increment service	2
Docker	3
Pokretanje	4
Zadatak - docker swarm	5
Osnovne postavke	5
Docker swarm	6
Pitanja i odgovori	6
Zadatak - benchmarks	8
TCPDump	9
Hipotetske arhitekturalne i konfiguracijske promjene za maksimizaciju performansi	9





Zadatak - web servis i docker

Increment service

Web servis za inkrementiranje ključa izveden je u *Python* programskom jeziku koristeći programsku knjižicu *Flask*.

Servis se sastoji od dvije rute:

1. "/" (root) - inkrementira brojač "visitCounter" za jedan pomoću metode *incr(key)*

```
# increment the counter by one
@app.route("/")
def increment():
    r.incr("visitCounter")
    print(r.get("visitCounter"))
    return "Incrementing counter"
```

2. "/reset" - postavlja brojač "visitCounter" na vrijednost 0 pomoću metode *set(key)*

```
# reset the counter to zero
@app.route("/reset")
def reset():
    r.set("visitCounter", 0)
    return "Resetting counter to 0"
```

Kod strukture servisa valjda napomenuti varijable *redis_host* i *redis_port* koje postavljaju potrebne vrijednosti radi komunikacije docker imagea servisa inkrementiranja i *redis* dockera.

```
app = Flask(__name__)
redis_host = "redis"
redis_port = int(os.environ.get("REDIS_PORT", 6379))
```

Servis se pokreće prilikom pokretanja *Python* skripte skretnicom `if __name__ == "__main__":`. Kod pokretanja servisa potrebno je obratiti pažnju na postavljanja hosta aplikacije na adresu 0.0.0.0, kako bi docker mogao pratiti potrebne zahteve za pristupom.

```
if __name__ == "__main__":
    r = redis.Redis(host="redis", port=redis_port, decode_responses=True)
    app.run(host="0.0.0.0")
```



Docker

Spajanje servisa za inkrementiranja s redis dockerom izvedeno je pomoću docker-compose.yml. Ovisnosti servisa za inkrementiranje napisane su u requirements.txt. Iako servis koristi samo *redis* i *flask*, u tekstuanoj datoteci ovisnosti izričito mora biti navedena i verzija *Werkzeuga* zbog osjetljivosti verzioniranja komponenti flaska.

```
redis==5.0.1
...
flask==2.0.2
...
Werkzeug==2.2.2
```

Docker-compose.yml prilično je jednostavan no treba izdvojiti nekoliko zanimljivosti. Iako to za prvi zadatak nije potrebno u increment_service dijelu datoteke prvo je navedena oznaka image: lovro-docker, jer kasnije docker swarm ne podržava build iz docker datoteke nego mora izričito koristiti image. Bilo je i potrebno preusmjeriti port 5000 na 5001 kako se docker ne bi zabunio.

Nakon pogleda na dio yml datoteke koji definira variable okoline, u oči upadaju REDIS_HOST i REDIS_PORT variable. Iako su ovdje vrijednosti već definirane, docker jednostavno nije htio čitati tu varijablu prilikom pokretanja servisa inkrementiranja pa ju je bilo potrebno manualno zapisati u varijablu servisa. Iako se zapisivanje varijabli okoline u programski kod smatra izuzetno lošom praksom i sigurnosnim manjkom, nije mi se više dalo rezati s tim pa sam ostavio u kodu.

```
services:
  increment_service:
    image: lovro-docker
    build:
      context: .
      dockerfile: lovro-docker
    ports:
      - "5001:5000"
    depends_on:
      - redis
    environment:
      REDIS_HOST: redis # Use the service name as the hostname
      REDIS_PORT: 6379

  redis:
    image: redis:latest
    ports:
      - "6379:6379"
```



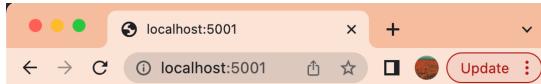
Pokretanje

Docker se pokreće jednostavno sa docker-compose up(pod prepostavkom da je docker upaljen), ali za svaki slučaj preporučam prvo pokrenuti naredbe za brisanje svih imagea i ponovno kreiranje jer nekad zbog nekog razloga uzima stari image umjesto da napravi novi.

```
~$ bazuso @ Jure-Matesina in ~/R/UniqCast
[< (master)* ->] docker-compose down --rmi all
[+] Running 5/5
✓ Container unicast-increment_service-1 Removed
✓ Container unicast-redis-1 Removed
✓ Image redis:latest Removed
✓ Image lovro-docker:latest Removed
✓ Network unicast_default Removed
[< bazuso @ Jure-Matesina in ~/R/UniqCast>
[< (master)* ->] docker-compose up -d --force-recreate --renew-anon-volumes
[+] Running 9/9
! increment_service Warning
  redis 7 layers [::] 0B/0B Pulled
✓ 1bc163a14ea6 Pull complete
✓ dc70419cf2a3 Pull complete
✓ 7288668abaf4 Pull complete
✓ 85c8727158e7 Pull complete
✓ 6b62d1554b5c Pull complete
✓ 4f4fb700ef54 Pull complete
✓ c8079d502bc1 Pull complete
[+] Building 1.7s (10/10) FINISHED
--> [increment_service internal] load .dockerrcignore
--> transferring context: 2B
--> [increment_service internal] load build definition from Lovro-docker
--> transferring dockerfile: 315B
--> [increment_service internal] load metadata for docker.io/library/python:3.8-slim-buster
--> [increment_service 1/5] FROM docker.io/library/python:3.8-slim-buster@sha256:8799b0564103a9f36
--> [increment_service internal] load build context
--> transferring context: 8.25MB
--> CACHED [increment_service 2/5] WORKDIR /src
--> CACHED [increment_service 3/5] COPY src/requirements.txt requirements.txt
--> CACHED [increment_service 4/5] RUN pip install --no-cache-dir -r requirements.txt
--> [increment_service 5/5] COPY .
--> [increment_service] exporting to image
--> exporting layers
--> writing image sha256:b13e952f292113502b19061533ed4988c4989c78da5ad5d8b5c2505ae8f5a1d3
--> naming to docker.io/library/lovro-docker
[+] Running 3/3
✓ Network unicast_default Created
✓ Container unicast-redis-1 Started
✓ Container unicast-increment_service-1 Started
[< bazuso @ Jure-Matesina in ~/R/UniqCast>
[< (master)* ->]
```

Nakon što ste pokrenuli docker otvorite browser i posjetite localhost:5001 za inkrementiranje ključa i localhost:5001/reset za inkrementiranje ključa.

Za provjeru vrijednosti brojača upaliti redis-cli i napišite get("visitCounter") 😊.



Incrementing counter



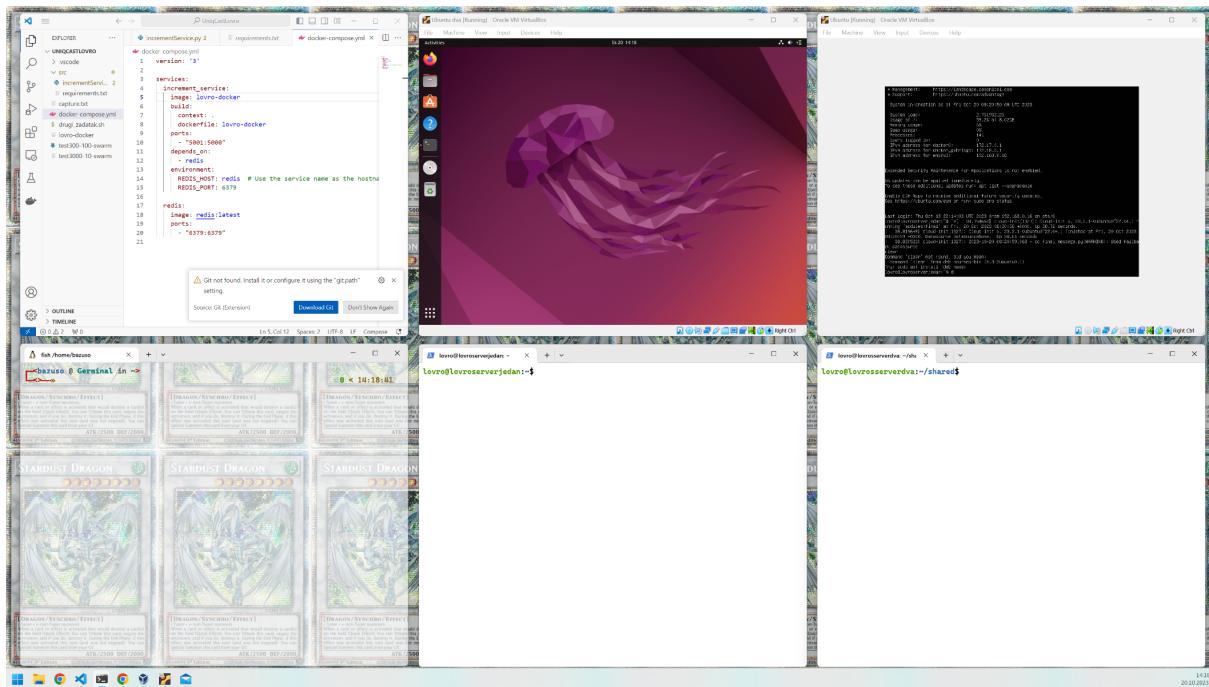
Zadatak - docker swarm

Osnovne postavke

Prvi zadatak u potpunosti je napravljen na *macbooku* na *appleovom* procesoru. Kada sam započinjao s rješavanjem zadatka u potpunosti sam zaboravio da je nakon razlaza od intelovih procesora na macu sve bolje osim virtualizacije koja je za moje skromne sposobnosti gotovo nemoguća. Zbog toga sam se prebacio na svoje *Windows* računalo što se pokazalo kao najgora moguća odluka u mom životu. Naime, čvor *docker swarm* jednostavno se ne želi priključiti ostalim čvorovima ako se jedan od tih čvorova nalazi u *Windowsu*. Istražujući po internetu naišao sam na urbane mitove o manager čvoru unutar *Windowsa*, no ne vjerujem dok ne vidim uživo.

Zbog nastalih problema ovaj zadatak napravljen je pomoću dva virtualna računala na kojima se vrti *ubuntu*. Jedan desktop i drugi server. Jedno virutalno računalo ima 4 dodijeljene jezgre, a drugo 2, oba imaju 4 GiB *RAMa*.

Nije mi se dalo boriti sa shared *clipboardom VirtualBoxa* pa sam se na kraju spojio sa *SSH*om na oba računala, a razmjena datoteka vršila se preko dijeljenog direktorija.





I tried to create a similar Swarm with a Windows manager node but never really got it to work. You can initialize a single-node Swarm from Windows with `docker swarm init`. However adding multiple worker nodes does not appear to be supported at the moment:
<https://docs.docker.com/engine/swarm/swarm-tutorial/>.

"Currently, you cannot use Docker Desktop for Mac or Docker Desktop for Windows alone to test a multi-node swarm".

The following options are possible:

- Pure Linux swarm (Linux manager + Linux workers) which runs only Linux containers
- Hybrid Swarm (Linux manager + Windows workers + Linux workers) which runs Windows and Linux containers
- (Sometimes) Pure Windows Swarm using Win Server 2019 as the manager. The regular Windows updates have been known to break various features of Swarm. For example,
<https://github.com/moby/moby/issues/40998>

Then everyone either tries workarounds or waits for the next Windows update to fix the problem.

Personally I've had good luck with hybrid Swarm. It works fine with simple Ubuntu manager + standard Windows 10 workers. No need for Win Server.

Neki čovjek na *stack overflowu* koji ima isti problem kao i ja.

Docker swarm

Budući da je prilikom rješavanja prvog zadatka obraćena pažnja na osjetljivosti *docker swarm* samo postavljanje je bilo prilično jednostavno. Potrebne naredbe sažeo sam u jednu *shell* skriptu, isključivo u svrhe prezentacije (ne pokretati).

```
$ drugi_zadatak.sh
1  #!/bin/bash
2
3  docker swarm init
4
5  docker stack deploy -c docker-compose.yml lovro
6
7  docker service scale lovro_increment_service=10
```

Konačni docker servis se sastoji od jednog redis servisa i 10 instanci servisa inkrementiranja.

Pitanja i odgovori

Jesu li postignute bolje performanse od rješenja iz prvog zadatka?

U slučaju dugog, velikog tereta niske konkurentnosti docker swarm postiže bolje performanse od rješenja iz prvog zadatka, no u slučaju kratkog tereta visoke konkurentnosti swarm bilježi lošije performanse.



Znači li deseterostruko više instanci servisa deseterostruko bolje performanse?

Deseterostruko više instanci servisa nikako ne znači deseterostruko bolje performanse. U određenim slučajevima dolazi do osjetnog poboljšanja, no uz loše upravljanje swarmom lako je doći i do gorih rezultata zbog raznih problema, kao što je “coordination overhead”, odnosno resursi potrebni za koordinaciju svih instanci servisa.



Zadatak - benchmarks

Usporedni benchmark servisa napravljen je pomoću apache *benchmark-a*, *tcp_dump-a* i *docker stats*.

	Latency (ms)	50% (ms)	Longest request (ms)	Total time (sec)	Throughput (#/sec)
3000-100 single	0.839	83	108	2.517	1192.10
30000-10 single	1.536	8	19519	46.073	651.14
3000-100 swarm	1.543	107	1135	4.630	647.89
30000-10 swarm	1.282	6	219	38.466	779.91

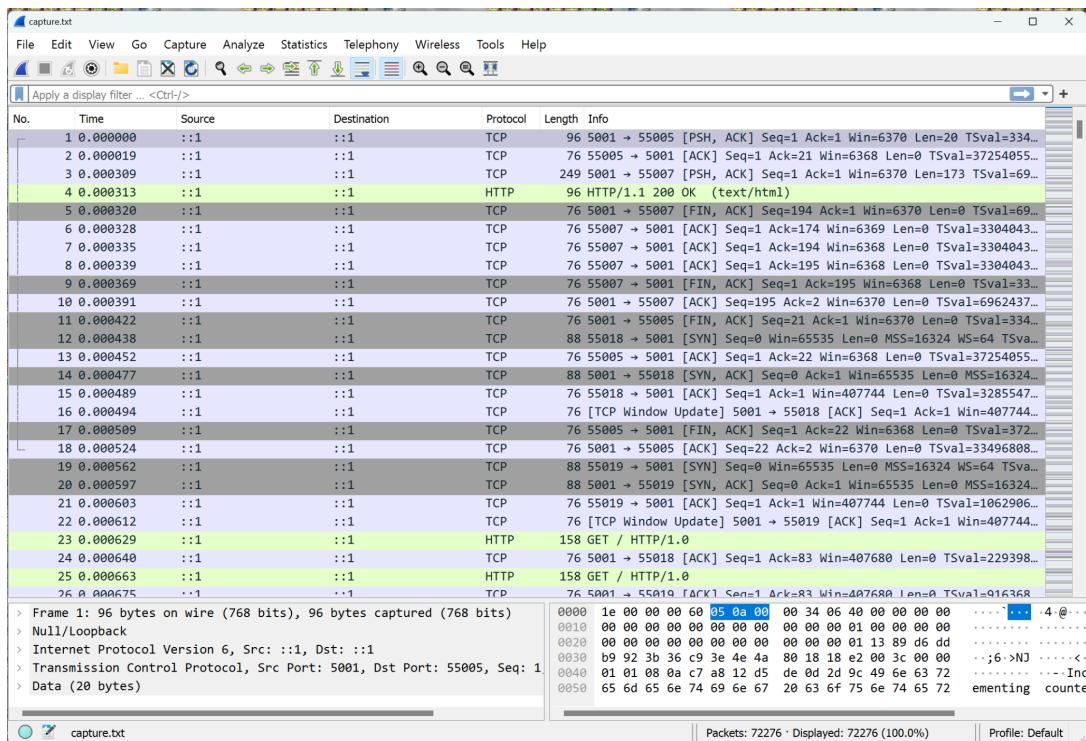
Tablica je organizirana u stupce i retke. U retcima se nalaze redom performanse za:

1. Običan *docker image*, test od tri tisuća zahtjeva sa razinom konkurentnosti od 100, odnosno sevisu će biti poslano ukupno tri tisuće zahtjeva po 100 paralelnih zahtjeva
2. Običan *docker image*, test od trideset tisuća zahtjeva sa razinom konkurentnosti od 10, odnosno sevisu će biti poslano ukupno trideset tisuća zahtjeva po 10 paralelnih zahtjeva
3. *Docker swarm* sa 10 instanci, test od tri tisuća zahtjeva sa razinom konkurentnosti od 100, odnosno sevisu će biti poslano ukupno tri tisuće zahtjeva po 100 paralelnih zahtjeva
4. *Docker swarm* sa 10 instanci, test od trideset tisuća zahtjeva sa razinom konkurentnosti od 10, odnosno sevisu će biti poslano ukupno trideset tisuća zahtjeva po 10 paralelnih zahtjeva

Iz testa je jasno vidljivo da docker swarm ima bolje performanse kada je potrebno poslužiti puno zahtjeva niske konkurentnosti, a običan se docker bolje snalazi u okruženjima više konkurentnosti, odnosno većeg broja paralelnih zahtjeva, zbog “*Coordination overheada*” i ostalih otežavajućih okolnosti paralelizacije preko mreže.



TCPDump



Hipotetske arhitekturalne i konfiguracijske promjene za maksimizaciju performansi

Koje biste arhitekturalne ili konfiguracijske promjene predložili za maksimizaciju performansi na jednom serveru?

Optimizacija performansi na jednom serveru prilično je jednostavna. Nabaviti najveće raspoložive količine *RAMa*, najbrži mogući višejezgredi procesor, *SSDove* itd. Uz to potrebno je sam redis držati na istom serveru kako bi se maksimalno smanjili gubitci prilikom prijenosa podataka putem mreže.

Koje biste arhitekturalne ili konfiguracijske promjene predložili za maksimizaciju performansi na više servera?

Za arhitekturu sa više servera potrebno je iskoristiti sve blagodati koje redis pruža, kao što je *Redis Cluster*, *Redis Cache* i njihovu potporu za *load balancing*. Osim toga potrebno je i poznavati sustav koji projektiramo, ovisno o broju operacija čitanja i pisanja, možda bi bilo korisno odvojiti jedne od drugih te smanjiti database isolation level. Ako naš sustav uglavnom obavlja operacije čitanja *Read uncommitted* bi ubrzao sustav.



Kako biste orkestrirali redis instance naspram servisnih instanci?

Nije mi baš jasno ovo pitanje pa se ispričavam ako sam fulao sve.

U svakom slučaju bi imao manje instanci redisa nego servisa jer su write operacije ionako najveće usko grlo sustava. Sustav je onoliko brz koliko njegovo najveće usko grlo. Ovisno o strukturi aplikacije koju je potrebno održavati, možda bi uveo poneku redis instancu za svako udaljenije geografsko područje. Npr. ako moja aplikacija ima dvije velike skupine korisnika, jednu u Americi, jedu u Europi, uveo bi po još redis instanci u Americi.

Ako moj sustav ima više operacija čitanja, broj instanci servisa bio bi daleko veći od broja instanci *redisa*, ako ima više operacija pisanja omjer instanci servisa i instanci *redisa* bio bi suprotan.