



Materijali za 1. laboratorijsku vježbu

Sadržaj

Git i GitHub	3
Što je Git i zašto ga koristiti?	3
Što je GitHub i zašto ga koristiti?	4
Osnovni koncepti	4
Korištenje Gita i GitHuba	5
Instalacija Gita, postavljanje GitHub računa, kreiranje i povezivanje s repozitorijem	5
Povezivanje lokalnih promjena s GitHub repozitorijem	6
Dohvaćanje najnovijih promjena s GitHub repozitorija lokalno.....	8
Git branching	8
Razrješavanje konflikta i pull request.....	10
Pravila zaštite grane (branch protection rule)	13
Završne pripreme udaljenog repozitorija.....	13
Platforma Docker	14
Uvod	14
1-Instalacija Dockera	15
2-Pokretanje "Hello world" kontejnera	15
3-Pisanje jednostavne Node.js aplikacije.....	16
4-Pakiranje Node.js aplikacije	17
5-Pokretanje.....	19
Apache Kafka	21
Uvod	21
Kafka <i>on-prem</i>	22
Pokretanje.....	23
Slanje poruka	25
Korisničko sučelje	26
Zadatci.....	27

Git i GitHub

Što je Git i zašto ga koristiti?

Git je sustav za upravljanje verzijama. Sustav za upravljanje verzijama je softver dizajniran za praćenje promjena napravljenih na datotekama tijekom vremena. Konkretnije, Git je distribuirani sustav za upravljanje verzijama, što znači da svi koji rade na projektu u Gitu imaju cijelu kopiju povijesti projekta, a ne samo uvid u trenutno stanje datoteka.

Čak i ako radite sami, ili pak uređujete tekstualne datoteke, prednosti korištenja Gita su brojne, kao što su:

- Mogućnost poništavanja promjena - ako napravite grešku, možete se vratiti na prethodnu točku u vremenu kako biste povratili raniju verziju vašeg rada.
- Potpuna povijest svih promjena - ako ikada poželite vidjeti kako je vaš projekt izgledao prije dan, tjedan, mjesec ili godinu dana, možete pregledati prethodnu verziju projekta kako biste vidjeli kako su datoteke izgledale tada.
- Dokumentacija o tome zašto su promjene napravljene - često je teško zapamtiti zašto je promjena napravljena. S porukama o commitu u Gitu lako je dokumentirati zašto radite promjenu za buduću referencu.
- Sigurnost u izmjene - budući da je lako povratiti prethodnu verziju vašeg projekta, možete imati sigurnost da možete napraviti promjene koje želite. Ako zamisli ne uspiju, uvijek se možete vratiti na raniju verziju svog rada.
- Višestruki tokovi - možete stvoriti različite grane za eksperimentiranje s različitim promjenama vašeg sadržaja ili za neovisnu izgradnju različitih značajki. Zatim ih možete spojiti natrag u glavnu povijest projekta (main branch) kada su završene ili ih izbrisati ako ste odustali ili pak niste zadovoljni.

Rad na timu omogućuje još širi raspon pogodnosti korištenja Git-a za praćenje vaših promjena. Neke od ključnih prednosti Git-a u radu u timu su:

- Mogućnost rješavanja sukoba - s Gitom više ljudi može raditi na istoj datoteci istovremeno. Obično će Git moći automatski spojiti promjene; ako ne može, pokazat će vam o kojim je sukobima riječ i time vam olakšati njihovo razrješavanje.

- Neovisni tokovi povijesti - različiti ljudi na projektu mogu raditi na različitim granama omogućavajući vam da neovisno radite na različitim značkama, a zatim ih spojite kada su gotove.

Što je GitHub i zašto ga koristiti?

GitHub je web stranica na kojoj možete uploadati kopije svog Git repozitorija koja vam omogućuje jednostavnije surađivanje s ljudima na projektu, odnosno Git platforma na kojoj se nalazi vaš udaljeni (remote) repozitorij. Git olakšava kolaboraciju na projektu pružanjem centraliziranog mjesta za dijeljenje repozitorija i web sučelja te raznim značkama koje vam omogućuju da učinkovite specificirate, raspravite i pregledate promjene sa svojim timom.

GitHub je puno više od mjesta za spremanje Git repozitorija - pruža brojne druge benefite kao što su:

- Kolaboracija na različitim tokovima - korištenjem grana i zahtjeva za povlačenje možete surađivati na različitim granama i značkama.
- Pregled rada u tijeku - možete vidjeti razne značajke na kojima se trenutno radi gledajući listu pull requestova te klikom na neki možete vidjeti zadnje promjene kao i sve rasprave o promjenama.
- Povijest napretka tima - pregledavanje povijesti commitova omogućuje vam da vidite na čemu je tim dosad radio.

Osnovni koncepti

Neki od osnovnih koncepata koje trebate razumjeti da biste mogli učinkovito raditi s Gitom i GitHubom uključuju:

- **Repository** (repozitorij) - repozitorij je "skladište" u kojem se čuva cijeli vaš projekt i sve promjene koje ste na njemu napravili; odnosno, baza podataka koja sadrži sve informacije potrebne za upravljanje revizijama i povijesti projekta. U Gitu repozitorij zadržava kompletnu kopiju cijelog projekta tijekom životnog ciklusa.

- **Commit** - commit je "snimka" cijelog projekta u određenom trenutku. Git ne bilježi pojedinačne promjene u datotekama; snima sliku cijelog projekta. Uz snimku, commit također sadrži informacije o autoru sadržaja i "committeru" koji je promjene objavio u repozitorij.
- **Commit message** (poruka o commitu) - svaki put kada napravite commit trebate priložiti poruku koja opisuje napravljene promjene - takva poruka je neprocjenjiva kada kasnije pokušavate razumjeti zašto je određena promjena implementirana.
- **Branch** (grana) - nezavisna serija commitova koju stvarate pri izradi novih značajki i eksperimenata.
- **Main branch** (glavna grana) - kada kreirate novi projekt postoji zadana grana koja je kreirana - grana main. To je grana na kojoj bi trebao završiti sav vaš rad jednom kada je spreman za produkciju. Negdje ćete naići i na naziv Master branch.
- **Merge** (spajanje) - način da se dovršeni rad iz jedne grane komponira u drugu granu - najčešće ćete spajati značajke s drugih grana u main granu.
- **Pull request** (zahtjev za povlačenje) - pull request se koristi kao zahtjev za tuđim pregledom vašeg rada kako bi se značajka sa druge grane mogla spojiti na glavnu.
- **Clone** (kloniranje) - Često ćete htjeti preuzeti kopiju projekta s GitHuba kako biste na njemu radili lokalno. Postupak kopiranja repozitorija na vaše računalo naziva se kloniranje.

Korištenje Gita i GitHuba

Kako biste vidjeli Git u akciji provest ćemo vas kroz osnovne korake baratanja ovim alatima. Kroz jednostavne primjere ćete naposljetku stvoriti repozitorij koji ćete koristiti kao prostor za predaju rješenja nadolazećih laboratorijskih vježbi.

Instalacija Gita, postavljanje GitHub računa, kreiranje i povezivanje s repozitorijem

Krenimo ispočetka - u ovom odjeljku ćete instalirati Git, kreirati svoj račun te se povezati lokalno s vašim GitHub repozitorijem:

1. Preuzmite Git sa sljedeće stranice <https://git-scm.com/downloads>.
2. Ukoliko nemate GitHub račun stvorite novi račun.

3. Prijavite se te kreirajte novi privatni repozitorij naziva tpiuo - označite Add a README file s kvačicom.
4. Uredite README.md datoteku tako da sadržava vaše osobne podatke - ime, prezime i FER email, npr:

```
- **Ime:** [Upišite vaše ime]
- **Prezime:** [Upišite vaše prezime]
- **Email:** [Upišite vaš FER email]
```

5. Pritisnite Code gumb kako biste dobili HTTPS URL repozitorija (<https://github.com/{naziv-računa}/tpiuo.git>)
6. Pokrenite naredbeni redak iz proizvoljne mape te pokrenite naredbu git clone [URL iz prethodnog koraka]
7. Pozicionirajte se u tpiuo mapu te postavite vaše ime i email adresu s git config naredbom:

```
a.git config --global user.email "vašemail@primjer.com"
```

```
b.git config --global user.name "vaše ime"
```

Povezivanje lokalnih promjena s GitHub repozitorijem

Nakon što ste uspješno uspostavili konekciju s GitHub repozitorijem, izvest ćete lokalne promjene koje ćete učitati natrag na GitHub repozitorij - kreirat ćete novu datoteku koju ćete objaviti na vaš udaljeni repozitorij. Krenimo redom:

1. Stvorite HTML dokument proizvoljnog sadržaja, npr:

```
<!DOCTYPE html>
<html>
<body>

Prva laboratorijska vježba (TPIUO)

</body>
</html>
```

2. Pokrenite naredbu

```
git add [imedokumenta]
```

kako biste premjestili datoteku iz radnog prostora (workspace) u prostor pripreme (staging area).

3. Pokrenite naredbu

```
git commit -m "commit message"
```

kako biste spremili promjene iz prostora pripreme u lokalni repozitorij, gdje "commit message" predstavlja kratku i jasnu poruku čiji sadržaj ukazuje kakve su promjene napravljene u dokumentu.

4. Pokrenite naredbu

```
git push
```

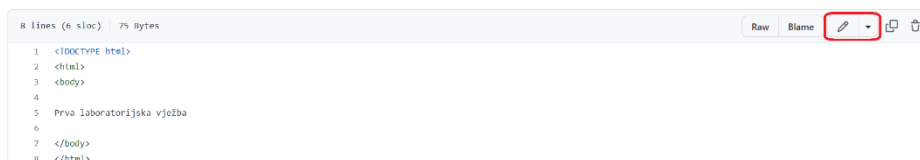
kako bi promjene iz lokalnog repozitorija postale vidljive na udaljenom GitHub repozitoriju.

5. Osvježite GitHub stranicu repozitorija i provjerite prisutnost vaše HTML datoteke.

Dohvaćanje najnovijih promjena s GitHub repozitorija lokalno

Pri zajedničkom radu vaše datoteke neće uvijek biti u najnovijoj verziji - vaši suradnici će objaviti nove verzije, a kako biste naučili kako dohvatiti te promjene lokalno rekreirat ćemo jednu takvu situaciju stvaranjem promjena direktno na GitHub stranici:

1. Otvorite svoju HTML datoteku na GitHub stranici vašeg repozitorija te naćinite proizvoljne izmjene.



2. U odjeljku "Commit changes" zapišite svoju poruku o commitu te potvrdite izmjene.
3. Pokrenite naredbu

```
git pull
```

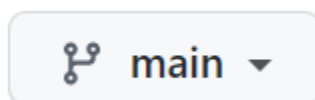
kako biste dohvatili najnovije promjene.

4. Otvorite svoju HTML datoteku te provjerite jesu li se uspješno izvršile izmjene.

Git branching

Git branching vam omogućuje rad na različitim verzijama iste datoteke bez ometanja glavnog koda projekta. Primjerice, kada želite raditi na novoj znaćajki bez ometanja originalne funkcionalnosti glavnog koda projekta ideja je kreirati novu granu koju ćete jednom kada znaćajka bude odobrena spojiti s glavnom granom.

Svoje grane možete pregledati na udaljenom repozitoriju klikom na sljedećići gumb:



ili lokalno pokretanjem

```
git branch
```

naredbe. Za bolje razumijevanje ovog koncepta stvorite novu granu te napravite promjene koje ćete usporediti s mainom:

1. Kreirajte novu granu naredbom

```
git checkout -b [ime-grane]
```

gdje ime-grane za uvodni primjer ima vrijednost lab1.

2. Upišite opet naredbu `git branch` te primijetite kako osim što se prikazala nova grana da se i zvjezdica (*) premjestila kraj grane lab1 - oznakom * se prikazuje trenutna grana.
3. Kreirajte proizvoljne promjene na HTML datoteci te pokrenite

```
git commit -am "commit message"
```

na lab1 grani - kraći oblik `git add` i `commit` naredbi u jednoj. Primjer promjena:

```
< DOCTYPE html>
<html>
<body>

Prva laboratorijska vježba (TPIUO) - izmjena na 5. liniji

</body>
</html>
```

4. Pokrenite naredbu

```
git push --set-upstream origin [ime-grane]
```

kako bi se vaša lokalna grana povezala i postala vidljiva na udaljenom repozitoriju.

5. Korištenjem naredbe

```
git checkout [ime-grane]
```

se vratite na main branch.

6. Osvježite svoju datoteku te primijetite kako promjene s lab1 grane nisu dohvaćene na main grani.

Razrješavanje konflikta i pull request

Merge conflict (konflikt spajanja) nastaje kada dva programera (ili vi sami) rade na istoj datoteci i istim linijama koda. U takvom scenariju Git ne zna kako popraviti problem i prebacuje odgovornost na vas da razriješite tu situaciju. U prethodnom poglavlju ste napravili izmjene na 5. liniji koda u HTML datoteci u svom lokalnom repozitoriju na grani lab1 - zamislite da je vaš suradnik napravio promjene na toj istoj liniji na grani main te objavio promjene na udaljenom repozitoriju. Vaš sljedeći korak je spajanje vaše grane s mainom koristeći zahtjev za povlačenje - pull requestovi su zamišljeni da započnu razgovor o predloženoj promjeni - obično se radi o novoj značajki ili ispravci pogreške. Rekreirajmo takav događaj - nakon što ste završili s prethodnim poglavljem vaš zadatak je spojiti novu značajku s main granom.

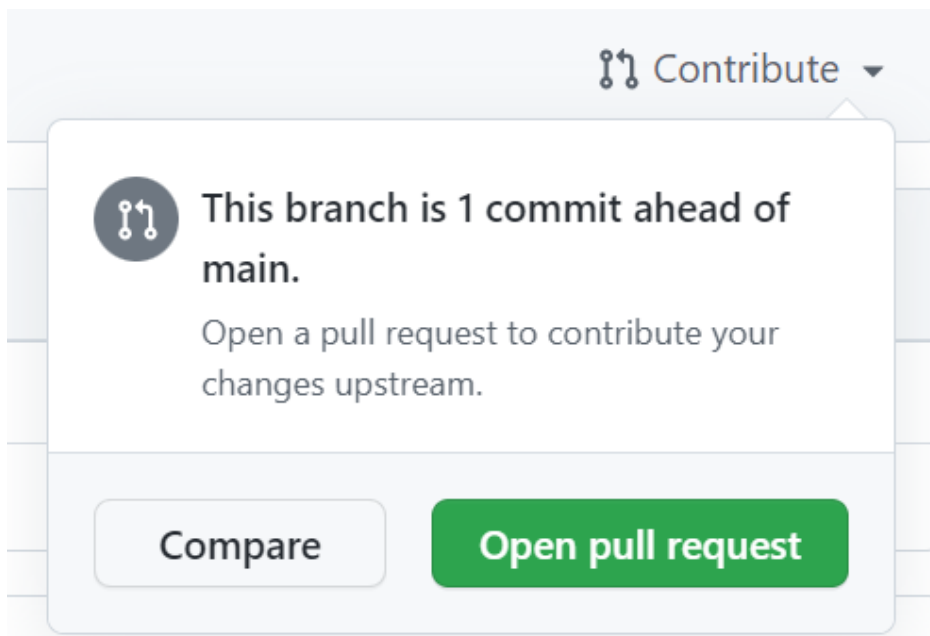
1. Otvorite svoju datoteku u udaljenom GitHub repozitoriju na grani main i učinite proizvoljne promjene na istoj liniji kao u prethodnom koraku te ih objavite, npr:

```
<!DOCTYPE html>
<html>
<body>

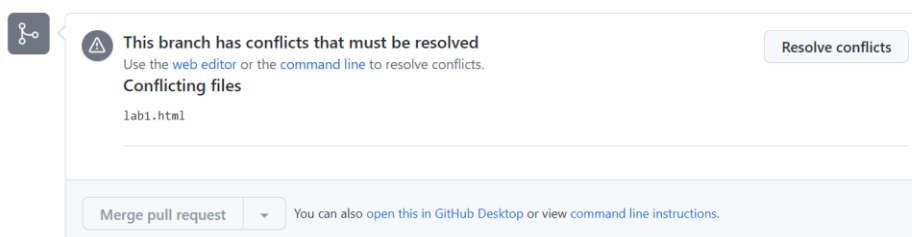
Prva laboratorijska vježba (TPIUO) - nova izmjena na 5. liniji

</body>
</html>
```

2. Sada se pozicionirajte na lab1 granu na GitHubu te kliknite gumb Contribute i odaberite opciju "Open pull request".



3. Pred vama će se otvoriti nova stranica - imat ćete uvid u promjene koje ste obavili, imati opciju imenovanja pull requesta te opisa što ste u njemu napravili. Obično ćete u odjeljak Reviewer dodati željenog recenzenta vašeg koda koji će vam odobriti zahtjev, ali kako sami rekreirate ovu situaciju ovo polje ćete ostaviti prazno. Kliknite na "Create pull request" gdje ćete zaprimiti sljedeću obavijest:



Dogodio se konflikt koji morate razriješiti da biste spojili promjene na novoj grani natrag na main.

4. Vratite se u naredbeni redak te se pozicionirajte na granu lab1 i pokrenite naredbu

```
git pull origin main
```

kojom ćete dohvatiti promjene s glavne grane na udaljenom repozitoriju. Nakon pozivanja naredbe zaprimit ćete sljedeću poruku:

```
CONFLICT (content): Merge conflict in lab1.html
Automatic merge failed; fix conflicts and then commit the result.
```

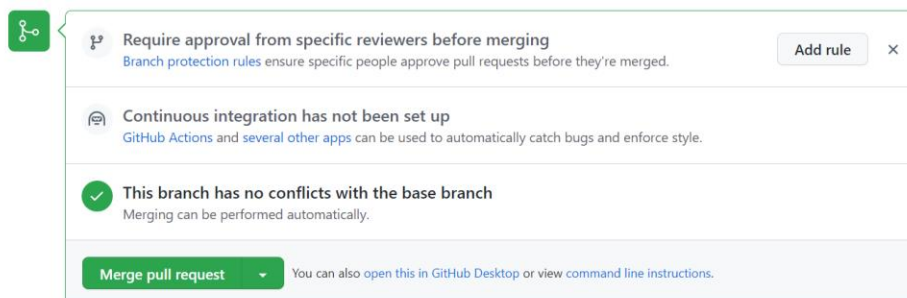
5. Sljedeći korak je razrješavanje konflikta - otvorite svoj dokument u uređivaču po izboru i odaberite liniju s vašeg lokalnog repozitorija te takav dokument objavite na udaljeni repozitorij. Takvu promjenu možete izvesti klikom na "Accept Current Change" ili brisanjem svih novih linija.

```
Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes
<<<<<< HEAD (Current Change)
Prva laboratorijska vježba (TPIUO) - izmjena na 5. liniji
=====
Prva laboratorijska vježba (TPIUO) - nova izmjena na 5. liniji
>>>>>> a29ce5e7a0071b3c658cc2cab181e22142f28177 (Incoming Change)
```

6. Commitajte promjene te potom pokrenite naredbu

```
git push -u origin lab1
```

7. Osvježite stranicu svog pull requesta gdje ćete biti obaviješteni kako konflikti ne postoje i da možete spojiti svoje promjene s lab1 grane na main granu. Pritisnite "Merge pull request" te potom "Confirm merge" te će se vaše promjene pojaviti na glavnoj grani.



Pravila zaštite grane (branch protection rule)

Na GitHubu, zaštićene grane (protected branch) su grane koje imaju dodatne restrikcije pristupa i spajanja kako bi se odražavala kontrola kvalitete i stabilnosti koda. Zaštićene grane se obično koriste kako bi se osiguralo da se kod prije spajanja na glavnu granu pregleda i testira kako bi se spriječilo prelijevanje bugova i drugih problema s odabrane na glavnu granu. Neka od čestih pravila zaštite uključuju:

- Zahtijevanje pull requesta prije spajanja s glavnom granom
- Ograničavanje tko može objaviti promjene na grani
- Određene vrste statusnih promjena (npr. automatizirani testovi) koje moraju proći da bi se mogle spojiti promjene

Kako biste pristupili tim ograničenjima pritisnite gumb "Settings" te potom u lijevom odjeljku gumb "Branches". Pod "Branch name pattern" ćete upisati naziv grane te označiti opcije koje želite omogućiti i time ih zaštititi od neželjenih promjena.

Završne pripreme udaljenog repozitorija

Ostalo je još nekoliko koraka do završetka vježbe i završnog osposobljavanja vašeg repozitorija za laboratorijske vježbe. Prvo, izbrišite svoju HTML datoteku s main grane. Kada uspješno objavite te promjene na udaljeni repozitorij istražite stranicu svog GitHub repozitorija - pronađite gdje se nalaze svi commitovi, kako vidjeti repozitorij u trenutku određenog commita te primijetite kako GitHub prikazuje promijenjene datoteke i linije koda u commitovima. Ako ste uspješno obavili sve do ovog retka - obavili ste prvi dio laboratorijske vježbe. Za potrebu svake sljedeće kreirat ćete novu granu naziva lab[n], gdje n predstavlja redni broj vježbe, te na nju objavljujati svoja rješenja laboratorijskih vježbi. Poželjno je da sve promjene pratite na svom GitHub repozitoriju - redovno objavite nove promjene na vašim datotekama bez obzira na tip datoteke.

Platforma Docker

Uvod

Tehnologije za [kontejnerizaciju](#) aplikacija postoje već duže vrijeme, ali su često bile komplicirane za korištenje. [Docker](#) je prva takva tehnologija koja je omogućila jednostavnost: pakiranje aplikacija u lagane i prenosive kontejnere s čitavim operacijskim sustavom i bibliotekama nikad nije bilo lakše. Kada se pokrene aplikacija zapakirana pomoću Dockera, ona uvijek vidi upravo one datoteke s kojima je zapakirana. Potpuno je nebitno izvodi li se ona na osobnom računalu ili na ogromnom poslužitelju koji se nalazi na fakultetu, koji vrlo vjerojatno ima potpuno drukčiji operacijski sustav i biblioteke.

Zamislite ovaj scenarij: preuzmete VirtualBox, pokrenete neku instancu Linux operacijskog sustava i na njemu razvijete aplikaciju sa svim potrebnim bibliotekama da sve radi. Nakon toga spremite svoj virtualni stroj na disk i pošaljete ga prijatelju da ga pokrene u svom VirtualBox-u. Upravo se to radi Dockerom, samo što on ne koristi ogromne virtualne strojeve, nego lagane kontejnere kojima se postiže približno jednaka razina izolacije aplikacije.

Tri glavna koncepta Dockera:

1. **Slika** - ono u što se pakira aplikacija i njeno okruženje (datoteke, biblioteke, metapodatci)
2. **Registar** - repozitorij (javni ili privatni) na koji se spremaju Docker slike za jednostavno dijeljenje, najpoznatiji je [Docker Hub](#)
3. **Kontejner** - obični Linuxov kontejner stvoren pomoću Docker slike, to je zapravo jedan proces, potpuno izoliran od operacijskog sustava domaćina i svih ostalih procesa koji se izvode na njemu

U sljedećih nekoliko poglavlja naučit ćete kako:

1. Instalirati Docker
2. Pokrenuti prvi "Hello world" kontejner
3. Napraviti jednostavnu Node.js aplikaciju
4. Zapakirati aplikaciju kao sliku
5. Pokrenuti aplikaciju kao zaseban kontejner

1-Instalacija Dockera

Naravno, da biste radili s Dockerom, prvo ga trebate instalirati. Da vam ne bude dosadno, u ovom koraku ćemo vas prepustiti na milost i nemilost Dockerovih uputa:

1. [Upute za Linux](#)
2. [Upute za Windows](#)



Ove upute su zapravo za Docker Desktop, što uključuje sučelje za komandnu liniju i grafičko sučelje. Na ovim vježbama koristit ćete komandnu liniju jer ćete tako bolje naučiti komande, grafičko sučelje možete tu i tamo pogledati da malo promijenite perspektivu.

2-Pokretanje "Hello world" kontejnera

Za ovaj zadatak koristit ćete [busybox](#). Nećete morati ništa preuzimati niti instalirati, sve će to Docker riješiti. Vaš jedini zadatak je napisati točnu naredbu kojom se specificira slika koja se želi preuzeti i eventualno koje naredbe treba izvršiti:

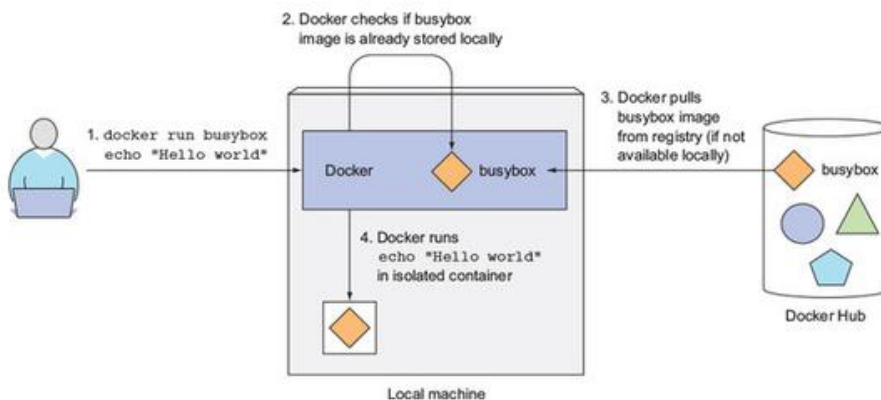
```
docker run busybox echo "Hello world"
```

Rezultat bi trebao izgledati ovako:

```
PS C:\Windows\system32> docker run busybox echo "Hello world"
Unable to find image 'busybox:latest' locally
latest: Pulling from library/busybox
205dae5015e7: Pull complete
Digest: sha256:
7b3ccabffc97de872a30dfd234fd972a66d247c8cfc69b0550f276481852627c
Status: Downloaded newer image for busybox:latest
Hello world
```

Ne izgleda impresivno, ali uzmite u obzir da ste cijelu aplikaciju preuzeli i pokrenuli sa samo jednom kratkom komandom. U ovom slučaju, aplikacija je samo jedna izvršna datoteka, ali mogla je biti iznimno kompleksna, s mnoštvom biblioteka o kojima ovisi, a

cijeli proces pokretanja bio bi identičan ovome. Dapače, najčešće je i jednostavniji od ovoga jer nije potrebno, kao u ovom primjeru, eksplicitno pisati naredbu koju želite izvršiti (echo "Hello world").



1 Pokretanje Docker kontejnera

3-Pisanje jednostavne Node.js aplikacije

Sad kada ste vidjeli kako Docker radi, vrijeme je za pisanje aplikacije. Otvorite [Visual Studio Code](#) ili bilo koji drugi editor, kreirajte datoteku app.js i zalijepite sljedeći kod:

```
const http = require('http');
const os = require('os');

console.log("Kubia server starting...");

let handler = function (request, response) {
  console.log("Received request from " + request.connection.remoteAddress);
  response.writeHead(200);
  response.end("You've hit " + os.hostname() + "\n");
};

let www = http.createServer(handler);
www.listen(8080);
```


Ne brinite za detalje, ono što morate znati je da ova aplikacija pokreće HTTP poslužitelj na portu 8080. Poslužitelj na svaki zahtjev odgovara statusom 200 OK i tekstom "You've hit <hostname>". Osim toga, poslužitelj još ispiše klijentovu IP adresu na standardni izlaz.

4-Pakiranje Node.js aplikacije

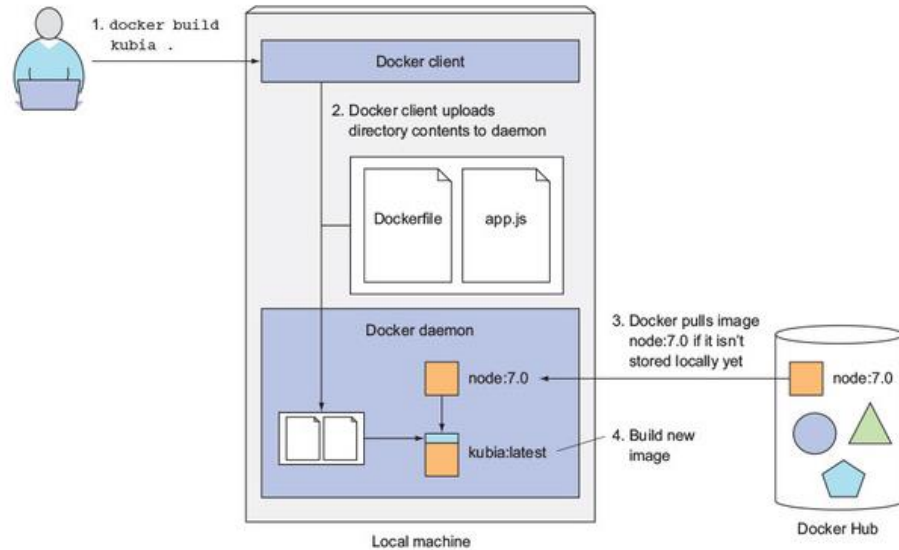
Kako biste zapakirali aplikaciju, trebate kreirati datoteku koja se zove Dockerfile i koja sadržava listu uputa koje će Docker koristiti da bi stvorio sliku. Dockerfile mora biti u istom direktoriju kao i app.js:

```
FROM node:7
ADD app.js /app.js
ENTRYPOINT ["node", "app.js"]
```

Ključna riječ FROM pokazuje koja će se slika koristiti kao polazište za vašu sliku. U ovom slučaju to je node s oznakom 7 (oznaka je tu jer može postojati više različitih verzija iste slike). U drugoj liniji kopira se app.js datoteka u korijenski direktorij slike. Treća linija definira koja komanda će se izvršiti kada netko pokrene sliku.

U ovom trenutku imate sve potrebno za izgradnju vašu slike. Da biste to napravili, izvršite sljedeću naredbu:

```
docker build -t kubia .
```



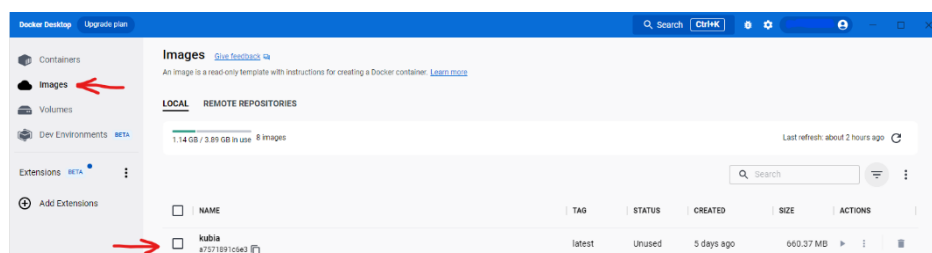
2 Dockerov posao

Rekli ste Dockeru da izgradi sliku koja se zove kubia, na temelju datoteka koje se nalaze u trenutnom direktoriju (zato je ona točkica na kraju naredbe). Kad se proces izgradnje završi, sliku možete vidjeti na svom računalu pomoću naredbe:

```
docker images
```

```
PS C:\Windows\system32> docker images
REPOSITORY          TAG             IMAGE ID        CREATED         SIZE
kubia                latest         a7571891c6e3   4 days ago     660MB
```

Također, svoju sliku sada možete vidjeti i na Docker Desktopu:



5-Pokretanje

Sada možete pokrenuti aplikaciju:

```
docker run --name kuba-container -p 8080:8080 -d kuba
```

Naredba kaže Dockeru da napravi novi kontejner imena kuba-container od slike kuba. Kontejner neće zauzeti terminal, nego će biti pokrenut u pozadini (zastavica -d). Port 8080 vašeg računala mapirat će se na port 8080 u kontejneru (-p 8080:8080) da biste mogli pristupiti aplikaciji na adresi <http://localhost:8080>.

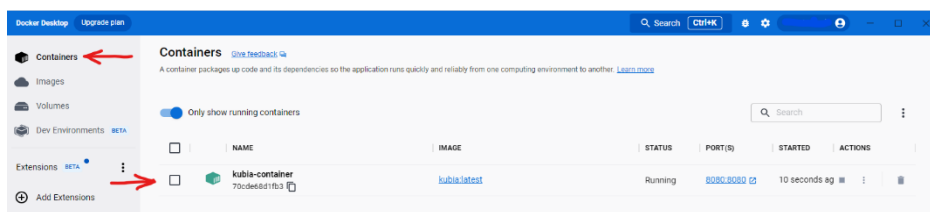
Ispis svih kontejnera koji se izvršavaju može se postići naredbom:

```
docker ps
```

```
PS C:\Windows\system32> docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
PORTS		NAMES		
70cde68d1fb3	kuba	"node app.js"	3 minutes ago	Up 2 minutes
0.0.0.0:8080->8080/tcp		kuba-container		

Isto kao i sliku, kontejner možete vidjeti u Docker Desktopu:



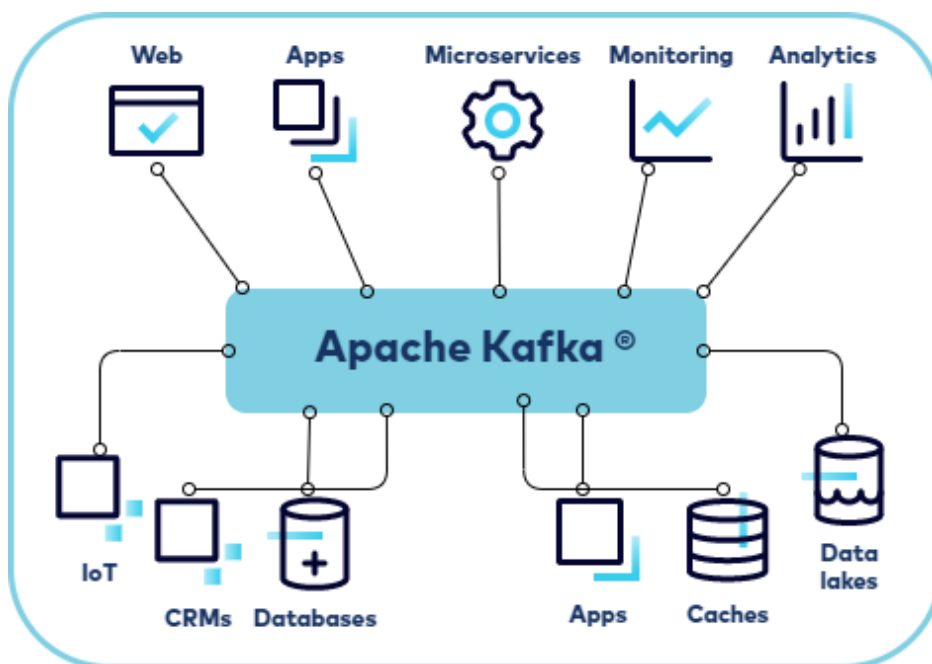
I na kraju, ako u svoj preglednik upišete adresu <http://localhost:8080>, trebali biste vidjeti ispis poslužitelja:



Apache Kafka

Uvod

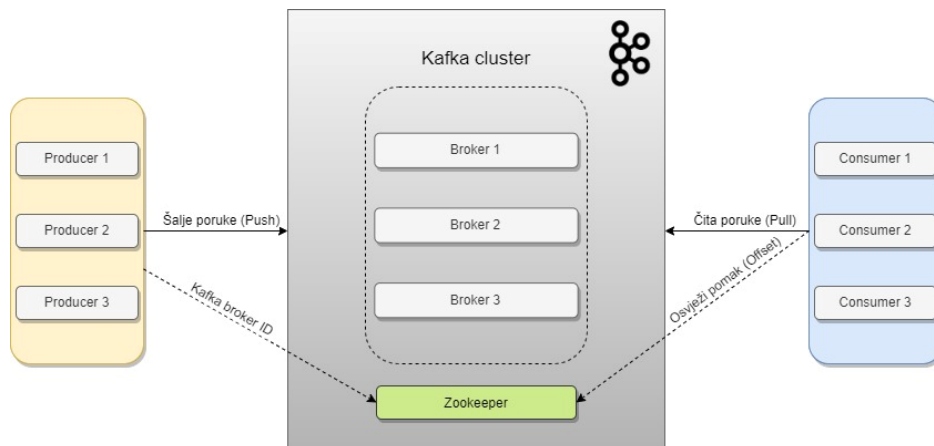
Kafka je raspodijeljena platforma za strujanje događaja/podataka (engl. *distributed event streaming platform*) preko koje raspodijeljeni sustavi mogu pisati i čitati poruke, spremati ih proizvoljno dugo vremena i obrađivati ih po njihovom dolasku u stvarnom vremenu. Da bi to bilo moguće, Kafka mora biti pouzdana i raditi bez ikakvih problema, a to se postiže podizanjem Kafke u skupinama (engl. *cluster*). Pojedine instance Kafke u skupini mogu raditi na različitim računalima (pritom računalo može biti fizičko, virtualno, kontejner ili oblak) na bilo kojim lokacijama u svijetu. Ako se dogodi kvar na jednom računalu na kojem radi Kafka, druge instance Kafke pokreću novu instancu Kafke na drugom računalu i sustav nastavlja s radom.



3 Raspodijeljeni sustav

Na slici 3, mnoge komponente moraju međusobno izmjenjivati podatke te kada ne bi bilo Kafke, mreža bi bila kompleksna. U toj situaciji, dodavanjem novih komponenata u sustav mora se paziti da se sve veze točno implementiraju. Uz to, postoji mogućnost da veza dvije komponente s vremenom pukne te sustav više ne radi kako je zamišljeno. Kafka pojednostavljuje sustav tako što smanjuje broj veza između komponenata sustava

i time smanjuje rizik od kvara. Komponente koje šalju poruke šalju ih na pojedini topic ili particiju unutar topica, a komponente koje čitaju se pretplaćuju (engl. *subscribe*) za pojedine poruke i preuzimaju ih po njihovom dolasku.



4 Arhitektura Kafke

Kafka se sastoji od dvije velike komponente, Zookeeper i Server(na slici Broker). Zookeeper je odgovoran za generalno održavanje Kafka clustera, a brokeri su serveri koji pohranjuju poruke iz jednog ili više izvora. Producers šalju poruke brokerima, a consumeri ih čitaju.



Za detalje o Kafki, preporučuje se sljedeća poveznica

<https://docs.confluent.io/kafka/introduction.html#introduction-to-ak>.

Kafka *on-prem*

Pokretanje Kafke *on-prem*, odnosno lokalno, može se ostvariti na nekoliko načina, a u ovoj vježbi pokazat ćemo kako se Kafka pokreće kroz Docker, (osim kroz Docker, Kafku je moguće pokrenuti ručno, gdje se Kafkin zookeeper i server pokreću odvojeno).

Pokretanje

Za pokretanje Kafke, preuzmite `docker-compose.yaml` i `config.yaml` datoteke i stavite ih u isti direktorij. Otvorite terminal u tom direktoriju i pokrenite naredbu:

```
docker-compose up -d
```

Ovom naredbom pokrenut će se zookeeper i broker.

```
version: '2.1'

services:
  zoo1:
    image: zookeeper:3.4.9
    hostname: zoo1
    ports:
      - "2181:2181"
    environment:
      ZOO_MY_ID: 1
      ZOO_PORT: 2181
      ZOO_SERVERS: server.1=zoo1:2888:3888
    volumes:
      - ./zk-single-kafka-single/zoo1/data:/data
      - ./zk-single-kafka-single/zoo1/datalog:/datalog

  kafka1:
    image: confluentinc/cp-kafka:5.5.0
    hostname: kafka1
    ports:
      - "9092:9092"
    environment:
      KAFKA_ADVERTISED_LISTENERS:
        LISTENER_DOCKER_INTERNAL://kafka1:19092,LISTENER_DOCKER_EXTERNAL://{DOCKE
        R_HOST_IP:-127.0.0.1}:9092
      KAFKA_LISTENER_SECURITY_PROTOCOL_MAP:
        LISTENER_DOCKER_INTERNAL:PLAINTEXT,LISTENER_DOCKER_EXTERNAL:PLAINTEXT
      KAFKA_INTER_BROKER_LISTENER_NAME: LISTENER_DOCKER_INTERNAL
      KAFKA_ZOOKEEPER_CONNECT: "zoo1:2181"
      KAFKA_BROKER_ID: 1
```

```

    KAFKA_LOG4J_LOGGERS:
    "kafka.controller=INFO,kafka.producer.async.DefaultEventHandler=INFO,state
    .change.logger=INFO"
    KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
    volumes:
      - ./zk-single-kafka-single/kafka1/data:/var/lib/kafka/data
    depends_on:
      - zool

```

```

kowl:
  image: quay.io/cloudhut/kowl:v1.2.1
  restart: on-failure
  hostname: kowl
  volumes:
    - ./config.yaml:/etc/kowl/config.yaml
  ports:
    - "8080:8080"
  entrypoint: ./kowl --config.filepath=/etc/kowl/config.yaml
  depends_on:
    - kafka1

```

```

kafka:
  brokers:
    - kafka1:19092

```

Da se uvjerite da Kafka uspješno radi, pokrenite naredbu:

```

docker exec <container_name> kafka-topics --bootstrap-server
<container_name>:9092 --list

```

Ovom naredbom će se ispisati svi postojeći topici.

Da kreirate novi topic, pozovite naredbu:

```

docker exec <container_name> kafka-topics --bootstrap-server
<container_name>:9092 --create --topic <topic_name>

```


Za brisanje postojećeg topica, koristi se naredba:

```
docker exec <container_name> kafka-topics --bootstrap-server  
<container_name>:9092 --delete --topic <topic_name>
```

Slanje poruka

Kada je kreiran topic, možete poslati neke poruke na njega tako da pokrenete naredbu:

```
docker exec --interactive --tty <container_name> kafka-console-producer  
--bootstrap-server <container_name>:9092 --topic <topic_name>
```

Iza znaka ">" napišite poruku koju želite poslati, a pošaljete je pritiskom tipke enter.

Konačno, poslane poruke se čitaju pokretanjem naredbe:

```
docker exec --interactive --tty <container_name> kafka-consoleconsumer  
--bootstrap-server <container_name>:9092 --topic <topic_name> --from-  
beginning
```

Zastavica `--from-beginning` omogućava čitanje svih poruka od zadnje pročitane, što znači da ako se na topicu nalaze poruke koje su stigle prije pokretanja consumera, on će ih i dalje pročitati. Pokrenite producera i consumera u odvojenim terminalima i pošaljite nekoliko poruka da se uvjerite da kafka radi.

Unutar topica poruke se mogu podijeliti na particije. Način na koji se to napravi je da se prema ključu poruke generira hash i po dobivenom hashu se određuje particija poruke. Prije slanja poruka, mora se kreirati topic s više particija (zadana vrijednost je 1). To ćete napraviti dodavanjem zastavice `--partitions 3` u funkciju koja kreira topic, gdje je 3 broj particija koje će se nalaziti unutar topica. Tada će funkcija izgledati ovako:

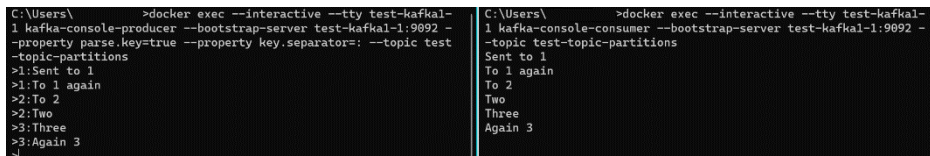
```
docker exec <container_name> kafka-topics --bootstrap-server  
<container_name>:9092 --create --partitions 3 --topic <topic_name>
```

Sada kada budete slali poruke, nadodat ćete i ključ na početak poruke koji će određivati particiju na koju ćete poslati poruku. Pokretanje producera također ćete malo modificirati tako što ćete postaviti način rada s ključevima i odrediti separator ključa i

poruke:

```
docker exec --interactive --tty <container_name> kafka-consoleproducer
--bootstrap-server <container_name>:9092 --property parse.key=true --
property key.separator=: --topic <topic_name>
```

Poruke koje šaljete sadržavat će ključ koji će od ostatka poruke biti odvojen znakom ":". Consumera pokrećete kao i zadnji put.



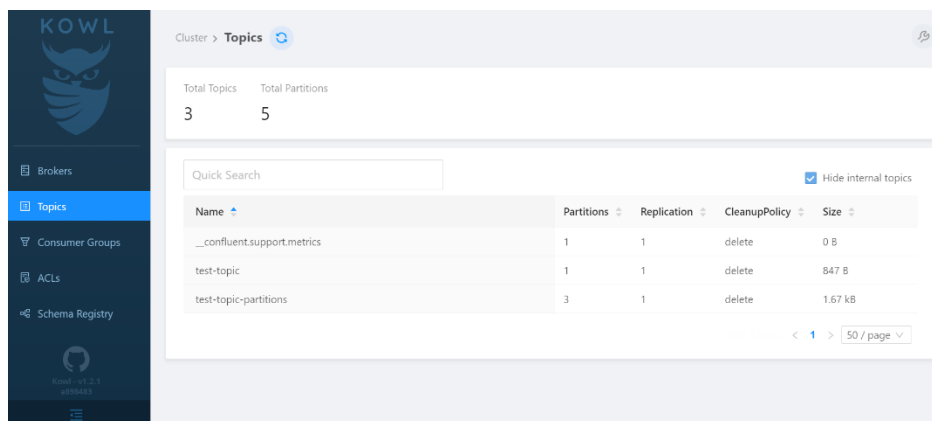
```
C:\Users\>docker exec --interactive --tty test-kafkal-
1 kafka-console-producer --bootstrap-server test-kafkal-1:9092 -
-property parse.key=true --property key.separator=: --topic test
-topic-partitions
>1:Sent to 1
>1:To 1 again
>2:To 2
>2:Two
>3:Three
>3:Again 3
>
```

```
C:\Users\>docker exec --interactive --tty test-kafkal-
1 kafka-console-consumer --bootstrap-server test-kafkal-1:9092 -
-topic test-topic-partitions
Sent to 1
To 1 again
To 2
Two
Three
Again 3
```

5 Producer i consumer poruka s particijama

Korisničko sučelje

Na slici 5 se može vidjeti da consumer čita poruke, ali ne ispisuje ključeve poruka. Kako biste vidjeli ključ i particiju kojoj pojedine poruke pripadaju, pokretanjem docker-composea isto tako pokrenut je i Kowl, program koji omogućava korisničko sučelje za Kafku. Da biste otvorili Kowl, otvorite bilo koji preglednik i na localhostu, na portu 8080 možete vidjeti sučelje.



6 Kowl

Uz Kowl možete vidjeti topice koje ste kreirali, možete vidjeti postavke pojedinih topica i

poruke koje se trenutno nalaze na njima.

Messages						
Consumers						
Partitions						
Configuration						
Documentation						
PARTITION	START OFFSET	MAX RESULTS	FILTER		Quick Search	
All	Newest-50	50				
Offset	Partition	Timestamp	Key	Value	Preview	Size
8	2	15/02/2023, 16:11:48	3	+ Again 3		7 B
7	2	15/02/2023, 16:11:46	3	+ Three		5 B
6	2	15/02/2023, 16:11:43	2	+ Two		3 B
5	2	15/02/2023, 16:11:40	2	+ To 2		4 B
8	0	15/02/2023, 16:11:36	1	+ To 1 again		10 B
7	0	15/02/2023, 16:11:31	1	+ Sent to 1		9 B

7 topic test-topic-partitions

Ovdje se vidi kako poruke imaju različite ključeve, ali i particije tih ključeva.

Zadatci

1. Kreirajte topic "input-topic" s 3 particije.
2. Ispišite sve topice i provjerite je li input-topic kreiran.
3. Pošaljite poruke tako da specificirate ključeve; pošaljite barem 6 poruka s 3 različita ključa (parovi poruka s istim ključem)
4. Pročitajte poslano poruke
5. Pogledajte stanje topica na Kowl UI-u (obratite pozornost na ključeve i kako oni utječu na particije)