Servisno orijentisane arhitekture

Predavanje 1: REST Servisi, JSON, Docker, Docker compose

Univerzitet u Novom Sadu
Fakultet Tehničkih Nauka

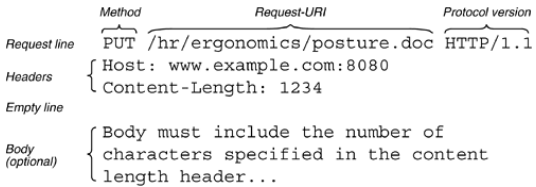
Uvod

- Fokus REST web service-a je na resursima, i kako omogućiti pristup tim resursima preko interneta
- Resurs moze biti predstavljen kao objekat u memoriji, file na disku, podaci iz aplikacije, baze podataka itd.
- Prilikom dizajniranja sistema prvo je potrebno da identifikujemo resurse, a zatim ustanoviti kako su oni povezani
- Ovaj postupaka je sličan modelovanju baze podataka

- Kada smo identifikovali resurse, sledeći korak je da uspostavimo način kako da te resurse reprezentujemo u našem sistemu
- Za te potrebe mozemo koristiti bilo koji format za reprezentaciju resursa (JSON, XML npr.).
- Da bi dobili sadržaj sa udaljene lokacije (od nekog server-a), client mora napraviti HTTP zahteh, i poslati ga web service-u
- Nakon svakog HTTP zahteva, sledi i HTTP odgovor od server-a ka client-u tj. onome ko je zahtev poslao
- Client može biti korisnik, ili može biti neka druga aplikacija (npr. drugi web service, mobilna aplikacija, ...)

HTTP zahtev

- Svaki HTTP zahtev se sastoji od nekoliko elemenata:
  - **[METOD]** GET, PUT, POST itd. odnosno koju operaciju želmo da uradimo nad podacima
  - **[URL/HOSTNAME]** Putanja do resursa nad kojim će operacija biti izvršena
  - **[HEADERS]** Dodatni podaci koji se šalju serveru
  - **[BODY]** Dodatni podaci koji treba da se pošalju serveru da bi opreracija (npr POST, PUT) bila uspešno izvršena



(HTTP Request)

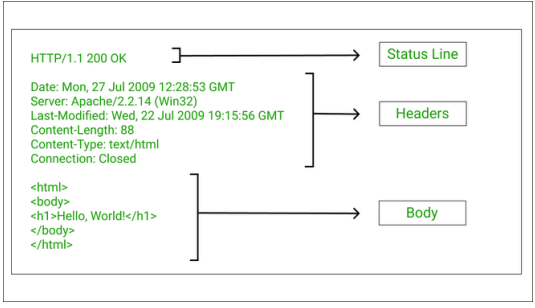
Slanje zahteva

- Svaki servis mora imati jedinstvenu adresu (URL) na koju šaljemo HTTP zahtev na primer:
  - http://MyService/Persons/1
- Ako zelimo da izvedemo nekakav upit nad web service-om, to možemo da uradimo koristeći HTTP METOD i parametrizaciom putanje web service-a
- Česta opcija je upotreba *?* simbol na kraj putanje (ova opcija više simbolizuje RPC nego REST zahtev, ali se može koristiti)
- Nakon specijanog simbola, slede parovi u obliku *key=value* spojeni & simbolom ako tih parametara ima vise od jednog na primer:
  - http://MyService/Persons/1?format=json&encoding=UTF8

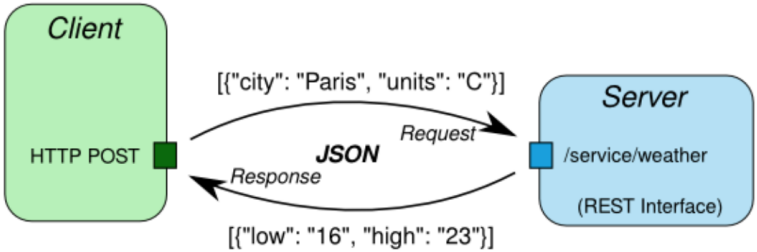
- Servisi koje pozivamo preko internet imaju unapred definisanu strukturu
  - tačnu putanju
  - metod kojim se pozivaju
  - podatke koje očekuju
  - kako se pretražuju
  - itd.
- Ako mi implementiramo web service-e, onda mi namečemo ova pravila
- Pojedinačna operacija nad resursom se često naziva *endpoint*
- Endpoint obično reprezentuje jedan resurs u sistemu

HTTP odgovor

- Nakon zahteva, klijent dobija HTTP odgovor nazad od servisa
- Svaki HTTP odgovor se sastoji od nekoliko elemenata:
  - **[STATUS]** nam govori da li je prethodno zahtevana operacija izvršena uspesno, ili ne. Status code je reprezentovan celim brojem I to:
    - Success 2xx sve je prošlo ok
    - Redirection 3xx desio se redirect na neki drugi servis
    - Error 4xx, 5xx javila se greska
  - **[HEADERS]** dodatni podaci poslati od servera
  - **[BODY]** konkretni podaci



(HTTP Response)


Client
HTTP POST
[{"city": "Paris", "units": "C"}]
Request
JSON
Response
[{"low": "16", "high": "23"}]
Server
/service/weather
(REST Interface)

Graceful shutdown

- Kada korisnik ili operativni sistem prekinu izvršavanje web service-a, mi ne moramo istog momenta da ugasimo naš web service, zato što možda postoje akcije koje se nisu završile, a trebalo bi
- Možda tim akcijama treba dodatno vreme da sačuvaju svoje stanje koje bi inače bilo izgubljeno
- Dobra je praksa, da se web service-u da neko dodatno vreme u kom on neće prihvatati nove zahteve, ali će ostali elementi moći da urade svoj posao
- Na primer, završe svi odgovori korisnicima, završi interkacija sa bazom, sašuvaju logovi itd.

- Za ovaj mehanizam možemo da iskoristimo sistemke pozive (i neke Go-ove pogodnosti)
- Kada dobijemo specifičan sistemski signal možemo da reagujemo na njega
- Ovo postaje bitno kada budemo radili sa kontejnerima, gde drugi sistemi mogu da ugase vašu aplikaciju u svakom momentu
- Mi možemo da zaustavimo web service da više ne prihvata zahteve, ali ostalim akcijama damo proizviljno vreme da urade sve što treba pre nego što se program načisto zaustavi

Uvod

- JavaScript Object Notation (JSON) je format za laksu razmenu podataka, kao i XML nezavistan je od programskog jezika i tehnologije koja se koristi
- Podaci se zapisuju kao parovi **ključ:vrednost**
- Ključ se navodi kao tekst pod duplim navodnicima nakon cega sledi vrednost na primer:
  - "firstName":"John"
- JSON vrednosti su unapred definisane i možmeo izabrati iz konačnog skupa opcija
- JSON podseća na rad sa mapa u drugim programskim jezicima

Unapred definisane vrednosti

- JSON vrednosti mogu biti neke od unapred definisanih:
  - Broj (integer or floating point)
  - String (in double quotes)
  - Boolean (true or false)
  - Niz (koristi uglaste zagrade)
  - Objekat (koristi vitičaste zagrade)
  - Null

Konstrukcija

- JSON objekat se zapisuje u parovima **ključ:vrednost** koji se nalaze unutar vitičastih zagrada:
  - { "firstName":"John", "lastName":"Doe" }
- JSON niz sadrži ključ, nakon čega sledi niz elemenata u uglasim zagradama:

```
{
"employees": [
{"firstName": "John","lastName": "Doe"},
{"firstName": "Anna","lastName": "Smith"},
{"firstName":"Peter","lastName":"Jones"}
]
}
```

Napomena

- JSON može da kombinuje razne tipove podataka unutar jednog niza
- O tome treba voditi računa kada koristimo strogo tipizirane jezike da ne bi došlo do problema prilikom konverzije
- Mešanje tipova treba izbegavati, osim ako nemate baš jaku potrebu sa time
- U tom slučaju trebate naći način da rešite ovaj problem
- Ako koristite dinamičke jezike, ovo nije tako veliki problem

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

- Postavlja se pitanje sta raditi ukoliko imamo više aplikacija, od kojih je neke neophodno pokrenuti u vise instanci (kontejnera), koji moraju da komuniciraju
- Tada posao pojedinačnog kreiranja slika i pokretanja kontejnera nije baš idealan
- Za ove potrebe se koristi alat *docker-compose* koji nam značajno olakšava stvari po tom pitanju
- Omogucuje nam:
  - pokretanje i zaustavljanje aplikacija
  - zejdnički ispis logova svih aplikacija na jedan pseudo terminal
  - jednostavno održavanje mreže i DNS-a

- Sve što je neophodno jeste da se instalira alat
- Zatim je potrebno da kreiramo fajl pod nazivom *docker-compose.yml*
- On sadrži specifikaciju koje sve aplikacije (kontejnere) je potrebno pokrenuti, i kako organizovati mrežu itd.
- Ceo postupak nam omogućava da lako pokrenemo povezane aplikacije i da ih testiramo

Docker compose YAML

- Ovaj fajl je relativno jednostavan, i sadrži dosta direktiva koje možemo upotrebiti:
  - **version** ovde naglašavamo koju verziju formata želimo da koristimo — dovoljno je navesti verziju 3 (poslednja verzija formata)
  - **services** definiše niz objekata gde svaki predstavlja servis, odnosno kontejner
  - Obe sekcije su obavezne
  - Pored ove dve sekcije možemo definiati volumes, mreže

Services

- **services** definiše niz objekata gde svaki predstavlja servis, odnosno kontejner
- Ovaj element ima složenu strukturu, dalje unutar njega definisemo:
  - **build** ova direktiva ako je definisana, govori da je neophodno kreirati slike pri čemu se definišu odnosno putanja do direktorijuma na kojoj se nalazi Dockerfile (može . ako se nalazi na istoj lokaciji kao i Dockerfile)
  - **image** definiše naziv slika koja ce nastati prilikom build-ovanja
  - **container-name** definiše naziv kontejnera koji će biti pokrenut
  - **restart** definiše pod kojim okolnostima kontejner treba restartovati
  - **networks** definiše mrezu (mreze) u kojoj kontejner treba da se nalazi
  - **ports** vrši se mapiranje portova (host:kontejner)
  - **environments** postavlja vrednost environment varijable koje se nalaze u kontejneru
  - **volume** definiše volume za koje se kontejner kači **depends_on** nam govori prilikom pokretanja servisa koje su zavisnosti, odnosno koji servisi moraju biti pokrenuti pre nego sto se pokrene konkretan servis

Dodatni materijali

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

Kraj predavanja

Pitanja? :)