

Servisno orijentisane arhitekture

Predavanje 4: Mikroservisi i DevOps, RPC



Univerzitet u Novom Sadu
Fakultet Tehničkih Nauka

Uvod

- ▶ Termin koji se koristi za praksu u razvoju softvera koji naglašava kolaboraciju i neophodnost konstantne komunikacije između programera (developer) i IT profesionalaca koji nadziru operativnu upotrebu softvera
- ▶ Ova saradnja je usmerena na automatizaciju procesa isporuke softvera i izmena infrastrukture
- ▶ DevOps pristup je jedan od ključnih elemenata za uspešnu implementaciju mikroservisnih arhitektura
- ▶ DevOps predstavlja skup koncepata, praksi, alata i organizacionu strukturu timova koja omogućava da organizacije brže reaguju na potrebe klijenta

- ▶ Organizacije koje koriste ovaj pristup lakše realizuju isporuku i monitoring mikroservisnih sistema
- ▶ Ovakva organizacija omogućava da se brže odgovori na nove zahteve ili da se brže reaguje na probleme u produkciji
- ▶ DevOps obično podrazumeva:
 - ▶ Prakse agilnog razvoja softvera
 - ▶ Kontinuiranu integraciju
 - ▶ Automatizaciju isporuke softvera (release automation)
 - ▶ Funkcionalno testiranje modula
 - ▶ Testiranje integracija sistema
 - ▶ Nadzor servisa i infrastrukture softvera

Organizacija

- ▶ Timovi se organizuju za realizaciju poslovne funkcije i kontorlišu kompletan životni ciklus datog servisa:
 - ▶ definisanje i analiza zahteva
 - ▶ razvoj
 - ▶ testiranje
 - ▶ isporuka
 - ▶ nadzor operacija servisa (nadzor u produkciji)

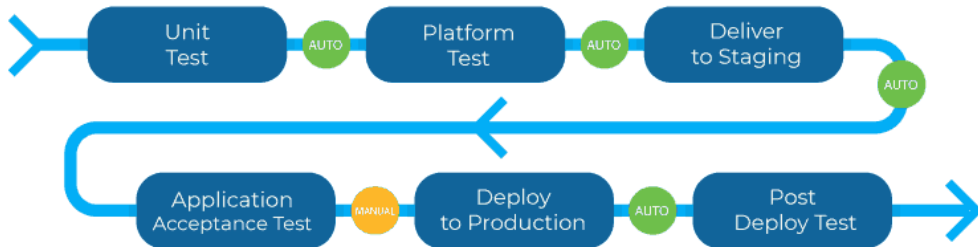
- ▶ Paralelno sa reorganizovanjem timova neophodno je razmisliti i o organizovanju DevOps timova za podršku
 - ▶ ovi timovi obezbeđuju alate, praćenje zavisnosti, upravljanje, upravljajju vidljivošću pojedinih servisa
 - ▶ ovi timovi istovremeno oezbeđuju i bolji uvid u rad mikroservisnih timova i samim tim bolju povratnu informaciju klijentima
- ▶ DevOps servisni timovi obezbeđuju neophodnu dostupnost informacija o tome koji servisi se isporučuju, koji servisi će biti korišćeni od strane ostalih timova i od strane krajnjih korisnika.

Benefiti

- ▶ Obezbedjuje kontinualnu isporuku putem tri osnovna koncepta:
 1. automatizacija – sve tradicionalne tačke primopredaje se u ovom pristupu automatizuju, maksimalna korist je i od automatizacije testiranja. Kako mikroservisi obično predstavljaju male poslovne funkcije nivo automatizacije koji se može postići je dosta visok.
 2. standardizacija – visok nivo pouzdanosti u momentu kada se dostigne vreme isporuke
 3. česte isporuke koda – omogućavaju da aplikacija ostane relevantna i dobro usaglašena sa poslovnim potrebama klijenta

DevOps continuous delivery

Continuous Delivery Model



(<https://www.plutora.com/blog/continuous-delivery-model>)

- ▶ Kontinuirana isporuka predstavlja praksu koja omogućava da se kod može brzo i sigurno primeniti na prdukciono okruženje, tako što se svaka promena isporučuje na okruženje koje je skoro identično produkcijom
- ▶ Automatizovanim testovima proverava se da aplikacije i servisi funkcionišu kao što je i očekivano
- ▶ Nakon uspešnog testa na ovakvom okruženju, date izmene se na produkciju mogu prebaciti “na klik”.
- ▶ Često korišćeni alati: Chef, Puppet, Docker, Ansible, Salt, Helm

Kontinuiran nadzor

- ▶ Omogućava nadzor koji pomaže razvojnim timovima i testerima da shvate performanse i dostupnost aplikacija, čak i pre nego se isporuče u produkciju
- ▶ Rana dostupnost ovih povratnih informacija je bitna za smanjenje troškova ispravljanja grešaka i nakandnih neophodnih izmena, i pomaže da se projekti uspešno privode kraju
- ▶ U produkciji tim za operativno upravljanje (Ops tim) obezbedjuje da se aplikacija ponaša kako je očekivano, i da je posmatrano okruženje stabilno
- ▶ Ponekad je neophodno da se u saradnji sa razvojnim timovima definišu posebni alati za nadzor kritičnih funkcija sistema

Kontinuirano prikupljanje povratnih informacija i optimizacija

- ▶ Efikasan DevOps mora omogućiti brzo prenošenje povratnih informacija od klijenta
- ▶ Na ovaj način razvojni tim dobija uvid u ponašanje korisnika kao i slabe tačke aplikacije koje korisniku predstavljaju problem za korišćenje
- ▶ Prednosti ovakvog pristupa:
 - ▶ Daje odredjeni nivo kontrole putem povratnih informacija svim zainteresovanim
 - ▶ Omogućava brz odgovor na zahteve
 - ▶ Omogućava da održite kopetitivne prednosti aplikacije

RPC Zahtevi


- ▶ Remote Procedure Call (RPC) je moćna tehnika za konstruisanje mrežnih aplikacija zasnovanih na klijent-server mehanizmu
- ▶ Zasnovan je na proširenju konvencionalnog lokalnog pozivanja procedure tako da pozvana procedura ne mora da postoji u istom adresnom prostoru kao i procedura koja poziva
- ▶ Dva procesa mogu biti na istom sistemu, ili mogu biti na različitim sistemima sa mrežom koja ih povezuje
- ▶ Prilikom RPC zahteva:
 1. Pozivno okruženje je suspendovano, parametri procedure se prenose preko mreže u okruženje u kome se procedura treba izvršiti, i procedura se tamo izvršava
 2. Kada se procedura završi i proizvede svoje rezultate, njeni rezultati se prenose nazad u okruženje koje poziva, gde se izvršavanje nastavlja kao da se vraća iz redovnog poziva procedure

- ▶ RPC je posebno pogodan za interakciju klijent-server (npr. upit-odgovor) u kojoj se tok kontrole menja izmedju pozivaoca i pozivaoca
- ▶ Konceptualno, klijent i server se ne izvršavaju u isto vreme
- ▶ Umesto toga, nit izvršenja skače sa pozivaoca na pozivaoca i zatim nazad
- ▶ RPC zahteci su više orijentisani ka akcijama, umesto ka podacima kao što je to REST
- ▶ RPC obezbedjuje APSTRAKCIJU, tj. priroda mrežne komunikacije koja prenosi poruku je skrivena od korisnika

- ▶ RPC često izostavlja mnoge slojeve protokola da bi poboljšao performanse – čak i malo poboljšanje performansi je važno jer program često poziva RPC-ove
- ▶ Sa RPC kodom naponi za ponovno pisanje / ponovni razvoj su minimizirani
- ▶ Danas se koristi u velikim sistemima za implementaciju mikroservisa (Google implementira RPC mehanizam)
- ▶ Obično je ulaz u sistem REST, da pokrije veliki broj raznih klijenata, a ostatak infrastrukture je RPC
- ▶ Neki alati, kao na primer gRPC, omogućavaju i integraciju za klasičnim REST servisima tako što koristi *gateway* pre samog servisa koji je implementiran kao REST

RPC vs REST

API ARCHITECTURAL STYLES				
	RPC	SOAP	REST	GraphQL
Organized in terms of	local procedure calling	enveloped message structure	compliance with six architectural constraints	schema & type system
Format	JSON, XML, Protobuf, Thrift, FlatBuffers	XML only	XML, JSON, HTML, plain text,	JSON
Learning curve	Easy	Difficult	Easy	Medium
Community	Large	Small	Large	Growing
Use cases	Command and action-oriented APIs; internal high performance communication in massive micro-services systems	Payment gateways, identity management CRM solutions financial and telecommunication services, legacy system support	Public APIs simple resource-driven apps	Mobile APIs, complex systems, micro-services



(shorturl.at/sIVZ3)

Dodatni materijali

- ▶ Building Microservices, Sam Newman
- ▶ Microservices • Martin Fowler • GOTO 2014
- ▶ What are microservices?
- ▶ Microservices patterns
- ▶ What is devops
- ▶ Remote Procedure Calls (RPC)
- ▶ RPC intro
- ▶ gRPC

Kraj predavanja

Pitanja? :)