Servisno orijentisane arhitekture

Predavanje 10: Mikroservisi i paterni, Obrasci za isporuku aplikacija, Dekompozicija sistema u servise

Univerzitet u Novom Sadu
Fakultet Tehničkih Nauka

Obrasci za isporuku (deployment) aplikacija

- Više instanci servisa po jednom hostu
- Jedna instanca servisa po jednom hostu
- Jedna instanca servisa na jednoj VM
- Jedna instanca servisa po kontejneru
- Implementacija bez servera (serverless deployment)
- Platforme za isporuku (deployment) servisa

Pretpostavke

- Ove pretpostavke važe za sva predložena rešenja
- Aplikacija je razvijena korišćenjem mikroservisne arhitekture i sistem je skup servisa.
- Svaki servis se instalira kao skup servisnih instanci kako bi se obezbedila neophodna raspoloživost i propusni opseg.

Problem

- Kako se spojedinačni servisi pakuju i isporučuju?

Faktori koji utiču na izbor

- Servisi su napisani na različitim jezicima i korišćenjem različitih razvojnih okvira i/ili verzija
- Svaki servis se isporučuje kroz više instanci servisnih modula
- Neophodno je obezbediti mogućnost nezavisne isporuke i skaliranja servisa.
- Pojedinačne istance servisa moraju biti medjusobno izolovane
- Neophodne je obezbediti mogućnost brzog razvoja i isporuke servisa
- Neophodno je ograničiti resurse (CPU, memoriju) koje servis troši
- Neophodno je obezbediti mogućnost nadgledanja svakog servis
- Instalirani servis treba da je pouzdan
- Aplikacija treba da se isporuči maksimalno ekonomično

Više instanci servisa po jednom hostu

- Moguće rešenje – izvršavanje više instanci različitih servisa na istom host (fizičkom serveru ili virtuelnoj mašini).
- Postoji više načina da se obezbedi ovakva instalacija, neki od njih:
  - instalirati pojedinačne servise kao nezavisne JVM procese (npr. jedan Tomcat server po instalirati više instanci servisa u istu JVM
  - instalirati više instanci servisa u istu JVM

Osobine

- Dobre osobine
  - Dobro korišćenje raspoloživih resursa u odnosu na izbor jedne instance servisa po hostu.
- Loše osobine
  - Rizik od mogućeg utrkivanja za resurse.
  - Rizik da se pojave oprečni zahtevi u pogledu verzija nekih biblioteka.
  - Teško je ograničiti resurse dodelje pojedinačnim instancama servisa.
  - Ukoliko je više instanci servisa instalirano u isti proces, teško je nadgledati korišćenje resursa od strane pojedinačnih instanci, a i teško je obezbediti medjusobnu izolaciju.

Jedna instanca servisa po jednom hostu

- Moguće rešenje - Instalirati svaki servisnu instancu na sopstvenom hostu.
- Dobre osobine
  - Servisne instance su medjusobno izolovane
  - Nema mogućnosti za konflikte oko resursa ili oko verzija bilioteka
  - Svaka instanca može konzumirati najviše onoliko resursa koliko nudi njen host
  - Pojednostavljen je monitoring, reinstalacija i upravljanje svakom instancom servisa
- Loše osobine
  - Potencijalno manje efikasna upotreba resura nego u slučaju više instanci servisa po hostu.

Jedna instanca servisa po jednoj VM

- Moguće rešenje - Zapakovati servis kao sliku VM, i svaku servisnu instancu pokrenuti kao posebnu VM.
- Dobre osobine
  - Pojednostavljuje skaliranje servisa - prostim podizanjem nove instance VM
  - VM enkapsulira tehnologiju korišćenu za razvoj servisa, Sve servisne instance se pokreću i zaustavljaju na isti način. Servisne instance su medjusobno izolovani.
  - VM definiše limit upotrebe CPU i memorije
  - IaaS rešenja obezbedjuju dobre alate za upravljanje VM
- Loše osobine
  - Proces pripreme VM je spor i zahtevan.
  - Proces pokretanja može biti spor
  - Bacanje resursa u nekim situacijama

Jedna instanca servisa po jednom kontejneru

- Moguće rešenje - Zapakovati servise kao sliku kontejnera, pokretati svaku servisnu instancu kao kontejner. Postoje i okruženja za upravljanje klasterima kontejnera (Kuberneti, Marathon/Mesos, Docker Swarm, Rancher, ...)
- Dobre osobine
  - Pojednostavljeno je skaliranje servisa (naviše i naniže) - promenom broja aktivnih instanci kontejnera (servisnih instanci).
  - Kontejner enkapsulira sve tehnološke detalje servisa. Pokretanje i zaustavljanje svih servisa radi se na isti način.
  - Izolacija svake servisne instance.
  - Kontejner postavlja limite na raspoložive resurse za svaku servisnu instancu.
  - Kontejneri se brzo grade i pokreću. Vreme pripreme kontejnerske slike, instalacije i pokretanja kontejnera je višestruko brže od VM.
- Loše osobine
  - Infrastrukturni alati za upravljanje kontejnerima su nešto manje bogati nego oni za upravljanje VM.

Implementacija bez "servera" – serverless

- Moguće rešenje - koristiti neku od postojećih infrastruktura za isporuku servisnih aplikacija. One "sakrivaju" koncept servera (rezervaciju resursa ili alokaciju resursa po unapred zadatim uzorcima - zauzimanje fizičkog ili virtuelnog servera, ili kontejnera). Sama infrastruktura prihvata KOD servisa i izvršava ga. Naplaćuje se po upućenim zahtevima ka datom servisu.

Osobine

- Da bi se servis isporučio na ovaj način kod se pakuje, uploaduje na deployment infrastrukturu i zadaju se željene specifikacije u pogledu performansi. Deployment infrastruktura je uslužni servis koji nude ponudjač usluga računarstva u oblaku (cloud provder-i).
- TIpično koriste bilo VM ili kontejnere za izolaciju pojedinih servisa.
- Ali ovo je skriveno od krajnjeg korisnika.
- Nijedan korisnik ovakve usluge nije zadužen da vodi računa o infrastrukturnim rešenjima (OS, VM...)

- Kod je obično jedna funkcija aplikacije
- Kada se dogodi očekivani dogadjaj cloud provider pronalazi slobodnu instancu funkcije (ili pokreće novu) i izvršava njen kod.
- Uvek se obezbedjuje potreban broj instanci da se podrži opterećenje koje korisnici stvaraju.
- Funckije je moguće pogoditi na razne načince
  - Dogadjaj
  - Direktan poziv
  - Cron-like mehanizmima po unapred poznatim vremenskim intervalima

Osobine

- Dobre osobine
  - Uklanja potrebu da se obavlja zahtevno konfigurisanje infrastrukturnih detalja.
  - Infrastruktura za podršku ovakvoj isporuci je jako elastična. Automatski se skalira u skladu sa potrebama.
  - Plaća se po izvršenom zahtevu, a ne kao većina drugih po rezervisanim resursima (koji se možda i ne koriste u punoj meri uvek npr. VM sa malo opterećenja).
- Loše osobine
  - Ozbiljna ograničenja - mnogo veća nego kod VM ili kontejnera. Npr. podrška samo za nekoliko programskih jezika.
  - Pogodne su samo za stateless implementacije, nisu pogodne za statefull aplikacije koje bi se dugo izvršavale.
  - Ograničen broj "ulaza" - lambde mogu reagovati samo na odredjene dogadjaje iz odredjenih izvora.
  - Aplikacija podržana na ovaj način mora jako brzo da se startuje - nisu pogodni za servise kojima treba mnogo vremena da se dovedu u operativno stanje.
  - Postoji rizik od relativno velike latentnosti - vreme koje je infrastrukturi potrebno da instancira funkciju, kao i vreme njene inicijalizacije može biti osetno za korisnika.

Dekompozicija sistema u servise

- Dekompozicija po poslovnim funkcionalnostima dekompozicija po poddomenima
- Samostalan (self-contained) servis
- Jedan servis po timu šablon

Dekompozicija po poslovnim funkcionalnostima

- Kontekst u kome se koristi - razvoj kompleksne aplikacije uz opredeljenje da se ona razvije na mikroservisnoj arhitekturi. Ovo podrazumevas strukturu aplikacije u formi slabo zavisnih servisa.
- Problem - kako dekomponovati sistem u servisne komponente?

Faktori koji utiču na izbor rešenja

- Arhitektura mora biti stabilna.
- Servisi moraju biti kohezivni. Jedan servis može da implementira mali skup medjusobno povezanhi funkcija.
- Servisi treba da su u skladu sa Common Closure Principle - stvari čije izmene su medjuzavisne (menjaju se zajedno) treba i da su upakovane u zajednički modul.
- Servisi treba da su slabo povezani
- Servis treba da je moguće nezavisno testirati.
- Svaki tim koji je zadužen za servis mora biti autonoman

Rešenje

- Definisati servise u skladu sa poslovnim funkcionalnostima
- Poslovna funkcionalnost je često direktno povezana sa odredjenim tipovima poslovnih objekata (domenskih podataka):
  - upravljanje porudžbinama
  - upravljanje finansijskim transakcijama...

Rezultat

- Stabilna arhitektura sistema, jer su poslovne funkcionalnosti često dosta stabilne - relativno slabo promenljive.
- Razvojni timovi su višefunkcionalni, autonomni i organizovani tako da isporuče odredjenu poslovnu funkcionalnost (više nego što su orijentisani samo na tehničke detalje).
- Servisi razdvojeni na ovaj način su interno kohezivni, a medjusobno slabo zavisni.
- Problemi
  - Kako tačno uočiti poslovne funkciopnalnosti? - Zahteva se razumevanje poslovnog modela, a često i organizacione strukture, poslovnih procesa organizacije itd.

Dekompozicija po poddomenima

- Definisati servise u skladu sa DDD (domain driven design) poddomenima. DDD definiše domen - prostor problema koje aplikacija rešava. Domen se deli na poddomene, pri čemu svaki od poddomena odgovara odredjenom delu poslovanja - rešava uži problem.
- Poddomene možemo klasifikovati kao:
  - osnovni (core) - u ovom domenu se obavljaju ključne poslovne operacije (one bez kojih pomoćni - u ovom domenu se obavljaju poslovne aktivnosti koje potpomažu osnovne poslovanje ne donosi nikakvu korist).
  - pomoćni - u ovom domenu se obavljaju poslovne aktivnosti koje potpomažu osnovn operacije, ali nisu ključne (mogu se i outsource-ovati).
  - opšti (generički) - u ovom domenu obavljaju se poslovne aktivnosti koje su opšte - idealno za podršku ovim aktivnostima koristiti neko generičko rešenje.

Rezultat primene

- Stabilna arhitektura sistema jer su poslovne funkcionalnosti relativno slabo promenljive.
- Servisi razdvojeni na ovaj način su interno kohezivni, i medjusobno slabo zavisni.
- Razvojni timovi su višefunkcionalni, autonomni i organizovani tako da isporuče odredjenu poslovnu funkcionalnost (više nego što su orijentisani samo na tehničke detalje
- Potencijalan problemi – kako tačno uočiti poddomene? - Zahteva se razumevanje poslovnog modela, a često i organizacione strukture, poslovnih procesa organizacije itd.
- Dobre polazne tačke za identifikaciju su:
  - organizaciona struktura
  - domenski model visokog nivoa

Samostalan (self-contained) servis

- Dizajnirati servis tako da može odgovoriti na sinhrone zahteve klijenta bez čekanja na odgovore drugih komponenti.
- Jedan način da se ovo postigne jeste da se kompletna funkcionalnost složene operacije implementira unutar jednog servisnog moduls (spajanjem postojećih servisa). Ali to nije uvek (skoro nikad) dobro rešenje.
- Drugi način da se ovo implementira jeste da se saradnja izmedju servisa obezbedi korišćenjem drugih šablona (CQRS, SAGA). Na taj način samostalni servis bi koristio Saga šablon da asinhronim (dakle neblokirajućim) porukama obezbedi konzistentnost podataka, a CQRS šablon bi koristio da obezbedi (opet asinhrono) održavanje odgovarajuće replike podataka čiji je vlasnik neki drugi servis, a ta replika datom servisu obezbedjuje visoku responzivnost (brz odziv).

Jedan servis po timu

- Timovi treba da su mali (5-9 ljudi).
- Tim treba da je autonoman i slabo zavisan od drugih timova.
- Fino granulirani servisi poboljšavaju održivost, dostupnost i mogućnost testiranja koda
- Ali fino granuliranje servisa povećava kompleksnost celokupnog rešenja
- Količina i složenost koda ne sme da prevazilazi kapacitet tima

Dodatni materijali

- Building Microservices, Sam Newman
- Microservices • Martin Fowler • GOTO 2014
- What are microservices?
- Microservices patterns

Kraj predavanja

Pitanja? :)