Mikroservisi

Šta su mikroservisi?

- Još jedan stil arhitekture softverskih sistema, u kome se velike složene aplikacije komponuju sklapanjem pojedinačnih servisa.
  - Koncept nije potpuna novina - predstavlja samo još jednan pristup implementaciji SOA
  - Mikroservisi mogu biti nezavisno *deployovani* i međusobno slabo spregnuti
  - Kod mikroservisnih arhitektura pojedinačni servisi obavljaju jedan zadatak
    - Taj jedan zadatak predstavlja jednu poslovnu funkciju celokupnog sistema

Osnovne karakteristike?

- Svaki mikroservis moguće je razvijati u programskom jeziku koji je najpogodniji, nezavisno od svih ostalih.
- Komunikacija između mikroservisa se obavlja programskim interfejsima API-jima koji su nezavisni od programskog jezika (npr. Representational State Transfer (REST).
- Mikroservisi (moduli koji ih realizuju) imaju potpuno ograničen kontekst - ne moraju biti svesni nikakvih implementacionih detalja i arhitekture drugih mikroservisih modula.

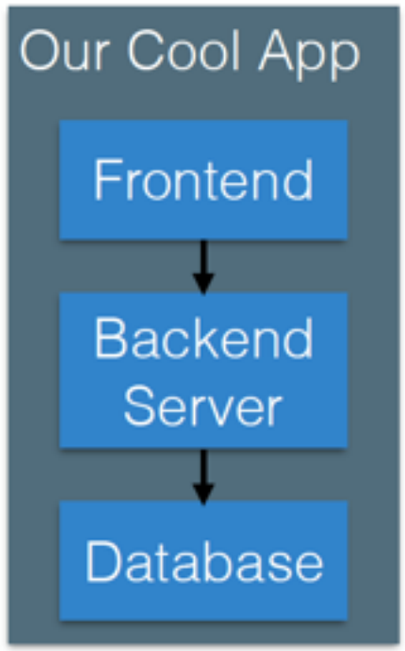
Osnovne karakteristike?

- Svaki mikroservis moguće je razvijati u programskom jeziku koji je najpogodniji, nezavisno od svih ostalih.
- Komunikacija između mikroservisa se obavlja programskim interfejsima API-jima koji su nezavisni od programskog jezika (npr. Representational State Transfer (REST).
- Mikroservisi (moduli koji ih realizuju) imaju potpuno ograničen kontekst - ne moraju biti svesni nikakvih implementacionih detalja i arhitekture drugih mikroservisih modula.

Zašto razmišljati o mikroservisnoj arhitekturi?

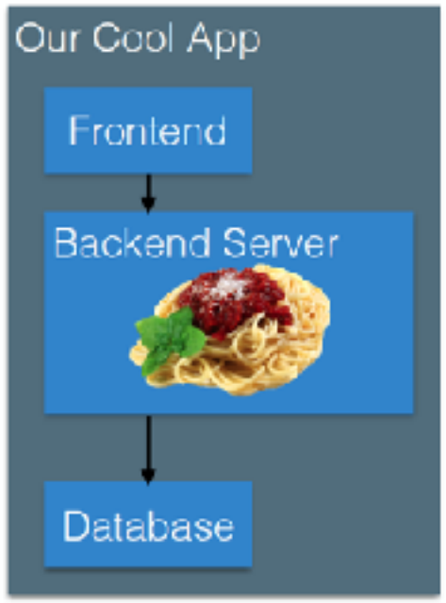
- Monolitne nasuprot mikroservisnim arhitekturama

- Kako najčešće gradimo velike softverske sisteme?
- Najčešće ih sagledavamo izdeljene po slojevima
- **Višeslojne** arhitekture
  - klijentski sloj
  - sloj biznis logike
  - sloj podataka

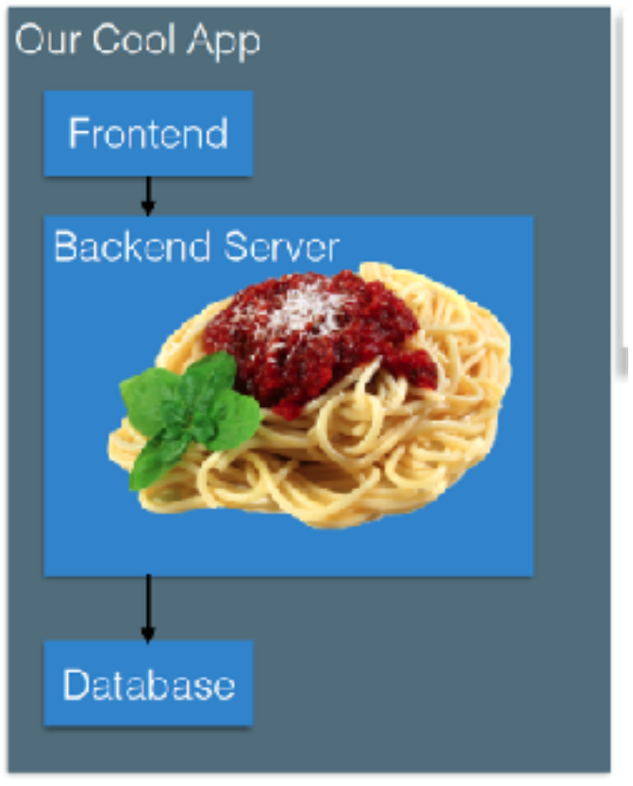


Šta se dešava kada višeslojnoj arhitekturi dodajemo nove funkcionalnosti?

- Obično se poveća složenost *backend* sloja (poslovna logika)

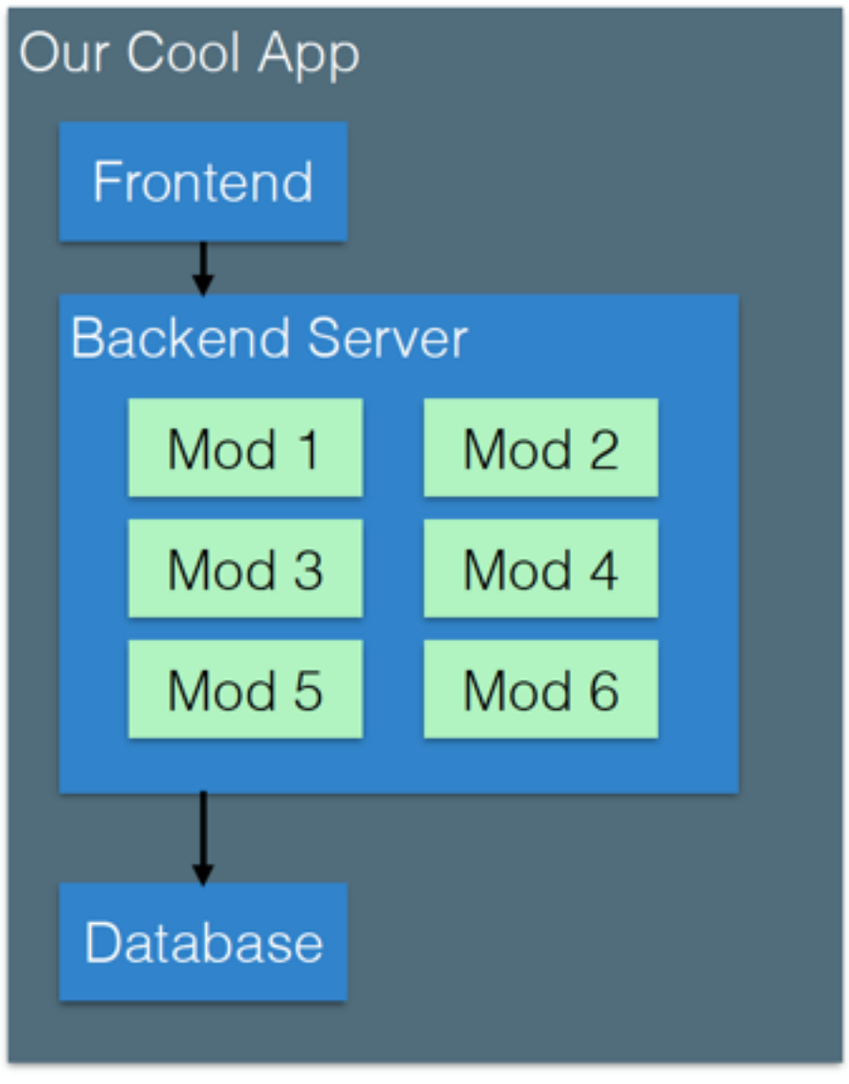


- Ako želimo još više funkcionalnosti?
  - značajno zakomplikujemo srednji sloj



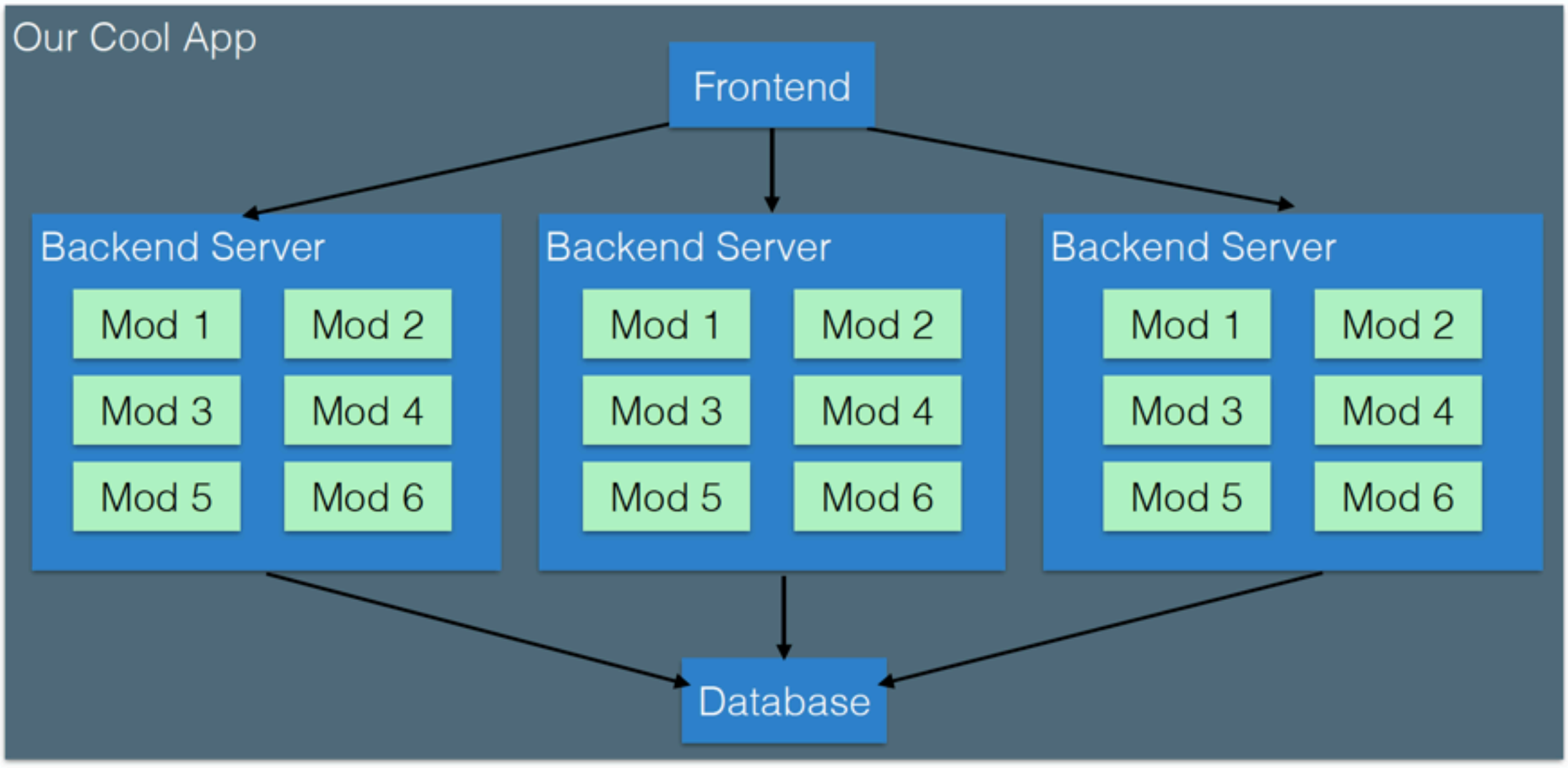
Šta se dešava kada višeslojnoj arhitekturi dodajemo nove funkcionalnosti?

- Kod je na sreću ipak organizovan po modulima, pa povećanje broja funkcija dovodi do povećanja broja modula u srednjem sloju aplikacije



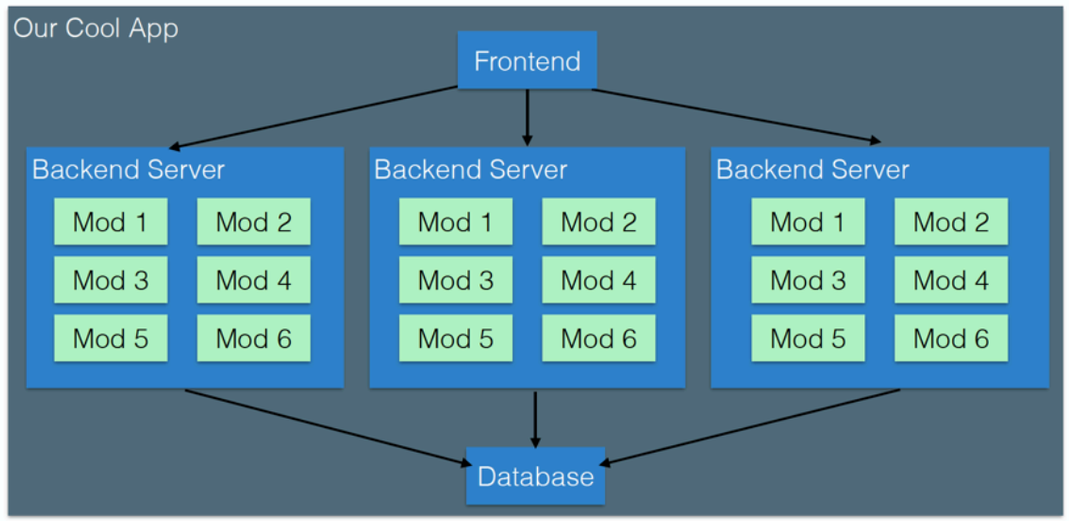
Kako se skaliraju ovakve aplikacije da odgovore povećanim zahtevima?

- Pokreće se više instanci *backend* aplikacija sa identičnim modulima kako bi odgovorile na povećane zahteve



Kako se skaliraju ovakve aplikacije da odgovore povećanim zahtevima?

- Ovakav princip skaliranja je tipičan za monolitne aplikacije
- Ako imamo veliki broj servera, na svakom se “vrte” isti moduli
- Šta ako su nam neke funkcije sistema više opterećene nego neke druge?
- Da li je baš najoptimalnije da su nam svi moduli implementirani na isti način (isti programski jezik, isti runtime, koriste istu bazu…)



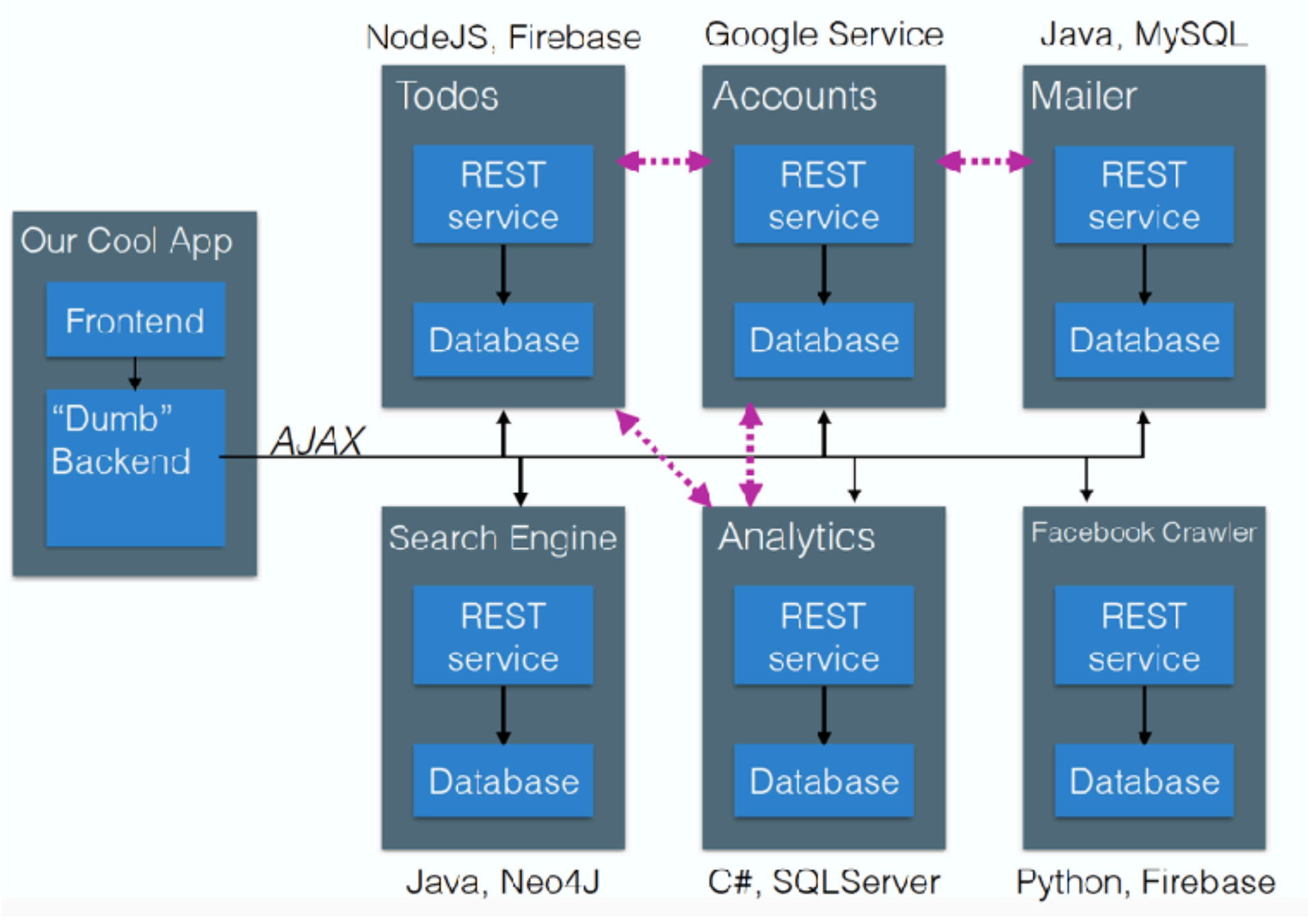
Mirkoservisni pristup?

- Mikroservisi se orijentišu na jednostavnu poslovnu funkcionalnost - jedan zadatak, i kao takvi su po pravilu mali moduli.
- Nema pravila koliko mali moraju biti, i ne treba se koncentrisati na broj linija koda nego na funkcionalnost.
  - “Pravilo 2 pizze” - ako vam je tim koji je neophodan da realizujete mikroservisni modul toliko veliki da ne možete da ga nahranite sa samo 2 pizze - nešto ste omašili
- Ključna je jednostavnost interfejsa - ona obično dovodi i do relativno male implementacije, ali to ne mora uvek biti slučaj.
- Mikroservisni modul treba tretirati kao nezavisnu aplikaciju ili nezavisni proizvod. Poželjno je da ima sopstveni repozitorijum za upravljanje kodom, i sopstveni *build* i *deployment.*

Ponovna iskoristivost i granularnost mikroservisa

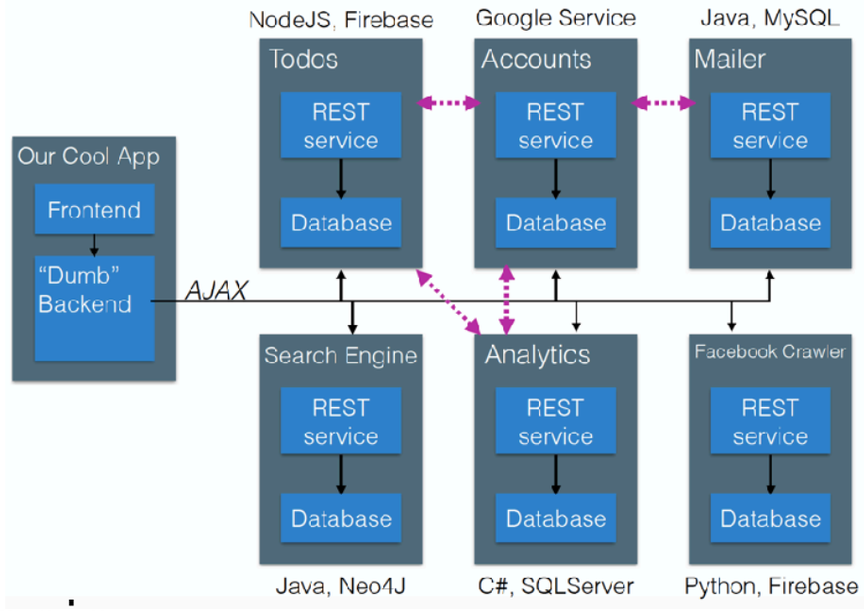
- Iako je ponovna iskoristivost poželjna nije i obavezna i nije jedini razlog njihovog uvođenja
  - lokalne optimizacije korisničkog interejsa kako bi se poboljšao odziv,
  - lakše prilagođavanje potrebama korisnika…
- Granularnost mikroservisa se takođe određuje na osnovu poslovnih potreba
  - npr. praćenje paketa, prognoze vremena se danas vrlo često mogu koristiti kao servisi treće strane
- Problem latentnosti servisa - ukoliko je previše usitnjen i zateva previše poziva ka drugim mikroservisima, može se osetiti problem usporenja aplikacije

Kako izgleda složena aplikacija realizovana po prinicipima mikroservisa

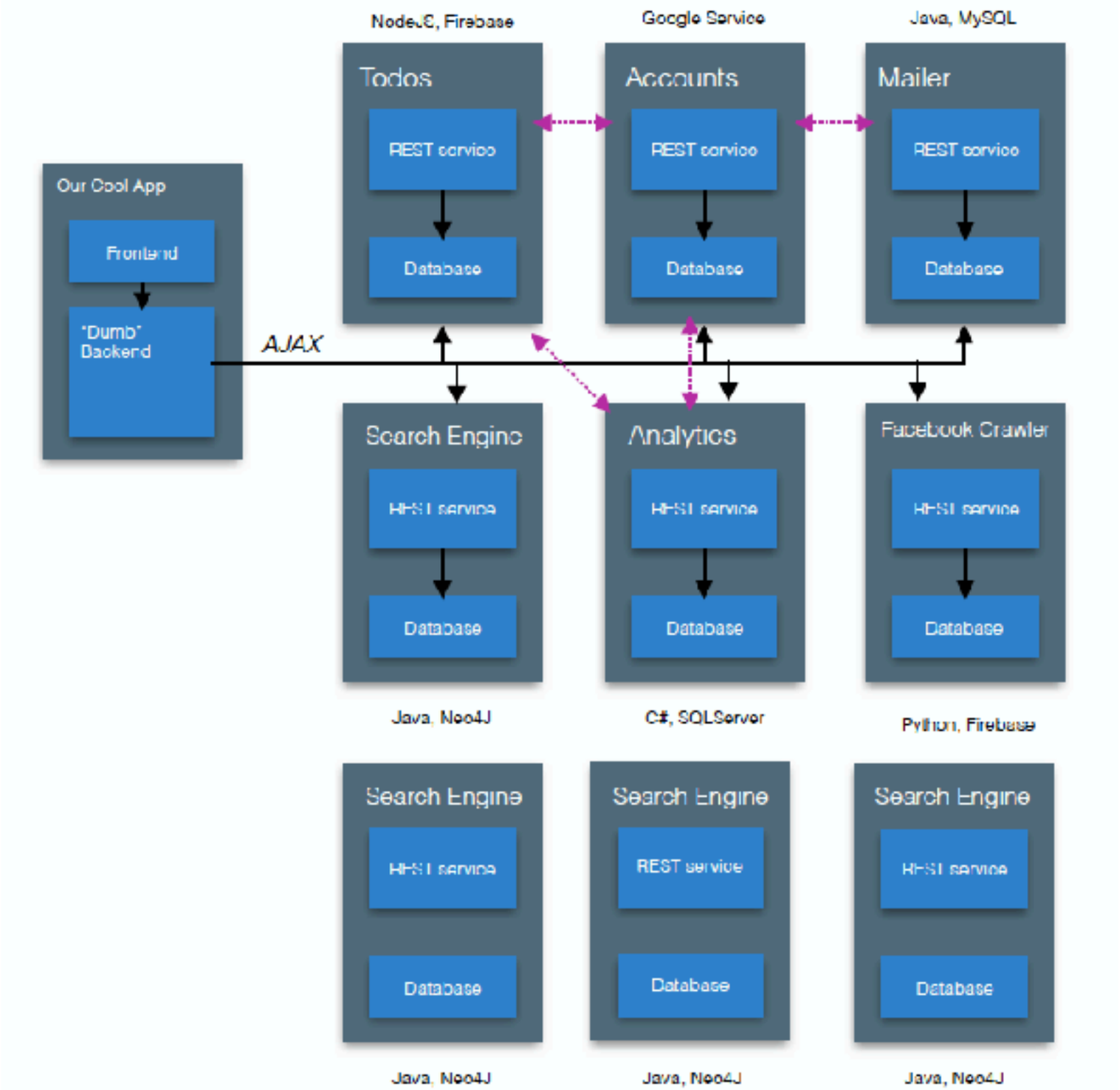


Šta je ovde bitno različito

- Nema glomaznih modula sa “špageti” kodom
- Komponente se mogu razvijati potpuno nezavisno jedna od druge
  - različiti programski jezici, izvršna okruženja, OS, hardware, DB
- Komponente se mogu relativno jednostavno menjati
  - Može se zameniti i kompletna tehnologija modula
- Moguće različito skaliranje različitih komponenti



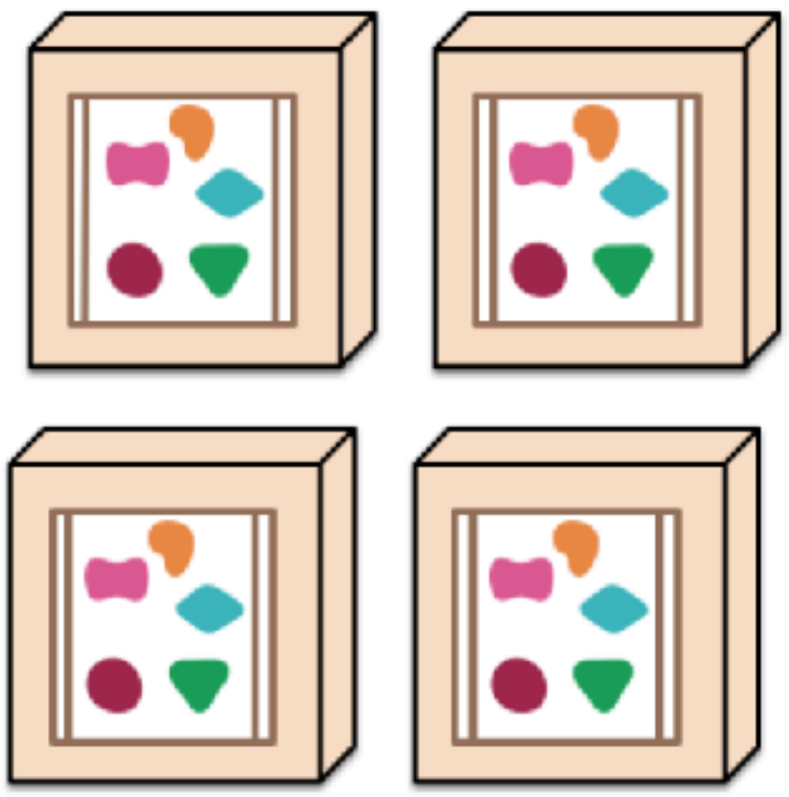
Kako se skaliraju mikroservisne aplikacije da odgovore povećanim zahtevima?



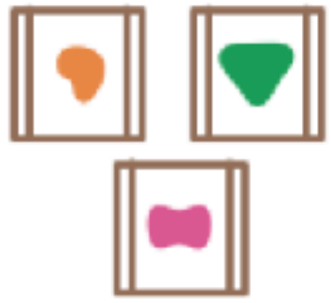
Razlika u skaliranju

A monolithic application puts all its functionality into a single process...

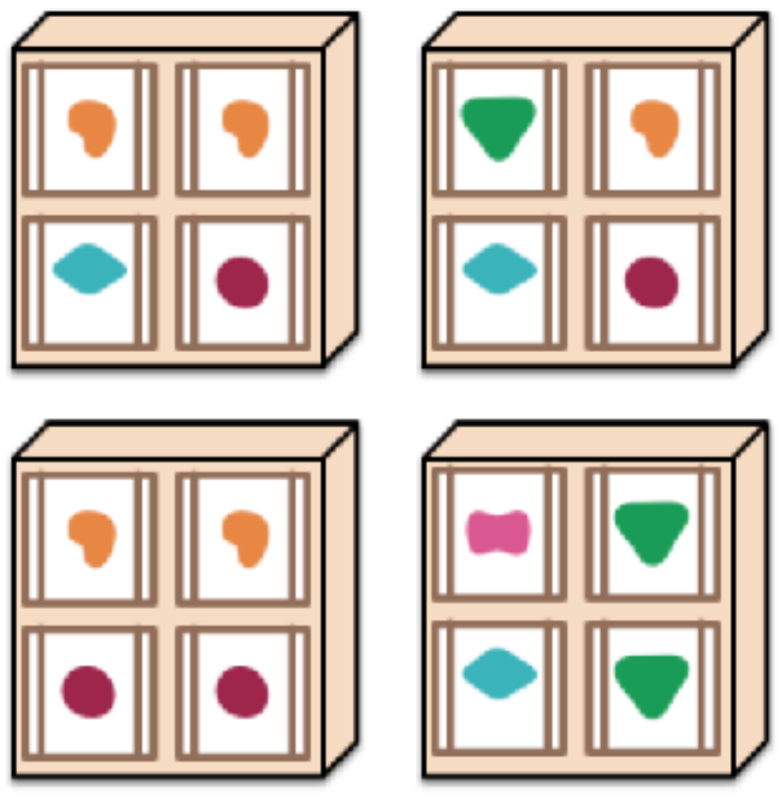
... and scales by replicating the monolith on multiple servers



A microservices architecture puts each element of functionality into a separate service...



... and scales by distributing these services across servers, replicating as needed.



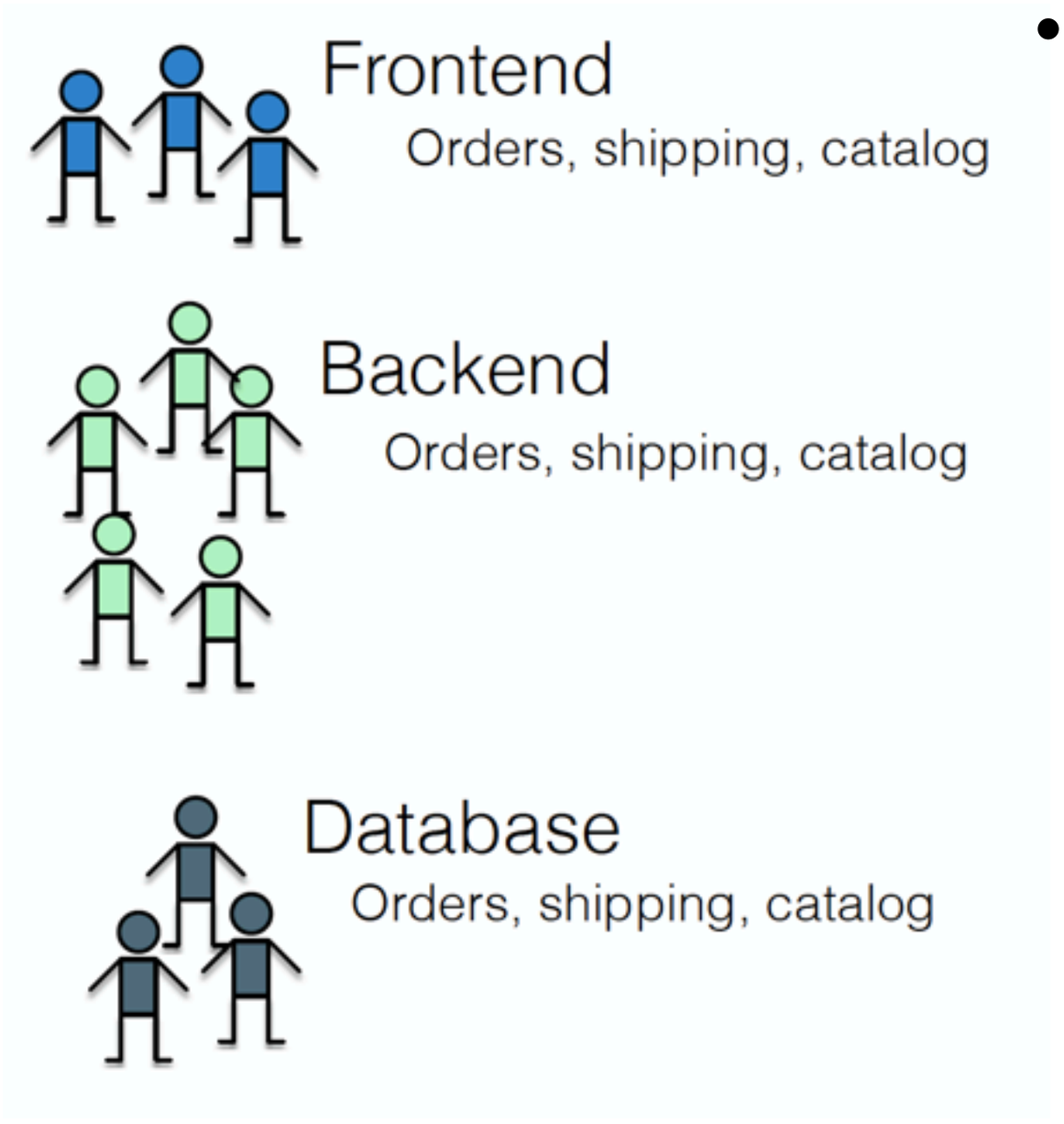
Pretpostavke za uspešnu mikroservisnu arhitekturu

- 1 komponenta = 1 servis
- “pametni” endpointi i “glupi” komunikacioni kanali (za razliku od npr. ESB, koji vrlo često sadrži složene mehanizme rutiranja, transformacija i sl.)
- decentralizovano upravljanje
- decentralizovano upravljanje podacima
- automatizacija infrastrukture
- dizajnirati arhitekturu da trpi otkaze
- evolutivni dizajn

Koliko velike komponente treba da budu?

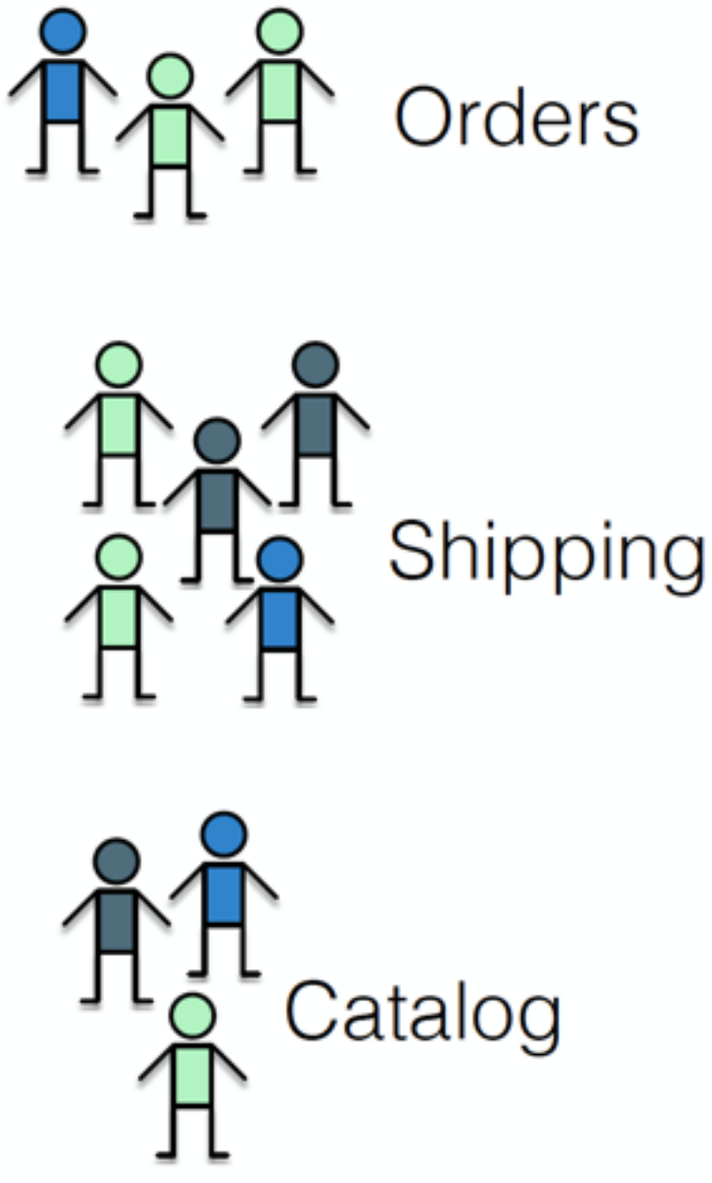
- Komponente se mogu formirati kao
  - biblioteke (moduli)
  - servisi
- Mikroservisi promovišu modularizaciju pomoću servise
  - svaku komponentu možete pojedinačno zameniti
  - svaku komponentu možete nezavisno ažurirati
- Ovo znači da se mogu i nezavisno razvijati, testirati…

Klasična organizacija razvoja



- 1 tim po sloju, i svaki tim se bavi funkcionalno-stima koje su u datom sloju bez obzira na to koji segment aplikacije opslužuju

Mikroservisi - organizacija razvoja usmerena na prozivod ili segment poslovanja



- 1 tim se bavi jednim segmentom poslovanja, obezbeđuje funkcionalnost za jedan proizvod i sastoji se od ljudi svih profila koji su neophodni da taj segment proradi
- Timovi se koncentrišu na samo jedan poslovni zadataka i mogu da komuniciraju direktno sa klijentima

Decentralizovano upravljanje

- Odluke o načinu implementacije se donose decentralizoanvo
- Servisi podatke razmenjuju ISKLJUČIVO preko javno dostupnih API-ja, nema oslanjanja na deljene baze podataka
- Omogućava svakom servisu da podacima upravlja na način koji je najpogodniji sa stanovišta tog servisa

Automatizacija infrastrukture

- Za efikasno korišćenje mikroservisnih arhitetkruea najverovatnije će biti neophodno da:
  - Razvijate servise nezavisno
  - Servise nezavisno puštate u rad (deployment)
- Morate obezbediti:
  - mogućnost da dobijete serverse kapacitete brzo kako bi mogli da iskoristite skalabilnost rešenja
  - dobar monitoring kako bi bili u stanju da vidite kada servisi ne komuniciraju kako treba
  - brz *deployment* novih ili ažuriranih servisa
  - razvijenu kulturu jake integracije timova koji rade nadzor servisa u radu i tim akoji radi razvoj ("devops")
- Ključna stvar za uspeh mikroservisa je automatizacija svega navedenog

Dizajniranje sistema tako da bude otporan na otkaze

- Konceptom mikroservisa ukupna struktura sistema može ozbiljno da se zakomplikuje
- Mnogo “pokretnih” delića koji mogu da otkažu:
  - Pojedini servisi mogu imati greške
  - Pojedini servisi mogu raditi vrlo sporo
  - Celi serveri mogu pasti
  - Sa 60,000 HD ova 3 dnevno će verovanto otkazati
- Ključna stvar - dizajnirati svaki servis pretpostavljajući da u nekom momentu sve ono od čega taj servis zavisi može prosto da nestane i bude nedostupno
  - servis tada mora otkazati “gracefully”
  - Netflix - “ChaosMonkey”

Održavanje konzistencije

- Jedno od pravila mikroservisa je ne koristiti deljene baze podataka
- Neke podatke ipak verovatno koristi više servisa
- Ažuriranje se šalje preko AJAX poziva
- Nema garancije da će svi moduli obaviti ažuriranje u istom trenutku
- Garantuje se da će oni u nekom momentu ažurirati podatke (**evenutal consistency**)

Održavanje konzistencije

- Glavni problem je što različiti servisi mogu odgovoriti na zahtev u različitim vremenskim trenucima
- Šta ako jedan zahtev rezultuje promenom resursa na jednom servisu, ali ostali još nisu procesirali korespondirajuće zahteve?
  - Moguće je da se dobiju nekonzistentna stanja za korespondirajuće resurse.
  - Mora se napisati dodatna logika da korektno obradi ovakve situacije.

Kada koristiti koji pristup arhitekturi?

- Monolitne:
  - Jednostavnija za razvoj
  - Mikroservisi zahtevaju distribuiranu obradu, mnogo asinhronih poziva
  - Kada je apsolutna sadržaj između modula - dobra za početne faze razvoja
- Mirkoservisi:
  - Kada nam treba parcijalna implementacija
  - Netflix ima vrlo pozitivno iskustvo - praktično mu omogućava dnevne i čak satna ažuriranja komponenti
  - Dostupnost - čak i ako jedan mikroservis otkaže ostatk funkcionalnosti se i dalje može koristiti
  - Modularnost se forsira ovim pristupom
  - Lako se razvija na različitim platformama
- Često se u praksi nalazi i neko hibridno rešenje