10
Obrasci za skalabilnost i arhitektonski obrasci

U svojim ranim danima, Node.js je bio prvenstveno neblokirajući veb server; njegovo izvorno ime je zapravo bilo *web.js*. Njegov autor, *Ryan Dahl*, brzo je uočio potencijal platforme i počeo da je proširuje alatkama koje omogućavaju izradu svih vrsta aplikacija za serversku stranu koje se zasnivaju na JavaScriptu i neblokirajućem modelu. Osobine platforme Node.js bile su savršene za implementiranje distribuiranih sistema, koji se sastoje od čvorova (engl. *node*) čije se operacije u mreži obavljaju uz međusobnu saradnju. Node.js je rođen za distribuiranje. Za razliku od drugih veb platformi, reč skalabilnost ulazi u rečnik autora Node.js aplikacija vrlo rano u životnom veku aplikacije, najviše zbog jednonitne prirode platforme, zbog čega nije u stanju da eksploatiše sve resurse mašine, ali često postoje i dublji razlozi. Kao što ćemo videti u ovom poglavlju, skaliranje jedne aplikacije ne znači samo obično povećavanje njenog kapaciteta, nego i bržu obradu zahteva; to je takođe ključno za postizanje visokog nivo dostupnosti i otpornosti na greške. Iznenađujuće je da to takođe može biti i način da se složenost aplikacije raspodeli na manje elemente kojima se upravlja. Skalabilnost je koncept s više stranica, njih šest, da budemo precizni, koliko ih ima i kocka— *kocka razmera* (engl. *scale cube)*.

U ovom poglavlju razmtramo sledeće teme:

- Šta je to kocka razmera
- Kako skalirirati aplikaciju izvršavanjem više instanci iste aplikacije
- Kako za skaliranje aplikacije upotrebiti raspoređivač opterećenja
- Šta je to registar servisa i kako se on može iskoristiti
- Kako iz monolitne aplikacije izvesti mikroservisnu arhitekturu
- Kako integrisati više servisa pomoću nekih jednostavnih arhitektonskih obrazaca

Uvod u skaliranje aplikacija

Pre nego što pređemo na neke praktične obrasce i primere, vredi reći nekoliko reči o razlozima za skaliranje jedne aplikacije i kako se to može postići.

Skaliranje Node.js aplikacija

Već znamo da većina se poslova koje obavlja tipična Node.js aplikacija odvija u kontekstu jedne niti. U poglavlju 1, saznali smo da to nije ograničenje, nego zapravo prednost, zato što to aplikaciji omogućava da optimizuje potrošnju resursa koji joj trebaju za obradu istovremenih zahteva, zahvaljujući neblokirajućem U/I modelu. Jednonitni način rada, kada je dobro iskorišćen za neblokirajuće U/I operacije, savršeno odgovara aplikacijama koje obrađuju umeren broj zahteva u sekundi, uglavnom nekoliko stotina u sekundi (to znatno zavisi od same aplikacije). Ako koristimo uobičajen komercijalni hardver, maksimalan kapacitet koji jedna nit može da podrži je ograničen, bez obzira na to koliko je server moćan, pa zato, ako za teško opterećenu aplikaciju želimo da koristimo Node.js, jedini način da je *skaliramo* jeste da ona radi u više procesa i na više mašina.

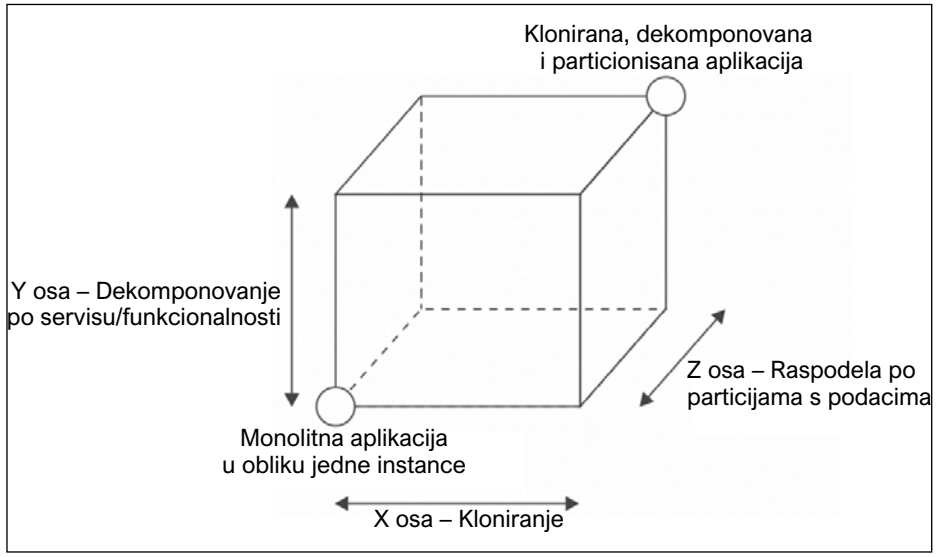
Međutim, opterećenje nije jedini razlog za skaliranje Node.js aplikacije; u stvari, pomoću istih tehnika možemo dobiti i druge poželjne osobine kao što su **raspoloživot** i **otpornost na greške**. Skalabilnost je koncept koji se odnosi i na obim i složenost aplikacije; u stvari, građenje arhitektura koje su u stanju da se automatski šire i rastu još jedan važan činilac pri projektovanju softvera. JavaScript je alatka koju treba koristiti uz veliki oprez jer neproveravanje tipa i mnogobrojne *zamke* u tom jeziku mogu biti prepreka za širenje aplikacije, ali uz disciplinu i dobar dizajn, to možemo preokrenuti u prednost. JavaScript nas često primorava da aplikaciju načinimo jednostavnom i delimo je na upravljive delove, što olakšava njeno skaliranje i distribuiranje.

Tri dimenzije skalabilnosti

Kada govorimo o skalabilnosti, prvi osnovni princip koji treba razumeti jeste **raspodela opterećenja** (engl. *load distribution*), što je *nauka* o raspoređivanju opterećenja jedne aplikacije na više procesa i mašina. Postoji više načina da se to postigne, a knjiga *The Art of Scalability* čiji su autori *Martin L. Abbott* i *Michael T. Fisher*, predlaže zgodan model za njihovo predstavljanje, nazvan **kocka razmera**. Taj model opisuje skalabilnost pomoću sledeće tri dimenzije:

- *x*-**osa**: Kloniranje
- *y*-**osa**: Dekomponovanje po servisu/funkcionalnosti
- *z*-**osa**: Raspodela po particijama s podacima

Te tri dimenzije mogu se predstaviti u obliku kocke, kao na narednoj slici:



Donji levi ugao kocke predstavlja aplikacije u kojima sve servise i funkcionalnosti obezbeđuje kôd iz jedne celine (monolitna aplikacija) koji se izvršava u jednoj instanci. To je uobičajen slučaj aplikacija koje rade pod malim opterećenjem u ranim fazama razvoja.

Najintuitivnija evolucija je da se monolitna i neskalirana aplikacija pomera udesno po *x*– osi, što je jednostavno, najčešće „jeftino" (u pogledu cene razvoja) i prilično efikasno. Princip na kojem se ta tehnika zasniva je elementaran, a to je kloniranje iste aplikacije *n* puta i dodeljivanje *1*/*n*-tog dela opterećenja svakoj instanci aplikacije na obradu.

Skaliranje duž *y*-ose znači dekomponovanje aplikacije na njene funkcionalnosti, servise ili slučajeve upotrebe. U ovom primeru, *dekomponovanje* znači izradu različitih, samostalnih aplikacija, svaka s vlastitim kodom, ponekad i s vlastitom namenskom bazom podataka, pa čak i sa vlastitim korisničkim interfejsom. Na primer, čest slučaj je razdvajanje dela aplikacije koji je odgovoran za administriranje od dela s kojim korisnik neposredno radi. Drugi primer je izmeštanje servisa koji obavljaju identifikovanje korisnika i izrada namenskog servera za identifikovanje korisnika. Način na koji će aplikacija biti podeljena po njenim funkcionalnostima zavisiće najviše od poslovnih zahteva, slučajeva upotrebe, podataka s kojima radi i mnogih drugih činilaca, kao što ćemo videti u nastavku ovog poglavlja. Zanimljivo je da je ovo dimenzija skaliranja koja najviše utiče, ne samo na arhitekturu aplikacije, nego i na način na koji se ona razvija. Kao što ćemo videti, mikroservis je reč koja zasad najviše asocira na sitno izdeljeno skaliranje po *y*-osi.

Poslednja dimenzija skaliranja je *z*-osa, gde se aplikacija deli tako da je svaka njena instanca odgovorna za rad samo s određenim delom podataka. Ta tehnika se koristi prvenstveno u bazama podataka, a naziva se još i **horizontalno partici-**

onisanje ili **cepkanje** (engl. *sharding*). U toj tehnici postoji više instanci iste aplikacije, a svaka obrađuje samo jednu od particija podataka, koji se raspodeljuju na osnovu različitih uslova. Na primer, korisnike jedne aplikacije možemo podeliti po državi u kojoj se nalaze (*particionisanje u obliku liste*) ili po početnom slovu imena (*particionisanje po opsegu*) ili tako što odluku o tome kako će korisnici biti raspoređivani po particijima prepustimo određenoj funkciji (*particinisanje na osnovu heša*). Svakoj particiji se zatim može dodeliti konkretna instanca naše aplikacije. Upotreba particionisanih podataka zahteva da svakoj operaciji s njima prethodi korak pretraživanja koji određuje koja je instanca aplikacije odgovorna za dati podatak. Kao što smo već rekli, particionisanje podataka se obično primenjuje na nivou baza podataka zato što je njegova glavna svrha rešavanje problema čiji se uzroci nalaze u obradi obimnih monolitnih skupova podataka (ograničen prostor na disku, nedovoljna memorija i mali kapacitet mreže). Primenu tih rešenja na nivou aplikacije vredi razmotriti samo za složene, distribuirane arhitekture ili za vrlo posebne slučajeve upotrebe kao što je, na primer, izrada aplikacije u kojoj su potrebna namenska rešenja za trajnost podataka, kada se koriste baze podataka koje ne podržavaju particionisanje podataka ili pri izradi aplikacija na nivou Google razmera. Zbog složenosti, skaliranje aplikacije duž *z*-ose trebalo bi razmotriti tek nakon iscrpljivanja svih ostalih mogućnosti na *x*- i *y*-osama kocke razmera.

U odeljcima koji slede, usredsredićemo se na dve najuobičajenije i najefikasnije tehnike za skaliranje Node.js aplikacija, a to su **kloniranje** i **dekomponovanje** po pojedinačnim funkcionalnostima/servisima.

Kloniranje i raspodela opterećenja

Tradicionalni, višenitni veb serveri se uglavnom skaliraju samo kada više nije moguće dopuniti resurse dodeljene određenoj mašini ili kada bi to bilo skuplje od stavljanja u upotrebu još jedne mašine. Kada pokreću više niti, tradicionalni veb serveri mogu da u punoj meri iskoriste procesnu moć servera, tako što upotrebe sve procesore na raspolaganju i svu memoriju. Međutim, u jedinom Node.js procesu to se teže postiže, pošto je platforma jednonitna i ima standardno ograničenje 1.7 GB memorije na 64-bitnim mašinama (što se može povećati pomoću specijalne opcije na komandnoj liniji `--max_old_space_size`). To znači da se Node.js aplikacije skaliraju obično znatno ranije u poređenju s tradicionalnim veb serverima, čak i u kontekstu jedne mašine, da bi mogli da iskoriste sve e resurse.

Na platformi Node.js, **vertikalno skaliranje** (dodavanje novih resursa istoj mašini) i **horizontalno skaliranje** (dodavanje novih mašina infrastrukturi) gotovo su ekvivalentni koncepti; oba zapravo podrazumevaju iste tehnike da bi se iskoristila puna moć za obradu.

Nemojte na to gledati kao na manu. Naprotiv, gotovo *prisilno* skaliranje ima povoljne efekte na druge osobine aplikacije, posebno na njenu raspoloživost i otpornost na greške. U stvari, skaliranje Node.js aplikacije kloniranjem relativno je jednostavno i često se primenjuje čak i kada nema potrebe za trošenjem dodat-