
TEHNIČKO VELEUČILIŠTE U ZAGREBU

POLITEHNIČKI SPECIJALISTIČKI DIPLOMSKI STRUČNI STUDIJ

Specijalizacija informatika

Danijel Hrček

**RAZVOJ VISOKO SKALABILNIH RJEŠENJA U
OBLAKU KORISTEĆI MICROSOFT AZURE**

DIPLOMSKI RAD br. I 735

Zagreb, prosinac, 2016.

TEHNIČKO VELEUČILIŠTE U ZAGREBU
POLITEHNIČKI SPECIJALISTIČKI DIPLOMSKI STRUČNI STUDIJ
Specijalizacija informatika

Danijel Hrček

JMBAG: 0246043627

RAZVOJ VISOKO SKALABILNIH RJEŠENJA U
OBLAKU KORISTEĆI MICROSOFT AZURE

DIPLOMSKI RAD br. I 735

Povjerenstvo:

Prof. dr. sc. Miroslav Slamić, profesor visoke škole

Nikola Majstorović, dipl. ing.

Tin Kramberger, struč. spec. ing. inf. tech., pred

Zagreb, prosinac, 2016.

Sažetak

Rad je podijeljen u 3 glavna poglavlja. U prvom poglavlju govori se općenito o nastanku interneta, povijesti, rješenjima u oblaku, *Microsoft-u*, *Azure* platformi, nedostacima i prednostima kao i njegovoj usporedbi sa *Amazon AWS* platformom. Poglavlje također sadrži opis važnijih mogućnosti koje *Azure* nudi.

Drugo poglavlje usmjereno je na izradu skalabilnih i modularnih rješenja na *Microsoft Windows Azure* platformi kao i opis glavnih komponenata koje pojedini servis čine podložnim visokim performansama. U ovom poglavlju razrađuje se glavna problematika kojom se bavi ovaj rad, kako osigurati jednake performanse sustava u oblaku bez obzira na opterećenja pod kojim se može naći.

U posljednjem poglavlju prikazat će se mjerenje opterećenja oglednog primjera skalabilnog sustava na *Microsoft Azure* platformi, zajedno sa popraćenom statistikom.

Ključne riječi: Oblak, Internet of things, Microsoft, Azure, Windows Azure, Skalabilnost, Skalabilne aplikacije

Sadržaj

1. Uvod	1
2. Teorijski okvir	2
2.1. Internet.....	2
2.2. Oblak	3
2.2.1. Prednosti računarstva u oblaku	4
2.2.2. Nedostaci računarstva u oblaku	5
2.2.3. Najčešće upotrebe računarstva u oblaku	6
2.3. Microsoft Azure Platforma	13
2.3.1. Primjeri ostalih komercijalnih računalnih oblaka	24
2.3.2. Usporedba: Microsoft Azure vs. Amazon AWS.....	25
3. Skalabilne aplikacije.....	30
3.1. Zaključavanje kod baza podataka	41
3.2. Caching.....	45
3.3. Asinkrone operacije	47
3.4. Redundancija	48
4. Izrada i testiranje skalabilne internet aplikacije	53
5. Zaključak.....	61

Dijagrami i slike

Slika 1 - Opći primjer infrastrukture	3
Slika 2 - Oblici i zahvat pruženih usluga u oblaku	7
Slika 3 - SaaS - Office internet aplikacije.....	8
Slika 4 - IaaS Primjer - Microsoft Azure	9
Slika 5 - Privatni, javni i hibridni oblik računarstva u oblaku	11
Slika 6 – Azure virtualni stroj - GitLab	14
Slika 7 - Azure Internet servisi.....	15
Slika 8 - Azure podatkovni servisi	16
Slika 9 - Azure analiza podataka.....	17
Slika 10 - Azure IoT servisi	19
Slika 11 - Azure networking	20
Slika 12 - Azure sigurnost	21
Slika 13 - Azure Developer	22
Slika 14 - Azure Intelligence	24
Slika 15 - Compute usporedba - Amazon – Azure.....	27
Slika 16 - Konačna usporedba - Amazon – Azure	29
Slika 17 – Primjer Slashdot efekta.....	30
Slika 18 – Primjer sustava za preraspodjeljivanje(engl. Load balancer)	32
Slika 19 - Azure, kreiranje nove Internet aplikacije	35
Slika 20 - Azure deployment slot.....	36
Slika 21 - Azure cjenovni rang Internet aplikacija.....	38
Slika 22 Azure, skaliranje instanci	39
Slika 23 - Azure, skaliranje internet aplikacije cpu	40
Slika 24 - SQL, repliciranje baze podataka	43
Slika 25 - Microsoft Azure status dostupnosti	49
Slika 26 - Eksponencijalna politika ponovnog pokušaja.....	50
Slika 27 - Izgled aplikacije	54
Slika 28 - Visual studio test.....	55
Slika 29 - Rezultati testa, skaliranje isključeno	56
Slika 30 - Rezultati testa, skaliranje isključeno	56

Slika 31 - Postavke skaliranja.....	57
Slika 32 - Rezultati testa, skaliranje uključeno.....	58
Slika 33 - Rezultati testa, skaliranje uključeno.....	58
Slika 34 - Skaliranje, pootvrda o preraspodjeli instanci.....	59

1. Uvod

Ideja iza upotrebe rješenja u oblaku zasniva se na principu plaćanja resursa po upotrebi. Korištenjem rješenja u oblaku, klijenti i korisnici ne kupuju infrastrukturu niti ulažu financijska sredstva u izgradnju informacijskih sustava već ju iznajmljuju od pružatelja usluga. Prednost takvog načina poslovanja jest što se poslodavci mogu usredotočiti isključivo na razvoj softverskih i ostalih rješenja bez potrebe za izgradnjom i potporom hardverskih zahtjeva.

Rješenja u oblaku pojam je koji se u zadnjim par godina često spominje, no takva rješenja dostupna su već godinama. *Facebook, Dropbox, Gmail, Skype, PayPal* samo su neki od primjera rješenja koje koristimo. Iako su prisutna već duži niz godina, u posljednje vrijeme sve se više tvrtki, manjih i većih, banaka a i drugih institucija okreće takvim rješenjima i za vlastite proizvode. Tome je pridonijela i činjenica da su se brzine interneta posljednjih godina znatno povećale. Samim time sada je više nego ikad dostupno obavljanje raznih zadataka sa udaljenih lokacija i preko interneta.

Cilj je ovog diplomskog rada objasniti upotrebu skalabilnih i modularnih rješenja u oblaku. Uz ovaj rad prikazat će se i kako razviti jedno od takvih rješenja te korištenjem Microsoft Windows Azure platforme, osigurati skaliranje takvog sustava u slučaju povećanja broja korisnika ili opterećenja sa kojim se sustav mora nositi.

2. Teorijski okvir

2.1. Internet

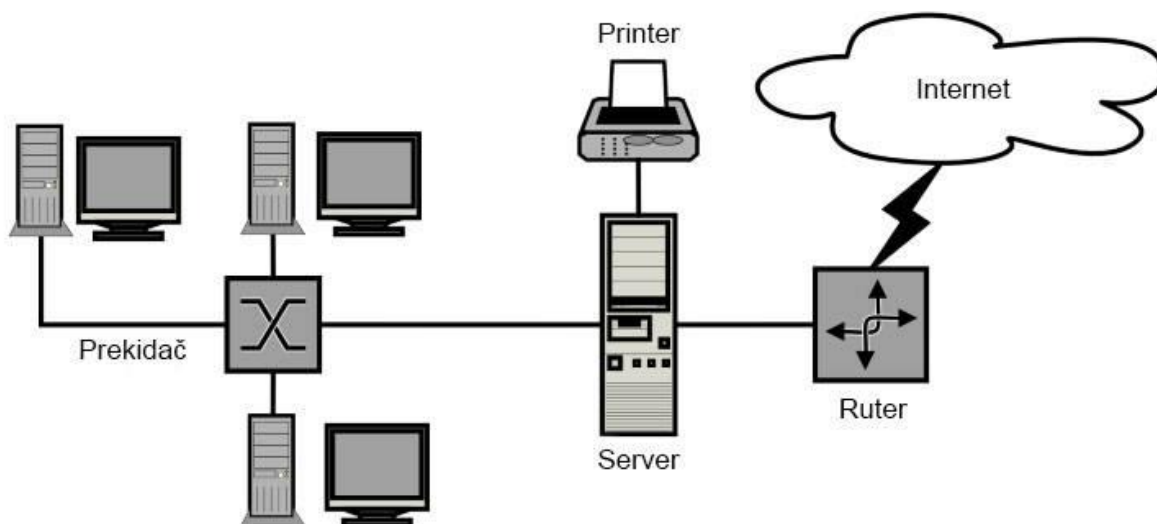
Internet predstavlja svjetski sustav međusobno povezanih računalnih mreža u kojoj korisnici sa bilo kojeg računala, ako im je to dozvoljeno, mogu dohvatiti podatke sa drugog računala. Za prijenos podataka te povezivanje korisnika preko Interneta koristi se TCP/IP protokol. Internet povezuje privatne, poslovne i javne mreže u jednu zajedničku mrežu. Ne nalazi se u ničijem vlasništvu i možemo ga na neki način gledati kao samoodrživi sustav kojemu ima pristup preko milijun korisnika diljem svijeta. Fizički gledano, Internet koristi samo dio današnjih postojećih telekomunikacijskih resursa. Radi na principu zahtjev – odgovor.

Računala koja pružaju servise i informacije na internetu nazivaju se serveri ili poslužitelji dok se računala koja zahtijevaju informacije od poslužitelja nazivaju klijenti. Poslužitelji su računala koja mogu obrađivati veliku količinu informacija i istovremeno posluživati više tisuća korisnika dok su klijenti većinom osobna računala, mobiteli, tableti ili slični uređaji koji traže određene informacije od servisa. Svaki korisnik na Internetu identificira se pomoću jedinstvene IP adrese. IP adresa označava lokaciju i identitet računala. Internet nema središnje upravljanje u svojoj tehničkoj implementaciji kao ni u načinu pristupa i korištenja. Svaka od sastavnih mreža postavlja svoja vlastita pravila i smjernice za njegov pristup. Ovo osigurava da nitko nema Internet pod svojom kontrolom te je samim time omogućena i sloboda izražavanja tj. nesmetana razmjena informacija bez obzira na njihov sadržaj. Prijenos paketa na internetu odvija se preko raznih mrežnih uređaja od kojih su najčešći usmjernici. Promet se odvija u ovisnosti o IP adresama, koje služe kao smjernice za usmjeravanje paketa preko mreže.

Za prijenos podataka preko Interneta najviše se koristi HTTP protokol, koji je zapravo opisni jezik za prijenos informacija. Internet je na neki način kolekcija međusobno povezanih dokumenata (Internet stranica). Te stranice međusobno su povezane pomoću hiperlink-ova i URL-ova.

2.2. Oblak

Što je to točno oblak, gdje se nalazi i jesmo li trenutno u oblaku neka su od pitanja koja se često protežu kroz IT zajednicu. Pojam računarstva i računala u oblaku danas se može naći gotovo svagdje. Ukratko, oblak se može jednostavno objasniti kao sredstvo spremanja i pristupanja podacima i programima preko interneta umjesto lokalnog mjesta za pohranu kao što je to tvrdi disk. Mogli bismo reći da je oblak na neki način apstrakcija za Internet i arhitekturu koja leži iza toga i na kojoj se vrte rješenja u oblaku. Sinonim se može povući i sa starijih prezentacija i dijagrama toka gdje se na pojavu Interneta obično crtao jednostavno oblak vezan na neki uređaj.



Slika 1 - Opći primjer infrastrukture

U računarstvo u oblaku ne spada ništa što se odnosi na pohranu podataka lokalno, kada podaci koji su nam potrebni su također i fizički blizu. Pristup takvim podacima poprilično je brz i jednostavan, ali samo za osobu na tom lokalnom računalu ili lokalnoj mreži. IT industrija funkcionirala je na ovakav način godinama. Spremanje podataka na dedican server ili spremanje na uredsku mrežu također se ne smatra kao korištenje oblaka. Da bi se određena infrastruktura smatrala kao rješenjem ili računarstvom u oblaku potrebna je veza na Internet i pristup tim podacima preko

interneta. Kranji rezultat trebao bi biti jednak, pristup podacima bilo gdje, bilo kada. Međutim ponekad se ne može tako jednostavno odrediti što pripada kamo. To se događa iz razloga što je računarstvo u oblaku danas dio gotovo svega na računalima. Jednostavan primjer bio bi lokalna instalacija *Microsoft Office 365*¹ softvera koja na neki način objedinjuje računarstvo u oblaku koristeći *Microsoft One Drive*².

2.2.1. Prednosti računarstva u oblaku

Računala u oblacima nose sa sobom mnoge prednosti. Jedna od glavnih prednosti bila bi upravo fleksibilnost. Rješenja bazirana na računalima u oblaku idealna su za poslove gdje se zahtjevi softvera neprekidno i u hodu mijenjaju. Ukoliko se potrebe rješenja povećaju, platformu je vrlo lako skalirati sukladno novim potrebama. Isto vrijedi i ako trenutni resursi premašuju trenutne zahtjeve, ista se može skalirati na manje i tako uštediti. Ovakva vrsta fleksibilnosti ono je što razdvaja pružatelje rješenja u oblaku naspram konkurencije. Sigurnost je također prednost. Izgubljena prijenosna računala mogu biti problemi vrijedni milijune eura. Točnije, gubitak osjetljivih podataka koje poslovna računala mogu sadržavati. Prema nekim istraživanjima iz 2010 godine, tvrtke u prosjeku izgube 263 prijenosna računala godišnje. Kod spremanja podataka u oblak, pristupati im se može bilo kada od bilo gdje, što ujedno i znači da u slučaju bilo kakvog rizika podatke je moguće obrisati kako ne bi upali u pogrešne ruke. Serveri u oblaku nalaze se izvan dohvata korisnika i nema potrebe za izvršavanjem regularnih nadogradnji softvera ili operativnih sustava. Pružatelj usluga obavlja takve nadogradnje samostalno ostavljajući korisnike da se jednostavno usredotoče na krajnji proizvod. Takve automatske nadogradnje softvera još su jedna od prednosti rješenja u oblaku. Bez oblaka, život danas izgledao bi podosta drugačije. Takva rješenja jednostavno su postala dio našeg svakodnevnog života i koristimo ih bez da smo toga i svjesni. Kada bi nestali *Facebook*, *Twitter*, *Gmail*, *Spotify* ili srodni popularni servisi, životi pojedinaca okrenuli bi se naglavačke.

¹ Microsoft Office 365 - <https://products.office.com/en/office-365-home>

² Microsoft One Drive - <https://onedrive.live.com/about/en-us/>

2.2.2. Nedostaci računarstva u oblaku

Računarstvo u oblaku nedvojbeno je doprinijelo mnogim tvrtkama u poslovanju, smanjujući im troškove i omogućavajući im da se posvete isključivo razvoju njihove primarne djelatnosti, umjesto rješavajući probleme IT infrastrukture potrebne za njihov rad. Međutim, uz sve prednosti koje pruža, ima i svojih negativnih strana. Jedna od najčešćih i najvećih bila bi upravo njegova nedostupnost. Niti jedan od pružatelja rješenja u oblaku, čak ni onih najvećih ne usudi se tvrditi da je imun na prekide usluga. Sva takva rješenja zavise od Internet veze, što ujedno i znači da i svaki klijentski pristup ovisi upravo o tome. Kao i bilo koji dio hardvera, čak i same platforme mogu postati nefunkcionalne iz jednog od mnogih razloga. Zanimljiv incident dogodio se i 2014 godine kada je *DropBox*³ imao popriličan broj nedostupnih sati kroz puna dva dana. Jedan od načina kako se zaštititi od toga jest tražiti od pružatelja usluge SLA (engl. Service level agreement) koji garantira dostupnost usluge sukladno potrebama aplikacije. Sigurnost i privatnost još su jedan od nedostataka. Gdje god da je uključena upotreba osjetljivih podataka, posebna pozornost mora se posvetiti i upravljanju osjetljivim podacima. Od pružatelja usluge očekuje se da upravlja i čuva osnovni hardver i infrastrukturu rješenja, međutim pristup resursima izvana odgovornost je klijenta. U današnje vrijeme hakerskih napada ima podosta, a nedavno su se na meti našle i slike slavnih osoba i milioni korisničkih podataka. Kako bi se izbjegli problemi oko sigurnosti poželjno bi bilo uvijek znati tko treba imati pristup određenom resursu te proširiti sigurnost sve do krajnjeg uređaja za pristup. Neki od pružatelja usluga danas pružaju i preventivnu analitiku protiv bilo kakvih napada kako bi obavijestili administratore o mogućnostima napada. Ovisnost o pojedinoj platformi može se također pokazati kao nedostatak. Takva izravna ovisnost može doći do izražaja između sustava dvaju pružatelja kada klijent odluči prijeći sa jednog na drugog. Ne samo da takav prijelaz može biti izrazito kompleksan i skupocjen radi konfiguracije rješenja zahtjevima novog pružatelja, već može izložiti osjetljive podatke i sigurnost dodatnim ranjivostima. Detaljno poznavanje usluga koje vlasnici infrastruktura u oblaku prodaju ključno je za izbjegavanje navedenog problema. Nadalje, kada se radi o kratkoročnim projektima na nekoj nižoj razini, korištenje rješenja u oblaku može se pokazati skupljim izborom. Unatoč tome što može doprinijeti smanjenju ljudi koje je

³ DropBox - <https://www.dropbox.com/>

potrebno uključiti u razvoj. U ovom slučaju od velikog značaja mogu biti online kalkulatori izračuna cijene zakupa pojedine platforme. Dodatno kako bi se smanjila cijena korištenja rješenja u oblaku potrebno je voditi računa o skaliranju usluga po potrebi kao i zaustavljanju instanci kada nisu potrebne.

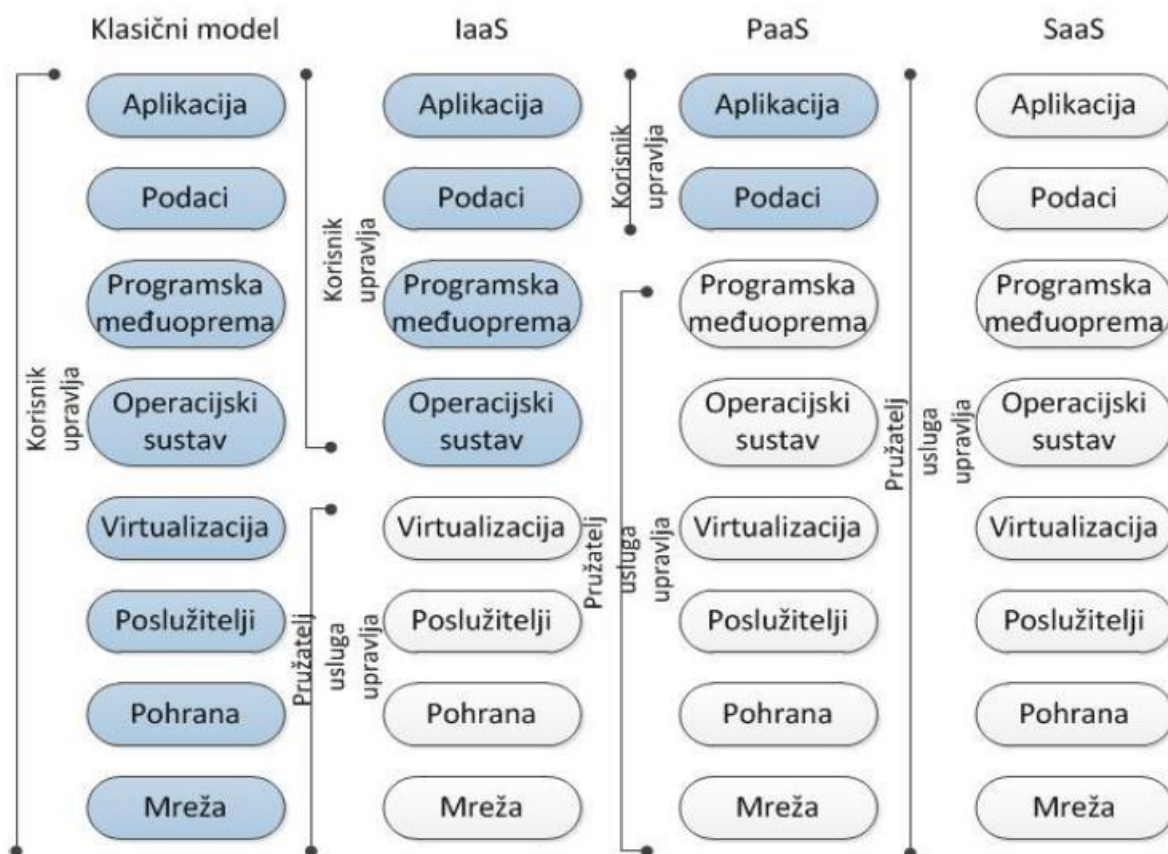
Unatoč nedostacima računarstva u oblaku, takva okolina ima neizmjerne prednosti za manje i veće poslovne modele. Uz konstantan rast platformi cijene će nastaviti padati a standardi pouzdanosti i sigurnosti zasigurno će se popraviti.

2.2.3. Najčešće upotrebe računarstva u oblaku

Za razvoj računala u oblaku zaslužna je upravo njihova kompetentnost kroz smanjenje cijena i optimizaciju upotrebe resursa. Postoji mnogo situacija gdje se ona mogu primijeniti za postizanje poslovnih ciljeva.

Neki od najčešćih scenarija korištenja računala u oblaku odnose se na:

- ▶ Softver kao servis(engl. Software as a service, SaaS)
- ▶ Infrastrukturu kao servis(engl. Infrastructure as a service, IaaS)
- ▶ Platformu kao servis(engl. Platform as a service, PaaS)

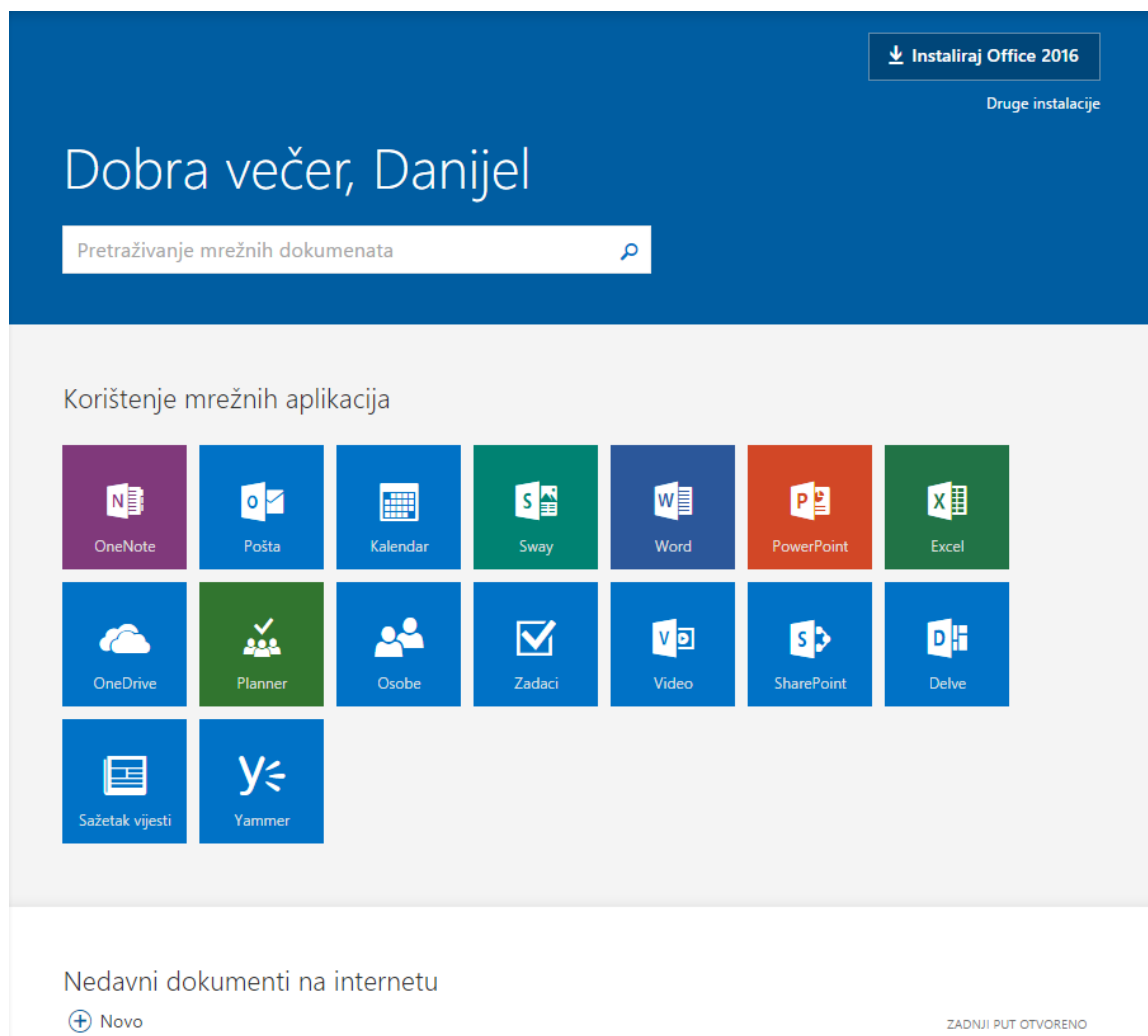


Slika 2 - Oblici i zahvat pruženih usluga u oblaku

Pojam softvera kao servisa odnosi se na aplikacije koje se vrte na udaljenim računalima koji su u vlasništvu i upravljana pružateljem usluga na koja se spajaju korisnici interneta tj. klijenti, najčešće preko Internet preglednika. Prednost takvog načina korištenja jest što su aplikacije i podaci dostupni sa bilo kojeg računala na kojem je omogućen pristup Internetu. Nema gubitka podataka ukoliko dođe do kvara klijentskog računala, pošto se svi podaci nalaze u oblaku. Ovakav način korištenja također uklanja trošak nabave hardvera, održavanja, licenciranja i podrške. Servise je moguće skalirati po potrebi, zahtjevima ili opterećenju. Primjer softvera kao servisa bile bi *Office Internet Aplikacije* u vlasništvu *Microsoft-a*. Ovi servisi dostupni su sa bilo kojeg mjesta upravo iz Internet preglednika. Aplikacije omogućavaju stvaranje datoteka direktno na internetu, koje se tada spremaju na *OneDrive*⁴ platformu za

⁴ Microsoft OneDrive - <https://onedrive.live.com/>

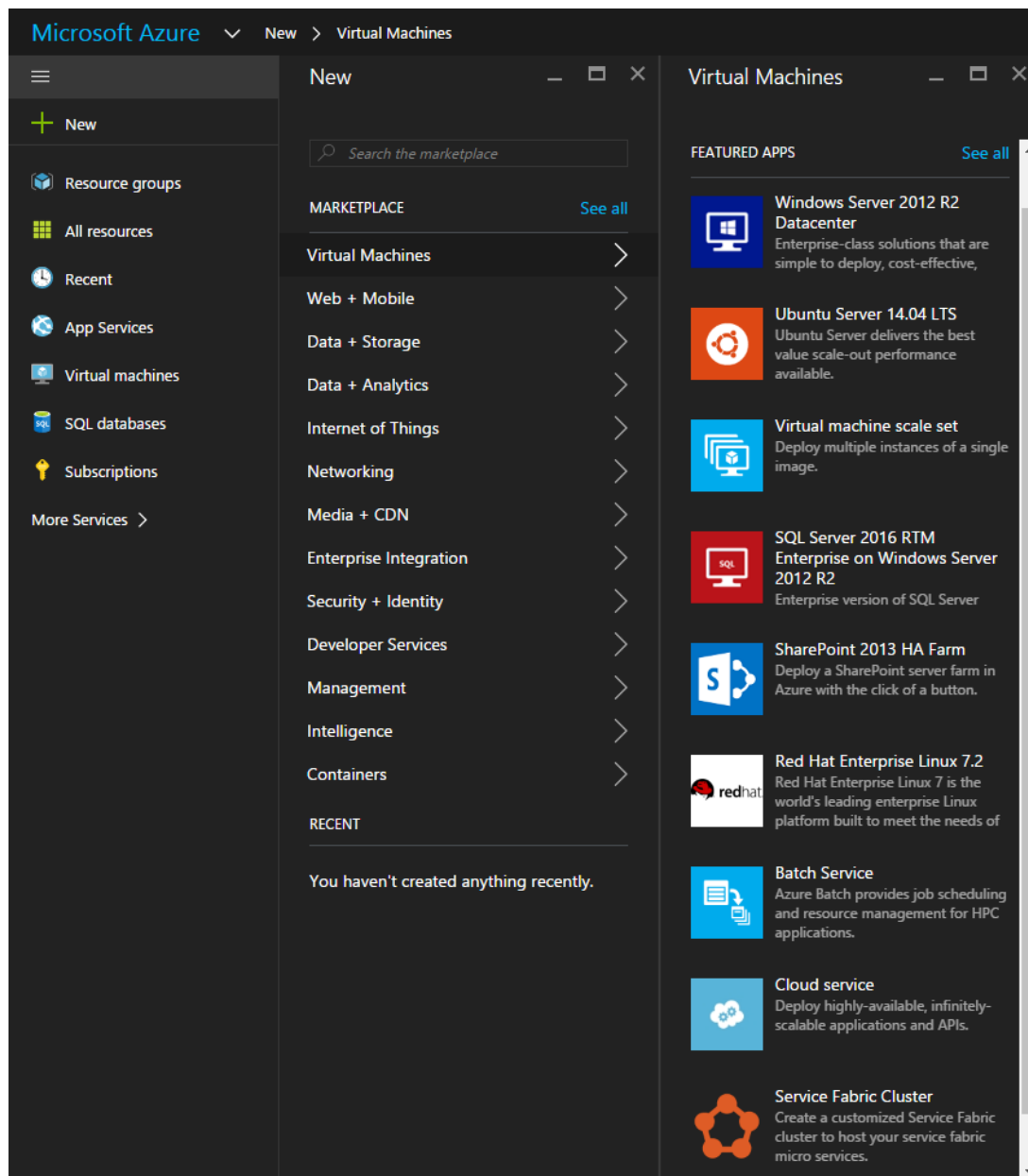
pohranu podataka. Datoteke pohranjene u oblaku mogu se izmijeniti i dijeliti sa drugim osobama u stvarnom vremenu što omogućuje bolju suradnju.



Slika 3 - SaaS - Office internet aplikacije

Korištenje infrastrukture kao servisa odnosi se na korištenje već postojeće infrastrukture na sistemu plaćanja po upotrebi. Ovo je očit izbor za mnoge tvrtke koje žele smanjiti troškove ili jednostavno uštedjeti na ulaganju nabave, održavanja ili upravljanja IT infrastrukturom. Infrastrukturu je također vrlo lako skalirati, resursi se mogu preraspodijeliti na zahtjev. IaaS je idealan za provođenje zahtjeva koji su privremeni, eksperimentalni ili se mogu neočekivano promijeniti. Moguće je i administracija administrativnih zadataka, dinamičko skaliranje, virtualizacija desktopa

i mnoštvo drugih zadataka. Primjer upotrebe infrastrukture kao servisa bio bi upravo *Microsoft Azure*, *Amazon AWS* ili slični.



Slika 4 - IaaS Primjer - Microsoft Azure

Platforma kao servis model je računarstva u oblaku koji sa sobom donosi sve što je potrebno za potporu kompletnog životnih ciklusa razvoja i pružanja Internet aplikacija, također bez troškova i kompleksnosti kupovanja i upravljanja potrebnog hardvera, softvera i usluge pružanja softvera. Kod PaaS modela, pružatelj rješenja u

oblaku kao uslugu dostavlja i hardver i alate softvera potrebne za razvoj aplikacija. PaaS model ne zamjenjuje cijelu infrastrukturu unutar neke organizacije ili tvrtke. Primjer bi bio razvoj REST(engl. Representational State Transfer) Internet servisa. Kako bi drugi inženjeri, koji recimo rade na razvoju klijenta za korištenje REST servisa mogli konzumirati resurse na servisu, potrebno je nabaviti hardver, operative sustave, među sloj kao što su npr. baze podataka ili neki drugi načini pohrane podataka, Internet server, osigurati nove resurse pravilnim pristupom i sigurnošću. Ovakav pristup može produjiti razvoj kao i pripremu pred razvoj. Pružatelj PaaS usluge brine se za sve prethodno navedene stvari, ostavljajući kupcu usluge mogućnost da se usredotoči isključivo na razvoj rješenja. Svrha većine PaaS platformi upravo je podrška za razvoj softvera. Na primjer, razvojnim inženjerima ovakav pristup omogućava često mijenjanje ili nadogradnje mogućnosti operativnih sustava kao i suradnju na projektima. Pristup se najčešće obavlja kroz Internet preglednik a usluga se naplaćuje na osnovu korištenja, tj. konekcija i pristupa. Naravno treba voditi računa i o neovlaštenim pokušajima pristupa.

Jedan od drugačijih pristupa odnosi se na korištenje rješenja u oblaku na jedan od sljedećih načina:

- ▶ Javni oblak(engl. Public cloud)
- ▶ Privatni oblak(engl. Private cloud)
- ▶ Hibridni oblak(engl. Hybrid cloud)



Slika 5 - Privatni, javni i hibridni oblik računarstva u oblaku

Javni oblak bazira se na standardima računarstva u oblaku kada pružatelj takve usluge resurse, aplikacije i skladištenje podataka čini javno dostupnim preko Interneta. Nalaze se u vlasništvu i upravljani su od strane pružatelja usluga koji nudi brz pristup preko Interneta. Ovakvi resursi mogu biti besplatni ili se naplaćuju po upotrebi. Neki od primjera javnih oblaka uključuju *Windows Azure* Servisnu platformu, *Amazon Elastic Compute* oblak, *Blue cloud* i mnoge druge.

Pojam privatnog oblaka odnosi se na infrastrukturu kada njome upravlja samo jedna organizacija, nebitno o tome da li se njome upravlja interno ili sa strane neke treće organizacije kao ni da li je podignuta interno ili negdje „vani“. Organizacija posjeduje infrastrukturu kao i ima kontrolu nad izvođenjem aplikacija na istoj. Privatni oblak ima slične prednosti kao i javni, uključujući skalabilnost, vlastite servise ali sve to kroz vlastitu arhitekturu. Glavna razlika nalazi se u tome što je privatni oblak namijenjen samo jednoj organizaciji dok je javni namijenjen višestrukim organizacijama. Ovo dolazi kao zgodno rješenje kada se zahtjeva direktna kontrola nad infrastrukturom na kojoj se radi ili razvija. Potrebno je najčešće kada se radi na aplikacijama čija je dostupnost ključna ili onima koje rade sa osjetljivim podacima gdje

je potrebna visoka razina sigurnosti. Nedostatak privatnog oblaka nalazi se u tome što je za njegovo održavanje zadužena upravo organizacija koja ga koristi.

Hibridni oblak jest oblik računarstva kada se objedini i privatni oblak i servisi javnog oblaka od nekog vanjskog pružatelja usluga, te nastane mix dviju različitih platformi i infrastruktura. Ovakav oblik poslovanja daje veću fleksibilnost oko upravljanja pohranom podacima i njihovom sigurnošću. Na primjer, ovakav oblik omogućava organizacijama da drže kritične podatke i aplikacije u tradicionalnom podatkovnom centru u privatnom oblaku a ujedno omogućava i korištenje vanjskih resursa kao što su SaaS i IaaS za skalabilne vanjske resurse po potrebi. Negativne strane ovakvog pristupa jesu što hibridni oblaci mogu predstavljati malo veći tehnički izazov. Pošto privatni oblak mora komunicirati sa vanjskim pružateljima, hibridni oblak zahtjeva kompatibilnost sa nekom vrstom servisa, na primjer REST.

Možda najbolji primjer upotrebe računarstva u oblaku jest korištenje testne i razvojne okoline u oblaku. Pristup razvoju na ovakav način omogućava uštedu novaca, vremena, ljudi i fizičke opreme koju je potrebno pripremiti za razvoj nekog rješenja. Dobro je napomenuti da nije potrebno ni konfigurirati ni vršiti bilo kakve konfiguracije na platformama u oblaku. Sa računarstvom u oblaku, samo par koraka dijeli organizacije od razvojnih okolina skrojenih njihovim potrebama.

Oblak također nudi mogućnosti spremanja velikih količina podataka. Spremati, pristupati i dohvaćati ih se onda može sa bilo kojeg mjesta koje ima pristup internetu. Sučelja ovakvih web servisa većinom su poprilično jednostavna. U bilo koje vrijeme i na bilo kojem mjestu, organizacije mogu uživati visoku dostupnost, brzinu, skalabilnost i sigurnost njihove okoline. U ovakvom scenariju organizacije plaćaju jedino za količinu podataka koje zapravo zauzimaju. Primjer ovakvog servisa bio bi *Azure Blob Storage*.⁵

Arhiviranje i stvaranje kopija podataka uvijek je bilo operacija koja je zauzimala podosta vremena. Uključivala je i održavanje diskova, njihovo ručno skupljanje i njihov premještanje na već predodređeno prosto za pohranu. Ovakav način pohrane nije bio imun na sve probleme koji su se mogli dogoditi. Na primjer nedostatak medija za pohranu podataka, vrijeme potrebno da se kopije podataka učitaju nazad, što obično traje neko vrijeme i postoji sklonost pogreškama uzrokovanim ljudskim faktorom.

⁵ Azure Blob storage - <https://azure.microsoft.com/en-us/documentation/articles/storage-dotnet-how-to-use-blobs/>

Pohrana podataka u oblaku jedan je od čestog korištenja rješenja u oblaku. Može se podesiti na način da se podaci otpreme preko žice sa sigurnošću da problem neće predstavljati sigurnost, dostupnost kao ni kapacitet.

2.3. Microsoft Azure Platforma

Microsoft Azure i njoj slične platforme često izazivaju različite reakcije i mišljenja oko njezinog korištenja i kako može pomoći inženjerima u razvoju. Podijeljeni stavovi javljaju se često radi manjka informiranosti. Čak i danas, mnogi studenti i kolege na spomen servisa u oblaku nisu upoznati sa svim prednostima koje ono pruža. U ovom odlomku objasniti će se što je to *Microsoft Azure*, zašto je tu, što se može napraviti pomoću njega, i zašto bi svaka tvrtka trebala uzeti u obzir korištenje servisa u oblaku.

Microsoft Windows Azure jest platforma podignuta u podatkovnim centrima u vlasništvu Microsoft-a. Lansirana je 2010 godine. Od početka lansiranja pa do kraja 2014 godine, *Azure* je uvelike narastao te 2013 godine donio *Microsoft-u* prihod u iznosu od oko 1 bilijun dolara. Danas je taj iznos zasigurno veći.

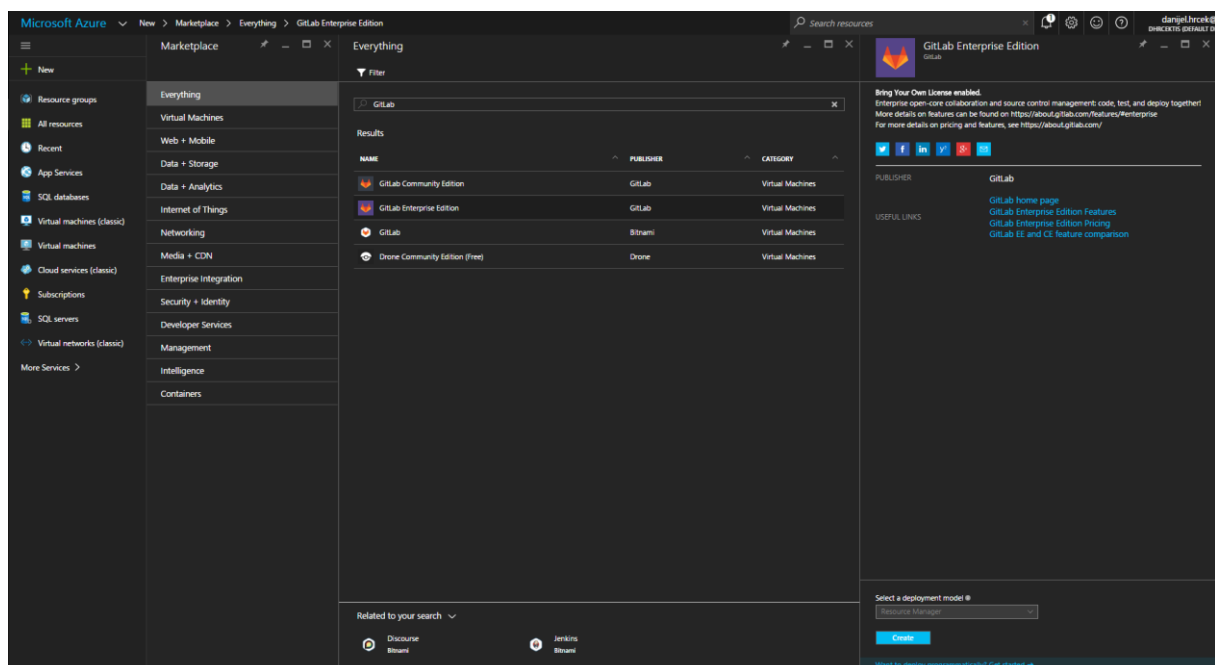
Azure pruža operativne sustave kao i set razvojnih servisa koji mogu biti korišteni zasebno ili zajedno. Korisnici mogu odabrati neki od servisa koji je njima potreban za razvoj i skaliranje novih ili već postojećih aplikacija u javnom oblaku. *Azure* se javno smatra i kao *Platform as a Service* i *Infrastructure as a Service*. Otvorena arhitektura *Microsoft Azure*-a omogućava inženjerima da razvijaju Internet aplikacije, aplikacije koje se vrte na više povezanih uređaja, računala, servere i hibridna rješenja. Važno je napomenuti da *Azure* ne podržava samo .NET rješenja već je moguće pokrenuti gotovo bilo koju platformu.

U vrijeme pisanja ovog rada, neki od važnijih *Azure* servisa podijeljeni su na:

- ▶ *Virtual Machines*
- ▶ *Web + Mobile*
- ▶ *Data + Storage*
- ▶ *Data + Analytics*
- ▶ *Internet of Things*

- *Networking*
- *Security + Identity*
- *Developer services*
- *Intelligence*

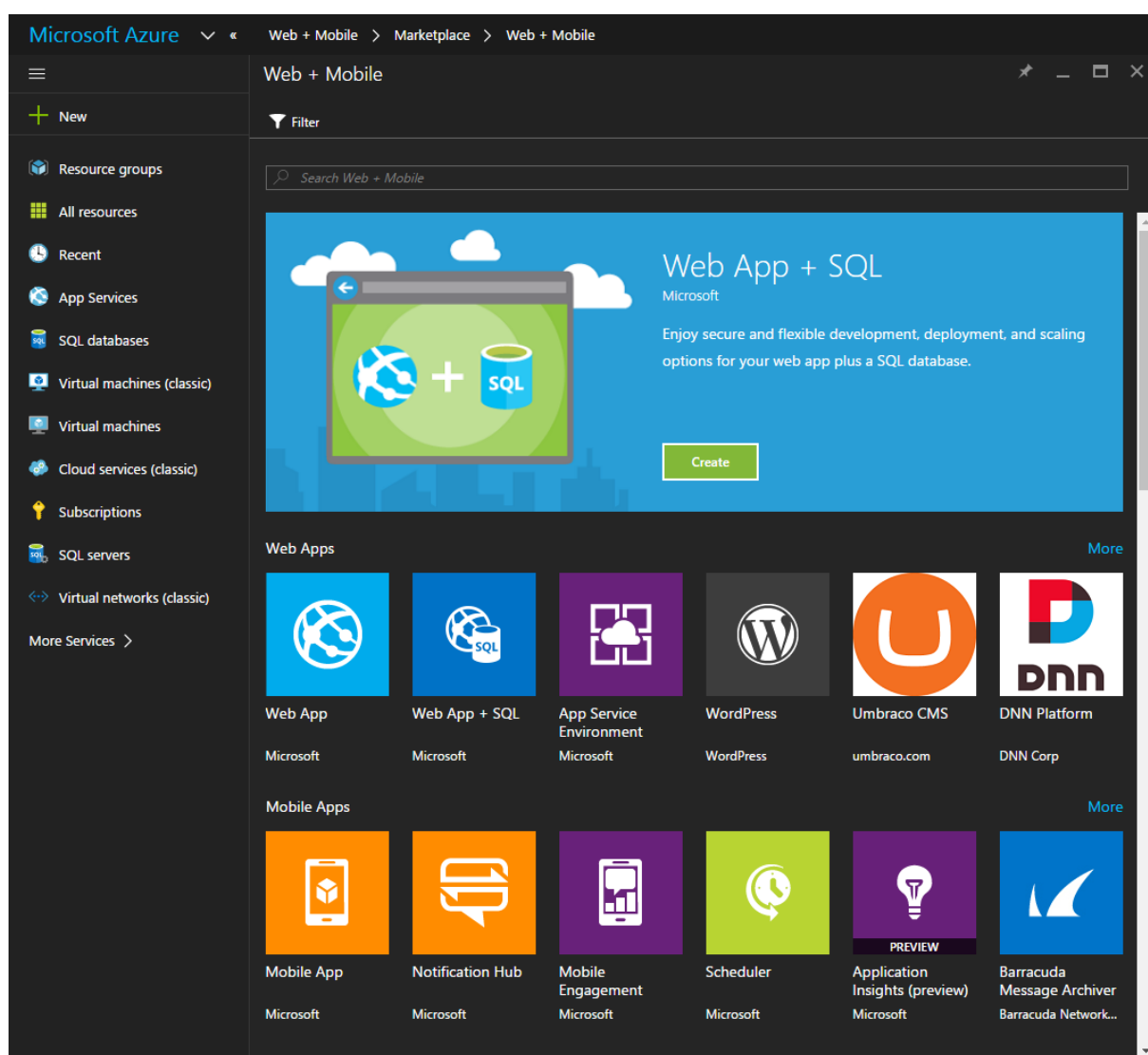
Virtual machines servisi omogućavaju zakup novog virtualnog stroja sa operativnim sustavom. Tako se na primjer može zakupiti Windows Server 2012(*datacenter*), Ubuntu server 14.04, SharePoint 2013 Ha Farm i mnogi drugi. Primjene virtualnog stroja u oblaku doslovno nemaju ograničenja. Razlog tome jest upravo to što je virtualni stroj računalo sa svojim operativnim sustavom na kojem se može pokrenuti i konfigurirati gotovo sve. Na virtualni stroj može se gledati isto kao i na osobno računalo sa razlikom da je smješteno u oblaku. Važno je napomenuti i kako *Microsoft Azure* posjeduje trgovinu, gdje je moguće zakupiti virtualni stroj sa već unaprijed podešenim i instaliranim softverom bilo za Linux ili Windows server virtualne strojeve. Primjer takvog zakupa bilo bi npr. pokretanje vlastitog sustava za suradnju i razvoj Gitlab⁶. Pokretanje tog servisa poprilično je jednostavno. Sve što je potrebno jest pronaći i odabrati u online trgovini verziju softvera koju želimo pokrenuti i *Microsoft Azure* će kreirati i zakupiti resurse za taj virtualni stroj.



Slika 6 – Azure virtualni stroj - GitLab

⁶ GitLab - <https://about.gitlab.com/>

Web + Mobile servisi omogućavaju stvaranje nove web aplikacije, fleksibilne i skalabilne, funkcijske aplikacije koje mogu biti jednostavne kao i pokretanje jednog zadatka. Servis za mobilne aplikacije, REST servise, *Notification Hub* i mnoge druge. Najčešći primjer upotrebe ovih servisa odnosi se upravo na pokretanje Internet aplikacija ili stranica. Tako je uz skalabilne i fleksibilne Internet aplikacije moguće odabrati i predefinirane *WordPress*⁷ ili druge sustave za upravljanje sadržajem(engl. CMS). Važno je napomenuti da ovi servisi omogućuju i stvaranje aplikacija nezavisnih o Microsoft platformama. Primjerice, moguće je pokrenuti i *Django*⁸ Internet aplikacije kao i mnoge druge.

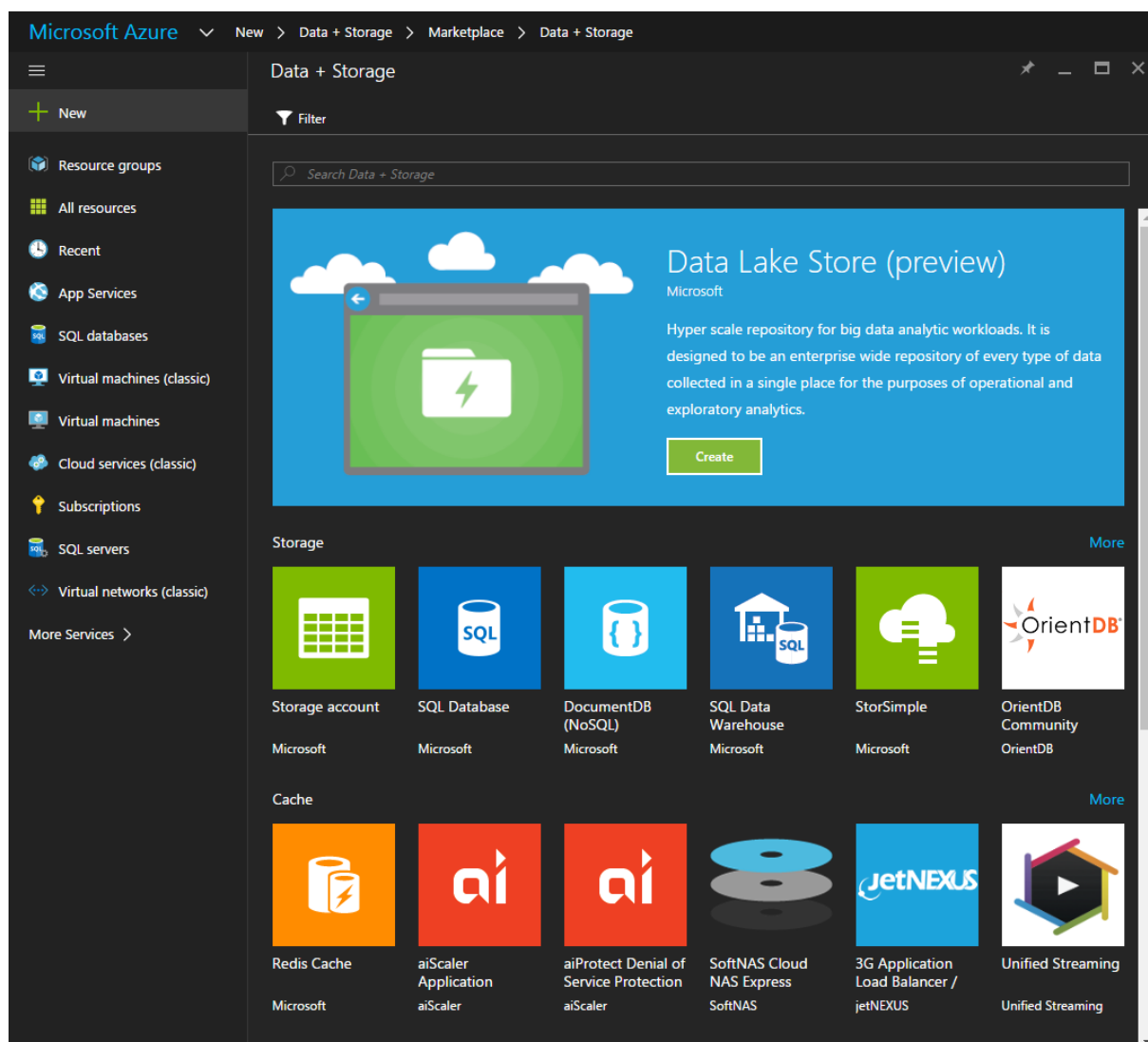


Slika 7 - Azure Internet servisi

⁷ WordPress - <https://wordpress.com/>

⁸ Django - <https://www.djangoproject.com/>

Data + Storage servisi omogućavaju kreiranje nove SQL baze podataka, skladišta podataka (engl. SQL Data Warehouse), *Azure Document* baze, računa za spremanje datoteka u binarnom obliku, Redis⁹ baze i mnoge druge. Ovisno o poslovnim potrebama i zahtjevima aplikacije ili servisa koji se razvija, ovi servisi nude širok spektar mogućnosti za gotovo sve zahtjeve.

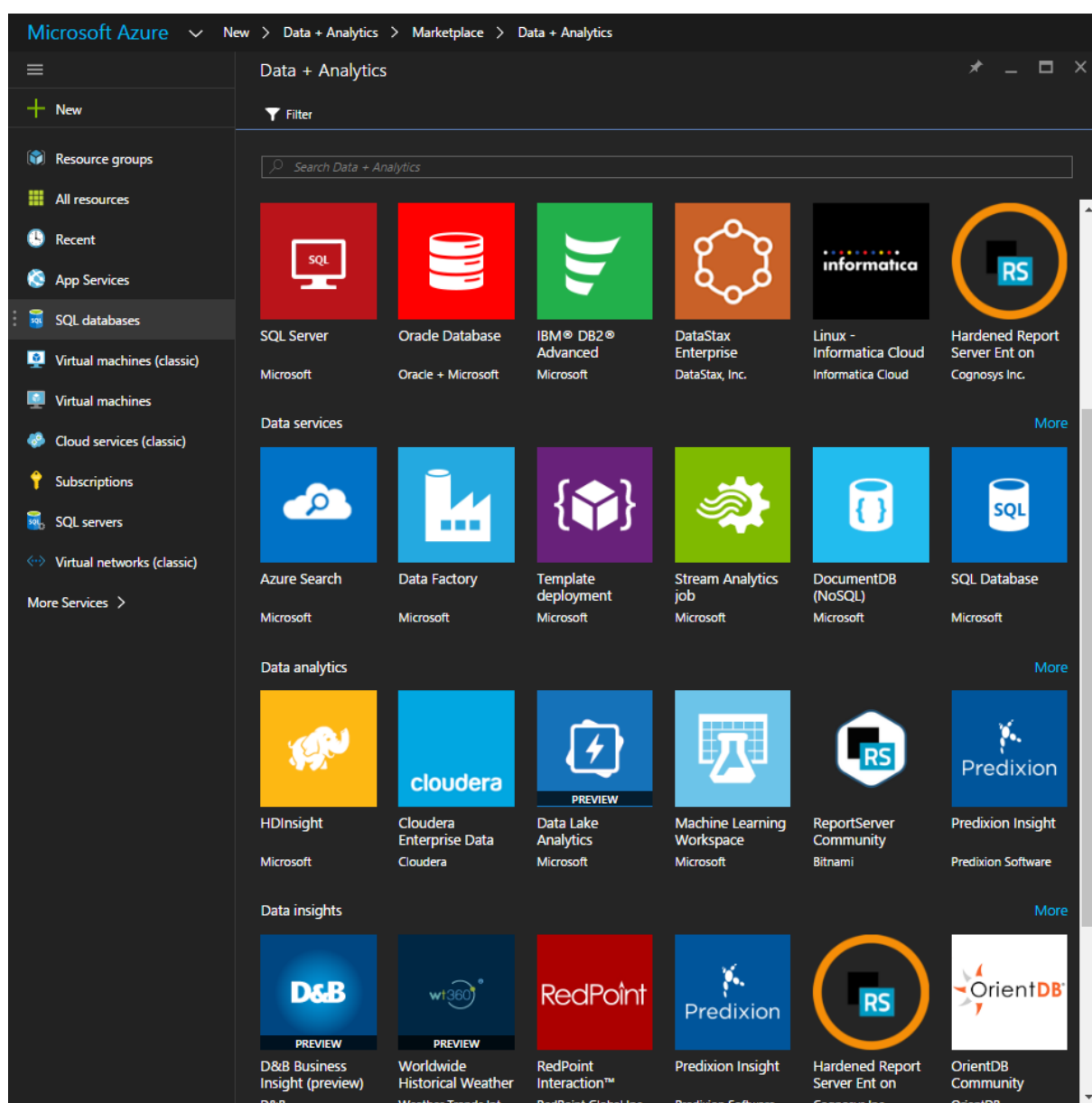


Slika 8 - Azure podatkovni servisi

Data + Analytics servisi omogućavaju stvaranje Power BI potpuno interaktivnog rješenja za vizualizaciju podataka u aplikacijama. Kognitivne servise za složene i

⁹ Redis - <http://redis.io/>

moćne algoritme, podatkovne kataloge, servise za umjetnu inteligenciju i mnoge druge. Ukratko, predviđeni su za *Big data*¹⁰ analitiku. Njihova namjena odnosi se na pružanje boljeg iskustva i više informacija analizirajući ogromne količine podataka u stvarnom vremenu. Pomoću ovih servisa moguće je dobiti uvid koji je potreban da bi se donijele pravovremene odluke vezane uz poboljšanje angažiranosti korisnika, povećanje prihoda ili smanjenje troškova.

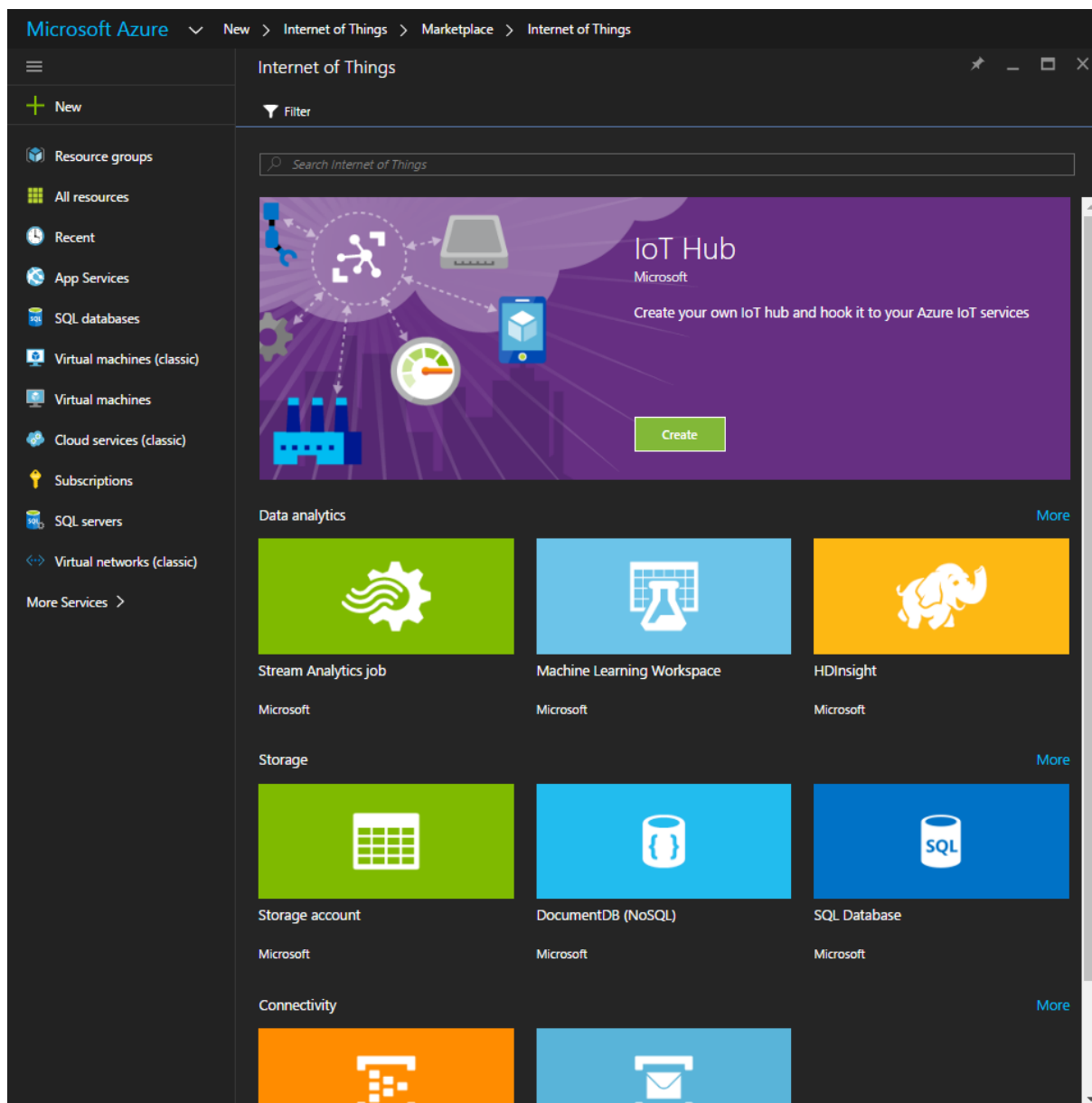


Slika 9 - Azure analiza podataka

¹⁰ Big data - https://en.wikipedia.org/wiki/Big_data

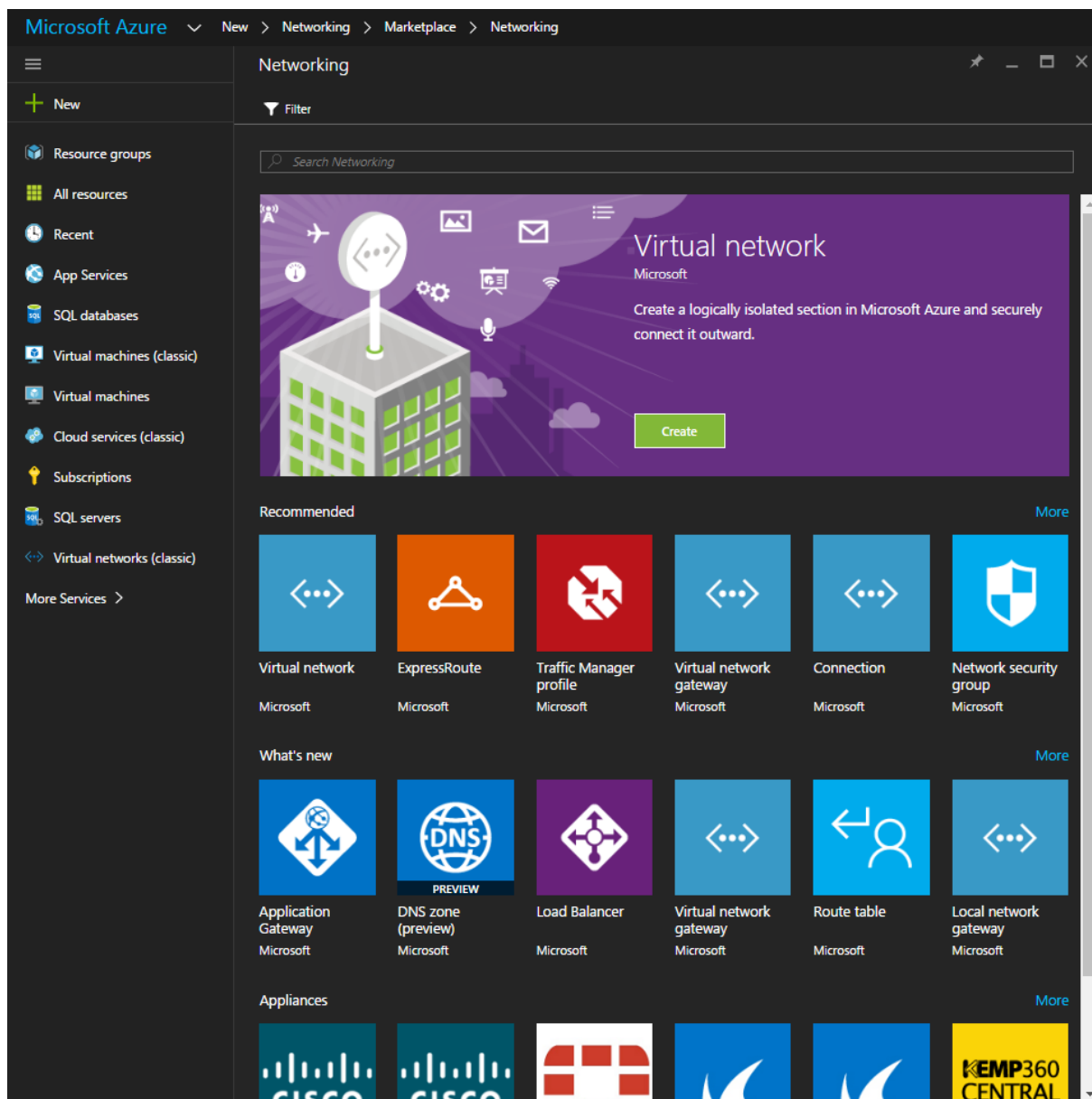
Internet of Things relativno je novi pojam, a odnosi se na pojam kada predmeti koje koristimo u svakodnevnom životu imaju Internet vezu koja im omogućava da šalju i primaju podatke u oblaku. Ovaj Azure resurs omogućuje stvaranje IoT servisa. Jedan od primjera korištenja takvih servisa najavljen je u gradu Hanover na *Messe Trade Show-u*. *Microsoft* u suradnji sa tvrtkom *Rolls-Royce* radit će na podršci za nove napredne inteligentne motore za avionske kompanije. *Rolls-Royce* će integrirati *Microsoft Azure Internet of things* servise zajedno sa *Cortana* paketom za umjetnu inteligenciju kako bi prikupili informacije o letu, potrošnji goriva i planiranju održavanja.¹¹ Zanimljiv primjer potencijalnog IoT predmeta postoji i u Hrvatskoj. Poduzetnik Ivan Mrvoš proizveo je solarnu klupu koja sadrži senzore za kvalitetu zraka, može služiti kao punjač za mobitele a sadrži i *hotspot* za Internet. Klupa je u potpunosti autonomna a u budućnosti mogla bi se iskoristiti i za analize podataka o korištenju, promjenama u njezinoj okolini ili slično.

¹¹ Christina Mercer, Computer World UK - <http://www.computerworlduk.com/galleries/cloud-computing/internet-of-things-best-business-enterprise-offerings-3626973/> (15.08.2016)



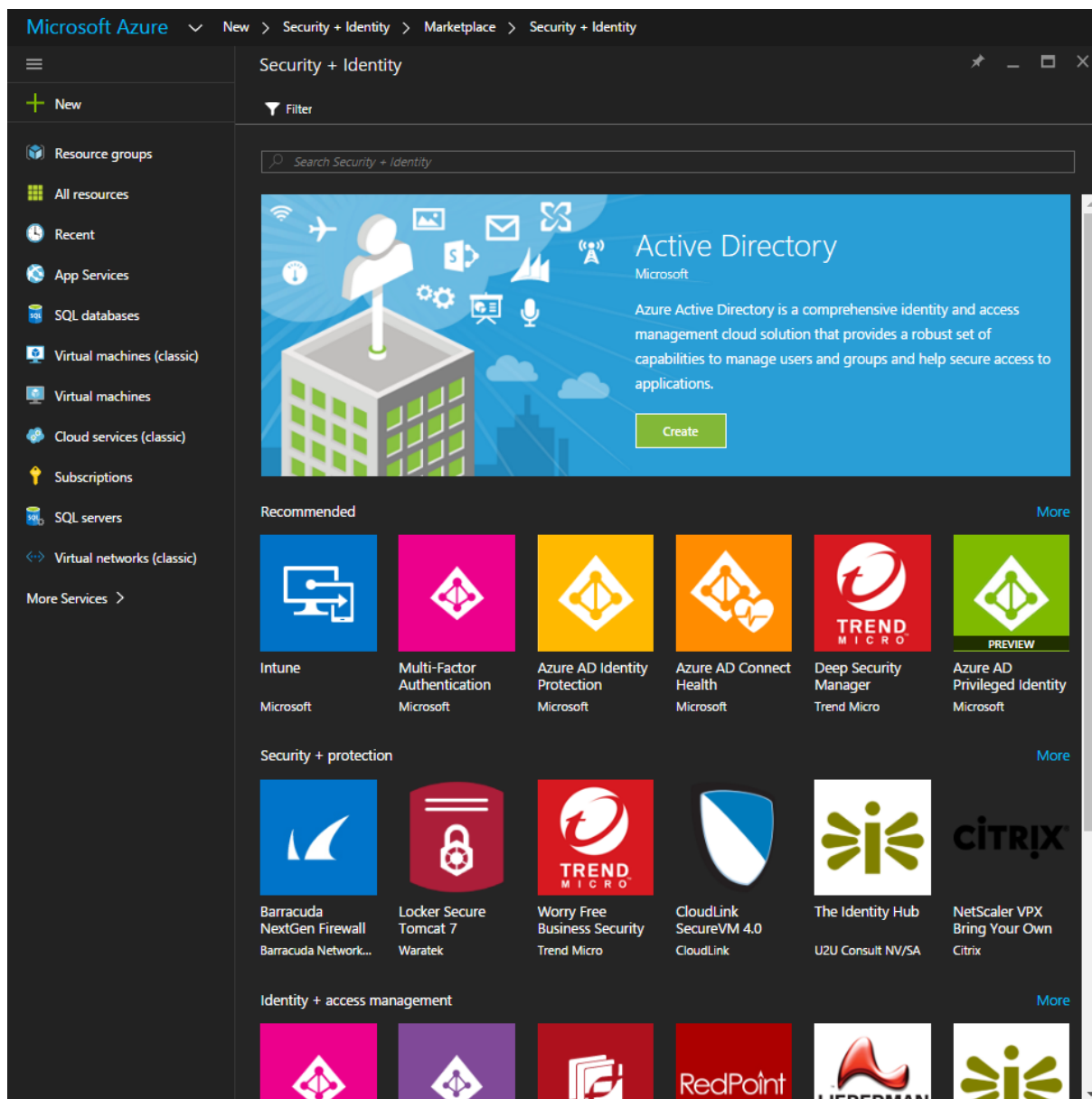
Slika 10 - Azure IoT servisi

Networking servisi omogućavaju stvaranje virtualnih mreža, analitičkih servisa za analiziranje podataka, sustava koji preraspodjeljuje(engl. Load balancer), čija je uloga preraspodijeliti dolazni promet između više instanci virtualnih mašina koje se nalaze u pozadini i mnoge druge. Ovi servisi mogli bi se na primjer iskoristiti za izgradnju privatnih mreža u oblaku sa vlastitim IP adresama i DNS serverima zajedno sa osiguranim konekcijama unutar VPN-a(engl. Virtual private network).



Slika 11 - Azure networking

Security + Identity servisi omogućavaju stvaranje *Active Directory*-a, ovjeravanje autentičnosti u više koraka, preventivne alate za praćenje sigurnosti i druge. Jedna od glavnih primjena ovih servisa jest mogućnost administriranja prava pristupa aplikacijama.



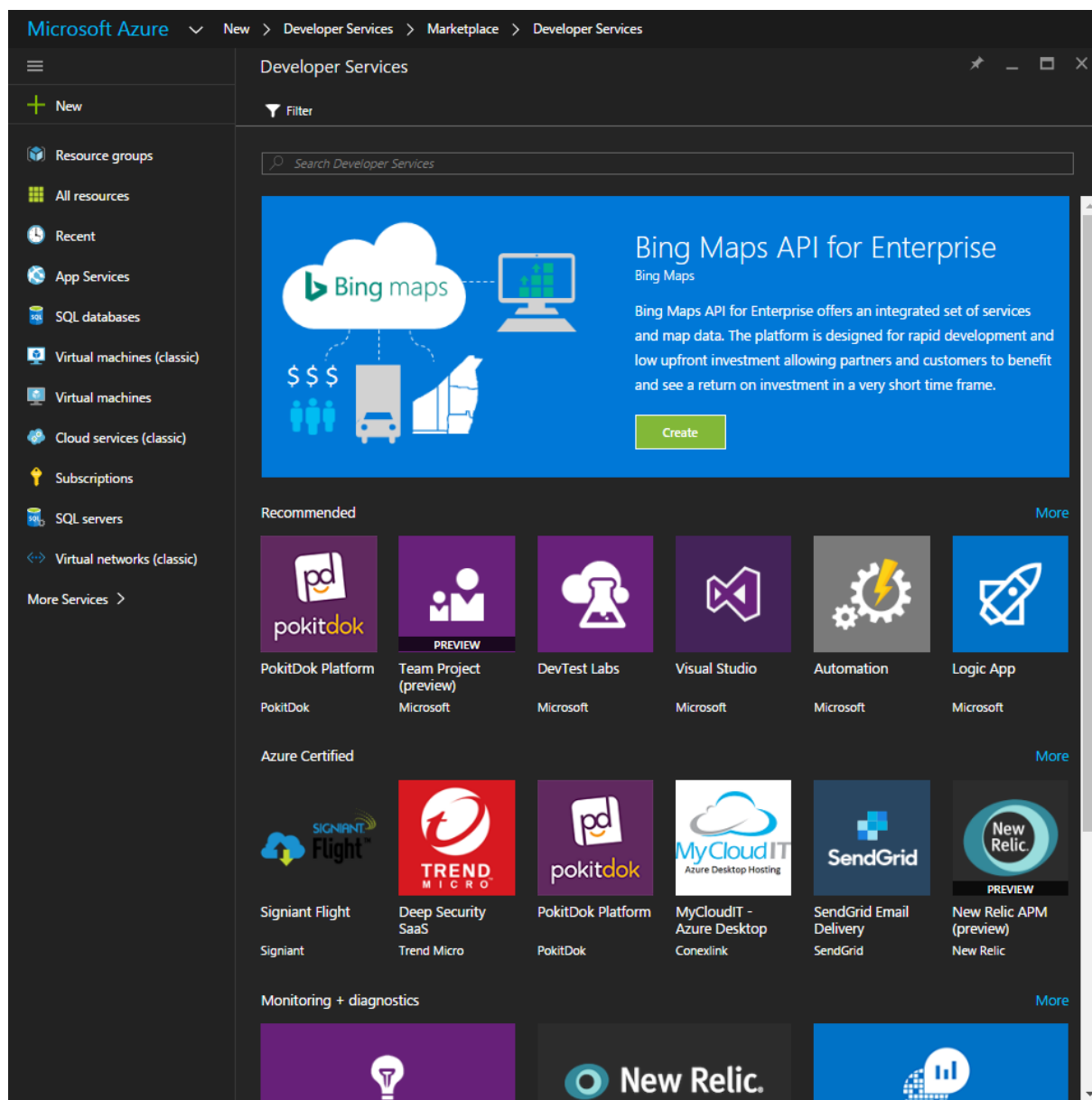
Slika 12 - Azure sigurnost

Developer services servisi jedan su od novijih resursa a omogućavaju rad i suradnju na projektima kao. Tako na primjer, *Application Insights*¹² resurs omogućava praćenje performansi, dostupnosti i korištenja pojedinih servisa. *Insights* alati preaktivno detektiraju moguće probleme te obavještavaju administratore pomoću na primjer emaila. Moguće je također korištenje *SendGrid*¹³ servisa, koji je ujedno i najveći servis

¹² Application Insights - <https://azure.microsoft.com/en-us/services/application-insights/>

¹³ SendGrid - <https://azure.microsoft.com/en-us/marketplace/partners/sendgrid/sendgrid-azure/>

za slanje emaila baziran u oblaku ili ipak korištenje resursa za prepoznavanje lica, kao na primjer *Face Api*¹⁴.



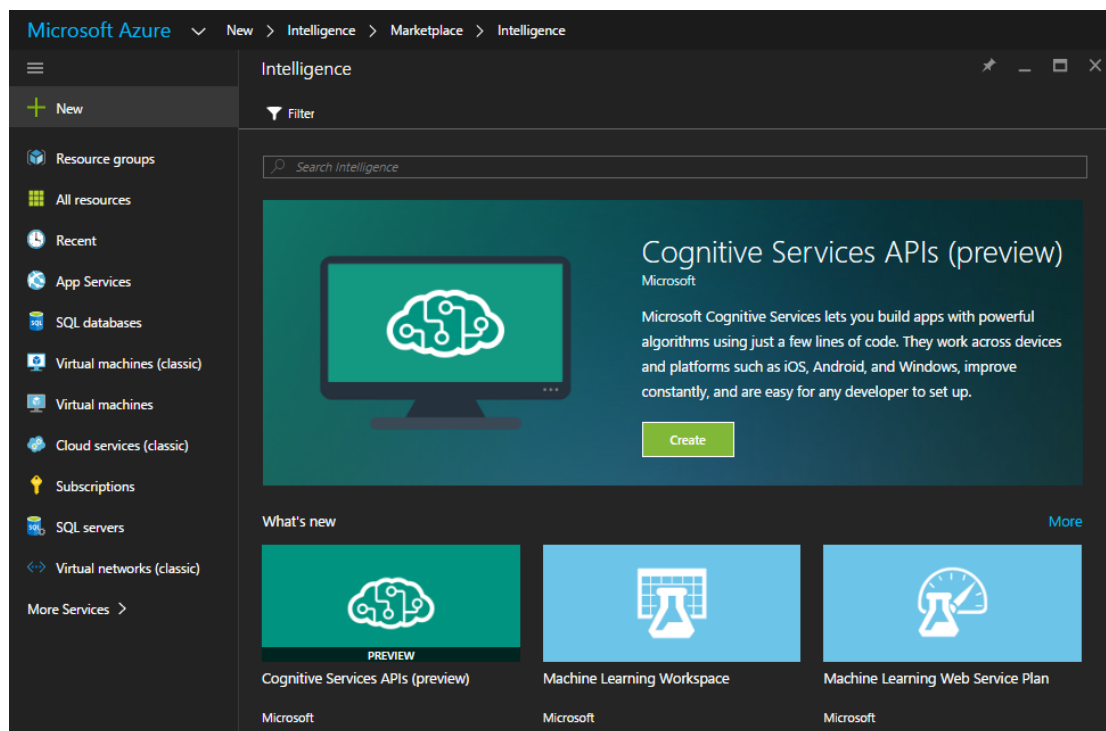
Slika 13 - Azure Developer

Intelligence servisi omogućavaju stvaranje i dijeljenje naprednih rješenja za analitiku. Ovo se najviše odnosi na *machine learning*. *Machine learning* servis dolazi sa već predefiniranim algoritmima koji su podijeljeni u četiri glavne kategorije. On tako sadrži

¹⁴ FaceApi - <https://azure.microsoft.com/en-us/marketplace/partners/faceapis/faceapis/>

Clustering, *Regression*, *Classification* i *Anomaly Detection*. *Clustering* se tako na primjer može koristiti za preporuku poklona. Njegov algoritam može pronaći uzorke među podacima te ih grupirati po sličnosti. *Regression* algoritam može poslužiti za predlaganje cijena kuća. Iz ulaznih podataka gdje se navedu parametri kao što su broj soba, kupaonica, kvadratura, cijena i drugi, algoritam može odrediti veze između tih podataka te za daljnje unose sam predložiti cijenu. *Classification* algoritmi mogu se koristiti za predviđanje oboljenja od raka dojke kod žena. Sa raznim ulaznim zdravstvenim podacima, poviješću bolesti i drugim podacima u mogućnosti je zaključiti da li i kada bi moglo doći do oboljenja. *Anomaly Detection* algoritam ima mogućnost prepoznati anomalije u postojećim podacima, kao na primjer odstupanje od dopuštene temperature u plinovodu ili slično. Ovi scenariji i primjeri opisani su detaljnije na blogu *Microsoft-a*¹⁵. S druge strane, ukoliko se koristi kognitivni servis, moguće ga je također upotrijebiti za mnoge scenarije. Analiza teksta po tematici ili semantici, analiza gramatike, auto sugeriranje, prepoznavanje, organiziranje i analiziranje lica u fotografijama, prepoznavanje emocija, algoritmi za preporuke, pretvaranje zvuka u tekst i mnogi drugi. Kognitivan servis također nudi mnogobrojne opcije i gotove algoritme.

¹⁵ Machine Learning Algoritmi - <https://blogs.msdn.microsoft.com/jennifer/2015/09/08/choosing-an-azure-machine-learning-algorithm/>



Slika 14 - Azure Intelligence

2.3.1. Primjeri ostalih komercijalnih računalnih oblaka

Radi sve veće popularnosti oblaka, danas postoji nekoliko većih korporacija koje pružaju usluge računarstva u oblaku. Najpoznatiji od njih su *Amazon*, *Google* i *Microsoft*. Konkurencija između ovih korporacija raste iz dana u dan dok cijene usluga padaju a dolazak novih značajki stalno se povećava. Najveći pružatelji rješenja u oblaku jesu:

- ▶ Amazon Web Services(AWS)
- ▶ Google Cloud Platform(GCP)
- ▶ Microsoft Azure

Amazon Web Services sveobuhvatna je platforma koja se stalno razvija, pokrenuta 2006 godine u vlasništvu Amazona. Primarna djelatnost jest pružanje usluga u oblaku. Kako bi smanjili udar na performanse svojih servisa, povećali broj dostupnih sati i robusnost cijele platforme, AWS je geografski podijeljen na regije koje se nadalje dijele na manja geografska područja zvana zone. Amazonu se bliži deseta godišnjica a činjenica da su bili prvi koji su izašli sa svojim servisima daje im prednost

nad konkurencijom. Osnovne značajke AWS-a sastoje se od *Compute* usluga, pohrane podataka i isporuke sadržaja, baza podataka i umrežavanja. Sve to odvija se ispod Amazon-ove opsežne administratorske platforme koja uključuje upravljanje identitetom, revizije, enkripciju podataka, kontrole pohrane podataka, nadgledanje i pohranu podataka o korištenju. *Compute* usluge podrazumijevaju zakup virtualnog okruženja sa mogućnošću korištenja Internet usluga za pokretanje okolina sa različitim operacijskim sustavima. Amazon je bio prvi od ove 3 korporacije koji je izašao sa ovim uslugama te im je to zasigurno dalo prednost pred konkurencijom.

Google cloud platform skup je javnih servisa u oblaku, osnovan 2008 godine u vlasništvu Google-a. Platforma također obuhvaća popriličan broj dostupnih servisa za *Compute*, pohranu podataka i razvoj aplikacija koje se vrte na infrastrukturi od *Google*-a. Servisima na *Google* platformi mogu pristupati razvojni inženjeri, administratori oblaka i ostali članovi, preko interneta ili neke druge privatne mrežne infrastrukture, kao što je to VPN. Postoje četiri glavna servisa koje omogućava *Google* platforma. *Google compute engine* pruža infrastrukturu kao servis nudeći korisnicima instance virtualnih računala za obavljanje zadataka. *Google app engine* pruža platformu kao servis nudeći tako razvojnim inženjerima pristup *Google*-ovom skalabilnom sustavu za podizanje aplikacija. Inženjeri također mogu iskoristiti *Google*-ov set biblioteka za razvoj produkata koji se mogu vrtjeti na *app engine*-u. *Google cloud storage* dizajniran je na način da pohranjuje ogromne količine podataka kao i nestrukturirane setove podataka. *Google* tako nudi NoSQL(engl. Non relational storage) baze, Sql(engl. structured query language) baze, točnije MySQL bazu kao i *Google-ovu Cloud Bigtable* bazu. *Google container engine* nudi sustav namijenjen za pokretanje aplikacija i biblioteka potrebnih za njihov rad na Linux platformi, omogućavajući aplikacijama da budu zapakirane unutar nekog okvira pomoću *Docker*-a. *Google* podržava još podosta servisa i koji se stalno razvijaju sa novim i boljim mogućnostima.

2.3.2. Usporedba: Microsoft Azure vs. Amazon AWS

Na trenutnom tržištu pružatelja usluga u oblaku, Amazon AWS glavni je predvodnik među postojećim tvrtkama. Procijenjeno je da AWS posjeduje oko deset puta više *Compute* kapaciteta naspram njegovih 14 najbližih konkurentskih kompanija

zajedno, uključujući i *Microsoft Azure*. Međutim, unatoč dominaciji *Amazon-a*, *Microsoft* je također izgradio ogromnu mrežu u oblaku. Iako nije veličinom i kapacitetom kao u *Amazon-u*, ipak je poprilično iznad svojih najbližih konkurenata. Odabir jednog od pružatelja usluga naspram drugog u osnovi se svodi na pojedinačne potrebe klijenata kao i potrebe rješenja koje je potrebno pogoniti. Zapravo, nerijetko je i slučaj da organizacije koriste više različitih pružatelja usluga u oblaku unutar istog rješenja, za različite tokove aplikacija tj. rješenja. *Amazon AWS* i *Microsoft Azure* do neke granice nude slične mogućnosti, u područjima *Compute*, pohrane podataka i mrežnih usluga. Oba pružatelja dijele neke zajedničke elemente javnog oblaka kao što su vlastito korisničko posluživanje kroz administracijske portale kao i zakup i kreiranje instanci u stvarnom vremenu, automatsko skaliranje, sigurnost i upravljanje značajkama identiteta. Obje kompanije također stalno ulažu sredstva u neprestanu potražnju za novim servisima u oblaku. Ovakva potražnja rezultirala je zrelijom nego ikad ponudom analitičkih servisa. Tako na primjer oba podržavaju *Hadoop*¹⁶ cluster-e. I *Amazon* i *Azure* dodali su servise posvećene *Machine learning-u* sa ciljem razvoja uređaja na IoT(engl. Internet of things) servisima. Ovo omogućuje korisnicima da na obje platforme razvijaju mobilne aplikacije ili stvaraju razvojne okoline visokih performansi bazirane na njihovim vlastitim potrebama. Virtualni strojevi u oblaku se najviše smatraju infrastrukturom kao servisom. Ovo ujedno znači da instance virtualnog stroja su zapravo prava računala sa svojom procesorskom moći, memorijom, mrežnim mogućnostima kao i kapacitetom pohrane podataka. Kada bismo usporedili *Compute* mogućnosti tih dviju platformi, dobili bismo sljedeću tablicu.

	Amazon EC2	Microsoft Azure VM
<i>Broj dostupnih unaprijed podešenih instanci</i>	39	40
<i>GPU Akceleracija</i>	Da	Ne
<i>Ručno kreiranje instanci</i>	Ne	Ne
<i>Limit procesora</i>	1-40	1-32

¹⁶ Hadoop - <http://searchbusinessanalytics.techtarget.com/definition/Hadoop-cluster>

<i>Limit memorije</i>	0,5 – 244GB	0,75 – 448GB
<i>Privremeni limit pohrane podataka</i>	Do 48TB(Više diskova)	2TB
<i>Podržane mrežne mogućnosti</i>	CDN, Direct connection, DNS, Load Balancing, Virtual private cloud network, VPN Gateway	

Slika 15 - Compute usporedba - Amazon – Azure

Važno je napomenuti kako konfiguracije iznimno visokih performansi možda nisu dostupne u pojedinim regijama. Uz odabir konfiguracija, ne samo da se može smanjiti ili povećati snaga virtualnog stroja može ga se također prilagoditi trenutnim potrebama odnosno skalirati na više ili manje po potrebi. *Amazon* tako može i automatski skalirati kao i preraspodijeliti virtualni stroj. Zadatak automatskog skaliranja sastoji se od odabira željenog broja instanci kao i uključivanje i isključivanje ove mogućnosti. Mogućnost automatskog skaliranja korisna je i iz razloga što pomaže održati virtualni stroj na vrhu performansi, ponovno pokrećući računalo ako je to potrebno. Skaliranje kod *Microsoft Azure* platforme funkcionira na način da se odabere broj instanci za koje se želi povećati ili smanjiti kao i kriterij za aktiviranje te mogućnosti. Kriterij tako na primjer može biti zauzeće procesora ili broj zahtjeva na čekanju(engl. Queue size). Sve instance mogu se lagano preraspodijeliti koristeći *Azure* administracijsku aplikaciju ili *PowerShell*¹⁷ naredbe. Nadalje, obje platforme podržavaju širok spektar operacijskih sustava koji su podržani. *Amazon* tu ima blagu prednost pošto *Azure* ne podržava *CloudLinux*¹⁸ i *FreeBSD*¹⁹ operacijske sustave. Podrška za operative sustave također može varirati ovisno od regije do regije. Dostupnost je također jedna od bitnih stavki i obje platforme pružaju SLA(engl. Service Level Agreement) na koji se pristaje zajedno sa uvjetima korištenja platformi. Sa njime se doslovno zadaje odgovornost pružatelja usluga na kvalitetu njegovih postrojenja. Ukoliko dođe do kršenja, korisnici imaju pravo na određene pogodnosti. *Amazon* tako pruža popust od

¹⁷ PowerShell - <https://azure.microsoft.com/en-us/documentation/articles/powershell-install-configure/>

¹⁸ CloudLinux - <https://cloudlinux.com/>

¹⁹ FreeBSD - <https://www.freebsd.org/>

10% ukoliko je virtualni stroj bio nedostupan duže od 0,05% vremena unutar cijelog mjeseca. 30% online kredita dostupno je ako je pristup virtualnom stroju bio onemogućen više od 1% a minimalna jedinica za nedostupnost koja se broji jest 1 minuta. *Microsoft Azure* SLA jest jednak kao onaj od *Amazon-a* sa razlikom da je maksimum za kredit 25%. Jedan od također bitnih mogućnosti jest stvaranje sigurnosnih kopija. *Amazon* pruža dva glavna načina stvaranja takvih kopija, pomoću stvaranja slike virtualnog stroja(engl. Image creation) i stvaranjem snimke(engl. Snapshot). Obje varijante stvaraju kopiju cijelog diska, ali stvaranjem snimke može se provjeriti konzistentnost pohranjenih podataka. Za dodatnu naknadu, kopije se mogu pohraniti na različitim servisima. Kopije dokumenata također su dostupne, ali su za njihovo korištenje potrebne dodatne skripte i alati za automatizaciju. *Azure* posjeduje sve značajke koje su navede iznad kao i dodatno poseban alat za stvaranje kopija kao i servis za oporavak. Za one kojima ovo nije dovoljno, može se dodatno uklopiti i *CloudBerry*²⁰ sustav za pohranu. Cijena je također jedna od važnijih stavki. U većini stavki, cijene između *Amazon-a* i *Azure-a* su usporedive. Nadmetanje među ovih dviju platformi bilo je iznimno korisno za sve korisnike pošto rezultira čestim spuštanjem cijena za razne servise. Sukladno tome, kako bi se olakšalo određivanje cijena za usluge obje platforme posjeduju kalkulator za izračun cijene usluga. Kalkulator za *Azure usluge* mogu se naći na službenoj *Azure* stranici²¹, dok se kalkulator za usluge na *Amazon* platformi nalazi na *Amazon* službenoj stranici²². Obje platforme također nude ulazne besplatne cjenovne rangove za upoznavanje sa platformom.

Obje platforme imaju svoje prednosti i nema jasnog pobjednika. Odabir će ovisiti o pojedinačnim preferencijama ili potrebama klijenata pošto i *Amazon* i *Azure* oboje pružaju skup jedinstvenih značajki, prednosti i mana. Konačna usporedba tih dviju platformi izgledala bi ovako:

²⁰ CloudBerry - <http://www.cloudberrylab.com/>

²¹ Azure kalkulator - <https://azure.microsoft.com/en-gb/pricing/calculator/>

²² Amazon kalkulator - <https://calculator.s3.amazonaws.com/index.html>

	Amazon EC2	Microsoft Azure
<i>Broj dostupnih unaprijed podešenih instanci</i>	39	40
<i>GPU Akceleracija</i>	Da	Ne
<i>Ručno kreiranje instanci</i>	Ne	Ne
<i>Limit procesora</i>	1-40	1-32
<i>Limit memorije</i>	0,5 – 244GB	0,75 – 448GB
<i>Privremeni limit pohrane podataka</i>	Do 48TB(Više diskova)	2TB
<i>Podržane mrežne mogućnosti</i>	CDN, Direct connection, DNS, Load Balancing, Virtual private cloud network, VPN Gateway	
<i>Broj podržanih operacijskih sustava</i>	11	9
<i>Automatsko skaliranje</i>	Da	Da
<i>Promjena veličine</i>	Da	Da
<i>SLA</i>	Kredit za 1+ minutu nedostupnosti, maksimalan mjesečni iznos od 30%, dostupnost SLA 99.95%	Kredit za 1+ minutu nedostupnosti, maksimalan mjesečni iznos od 25%, dostupnost SLA 99.95%
<i>Podrška za CloudBerry</i>	Da	Da
<i>Besplatni period</i>	Da, ograničen vremenom na jednoj instanci	Da, ograničen vremenom i resursima na bilo kojoj instanci

Slika 16 - Konačna usporedba - Amazon – Azure

3. Skalabilne aplikacije

Pojam visoke skalabilnosti kada su u pitanju web aplikacije sve se više spominje u IT društvu. Sa porastom aplikacija koje se vrte u oblaku, pojam skalabilnosti postao je jedna od ključnih stavki za klijente kao i krajnje korisnike usluga u oblaku. Međutim, iako skalabilne aplikacije imaju svoje prednosti, nisu sva rješenja danas rađena sa tim principom u vidu. Razlog zašto je to tako jest što su visoko skalabilne aplikacije veće složenosti i popratno time veće cijene od ostalih. Kod izrade takvih rješenja, treba imati u vidu da korist od izrade skalabilnog rješenja nadmašuje njegovu kompleksnost i cijenu.

Skalabilne aplikacije jesu one koje su u mogućnosti bez većih zastoja ili problema u performansama nastaviti raditi glatko i konzistentno bez obzira na stalan porast aktivnih korisnika ili nagli porast prometa u kratkom vremenu (engl. Slashdot effect).



Slika 17 – Primjer Slashdot efekta

Primjer takvog scenarija jest natječaj za smještaj u studentskim domovima studentskog centra u Zagrebu. Na dan kada se otvara mogućnost za prijavu u pojedine sobe, performanse stranica studentskog centra padaju na razinu neupotrebljivosti. Ovo se događa radi ogromne količine korisnika koji pristupaju aplikaciji u kratkom vremenskom periodu tj. istovremeno. Takvi problemi i zastoji u radu aplikacija rješavaju se upravo izradom skalabilnih sustava. Pojam skalabilnosti međutim ne odnosi se na čiste performanse kao što je to na primjer razlika u brzini procesora između 2 GHz i 3 GHz. Skalabilnost također ne podrazumijeva različite operacijske sustave kao što je to Linux naspram Windowsa, različite softverske tehnologije kao na primjer Java naspram Python-a ili Ruby-a ili optimizaciju koda, deset linija koda naspram tisuću linija koda.

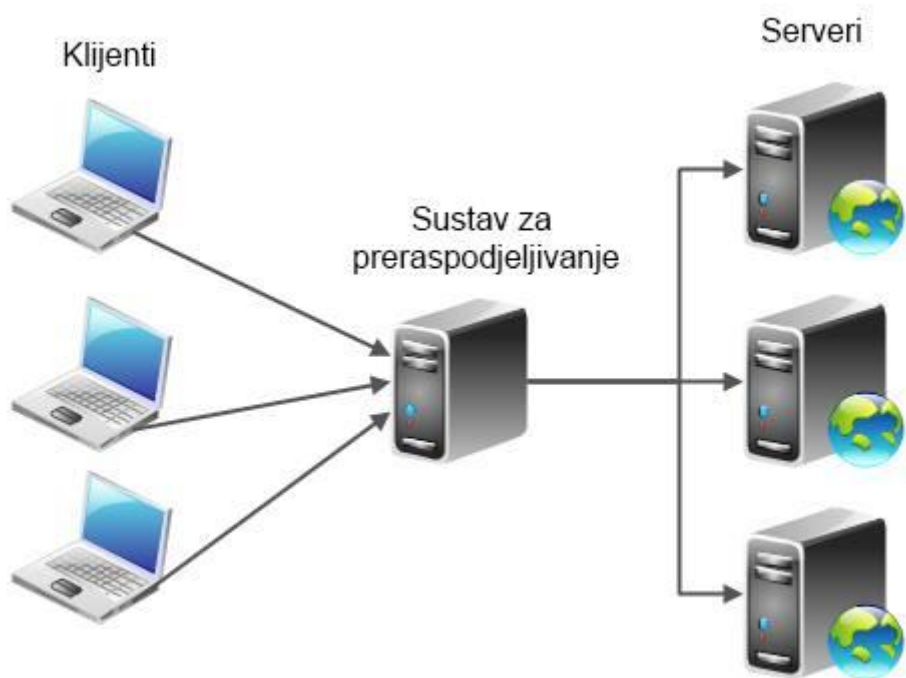
Proces izrade skalabilnih web aplikacija počinje dizajniranjem arhitekture web aplikacije. Kada se radi dizajn arhitekture imajući u vidu skalabilnost aplikacije, mogu se unaprijed izbjeći mnogi problemi koji bi mogli nastati. Drugi korak jest implementacija rješenja na osnovu prethodnog dizajna. Kod implementacije arhitekture, cilj je testirati aplikaciju i njezine performanse tako da može nesmetano raditi sa opterećenjem većim pet do dvadeset puta od predviđenog. Razlog zašto bi se aplikacija trebala testirati na ovakav način jest što je svaka promjena u ovoj fazi razvoja skup proces, pošto baza i kod aplikacije već postoje. Treći i posljednji korak odnosi se na postavljanje aplikacije u produkcijsko okruženje. Ovaj korak odnosi se na *Microsoft Azure* te mogućnosti koje takva platforma pruža kod izrade skalabilnih rješenja.

Porast performansi ne zahtjeva ogromne promjene u programskom kodu aplikacije ili arhitekturi servera. Ako postoji mogućnost da broj krajnjih korisnika aplikacije poraste u budućnosti, skalabilnost bi trebala biti jedan od glavnih prioriteta kod dizajniranja arhitekture aplikacije. U današnje vrijeme, izrada skalabilnih aplikacija lakša je nego ikada uz pomoć servisa u oblaku kao što su *Microsoft Azure*, *Amazon AWS* ili drugi. Međutim, ključna stavka kod odabira jest da je platforma koja se odabere kompatibilna sa arhitekturom aplikacije.

Nažalost, skaliranje aplikacija nije tako jednostavno kao povećanje ili zamjena hardverskih dijelova na serveru. Povećanje hardverskih mogućnosti može biti skup odabir a i potrebno je imati na umu da takav pristup ima svoje granice. Nadalje, kao dodatan nedostatak takvog pristupa jest nedostupnost servera koja se događa za

vrijeme nadogradnje servera. Dodavanje instanci servera rješenje je koje rješava problem skalabilnosti, ali kako bi se podržalo pokretanje više instanci potrebno je imati na umu dvije važne odluke kod izrade arhitekture aplikacije. Izrada aplikacije koja nije ovisna o stanju servera i neograničena i krajnje particionirana baza podataka.

Aplikacije koje nisu zavisne o stanju servera jesu one koje ne zavise o podacima spremljenima u memoriju ili podatkovni sustav servera. Takav scenarij može se objasniti na primjeru prijave korisnika pomoću sesije²³(engl. Session). Ako se korisnik prijavi u sustav, server bi trebao spremiti token sesije u bazu ili neki drugi vanjski izvor podataka. Token se ne bi trebao spremati u memoriju. Razlog zašto memorija nije pogodna jest taj da kada se pokrene druga instanca servera, podaci korisnika mogli bi završiti razdijeljeni između tih dviju instanci. To se može desiti iz razloga jer sustav koji preraspodjeljuje korisnike(engl. Load balancer) može u jednom trenutku prebaciti korisnika na drugu instancu servera, dok podaci o njegovoj prijavi ostanu spremljeni u memoriji drugog servera.



Slika 18 – Primjer sustava za preraspodjeljivanje(engl. Load balancer)

²³ Session - [https://en.wikipedia.org/wiki/Session_\(computer_science\)](https://en.wikipedia.org/wiki/Session_(computer_science))

U tom trenutku, korisnika bi se neočekivano odjavilo iz aplikacije što nije željeno ponašanje. S druge strane, ako se podaci spremaju u bazu podataka, onda obje instance servera mogu dohvatiti taj podatak. Naime, u tom slučaju nije bitno koliko instanci servera postoji jer bez obzira na to, izvor podataka je zajednički. Tu se naravno podrazumijeva da se baza podataka ne nalazi spremljena na lokalnom sustavu datoteka na nekoj od instanci servera. Druga stvar koju treba uzeti u obzir jest neograničena i particionirana baza podataka. Problem kod korištenja tradicionalnih, lokalno pokrenutih baza podataka jest što za aplikacije koje rastu eksponencijalno, takva baza podataka ubrzo dosegne svoj maksimum te samo dohvaćanje podataka jednostavno postane predugo. Problem može vrlo brzo eskalirati ukoliko aplikacija nudi mogućnost spremanja sadržaja kao što su to slike ili dokumenti od strane korisnika. *Microsoft Azure* u tu svrhu nudi *Table storage*²⁴ rješenje koje u potpunosti rješava taj problem. *Table storage* sam upravlja instancama baze podataka. Ono što bi bio važan korak integracije jest da su podaci logički particionirani. Svakom podatku koji se unosi u bazu podataka trebao bi se pridijeliti ključ za particioniranje. Ako aplikacija na primjer omogućuje korisnicima prijenos slika za dijeljenje sa drugim ljudima, dobra ideja bila bi particionirati prenesene slike po npr. *userId* ključu osobe koja je izvršila prijenos, tako da ako se dogodi da podaci aplikacije narastu, *Azure* će automatski proširiti bazu podataka na više servera. Bez obzira na to što su podaci razdijeljeni na više servera, *Azure* čuva podatke organiziranim po particijama koje su mu zadane, u ovom slučaju *userId*, tako da kada se zatraži dohvat na primjer dviju slika sa dvije različite instance, odaziv servera trebao bi biti podjednake dužine. *Table storage* također ima odlične opcije za spremanje podataka u binarnom obliku. Takvi scenariji rješavaju se pomoću *Blob storage* resursa.

Iza ovih odluka vezanih uz arhitekturu aplikacije, od velikog je značaja ako je sam server asinkron kako bi mogao obraditi više istovremenih zahtjeva. Više o tome kasnije u poglavlju.

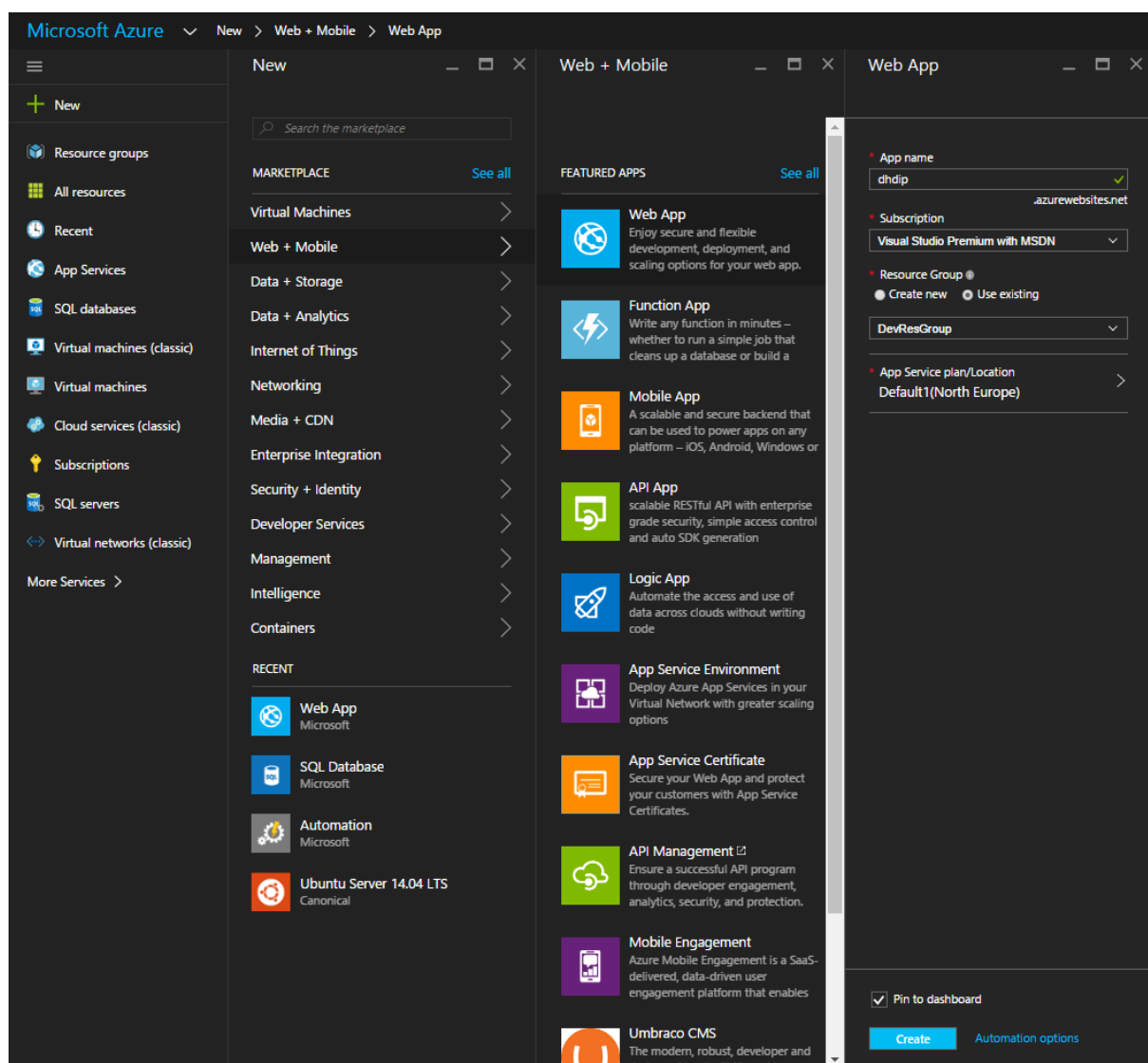
Nakon izrade aplikacije koja ne ovisi o stanjima servera i koristi sve prednosti particioniranih baza podataka, sljedeći korak bio bi odlučiti koju od platformi odabrati za aplikaciju. *Microsoft Azure* u tu svrhu nudi 3 mogućnosti.

²⁴ Azure Table Storage - <https://azure.microsoft.com/en-us/documentation/articles/storage-dotnet-how-to-use-tables/>

- ▶ Azure virtualne mašine(engl. Azure VMs)
- ▶ Azure Internet aplikacije(engl. Azure Web Apps)
- ▶ Azure servisi u oblaku(engl. Azure Cloud Services)

Dobro je imati na umu da uz skalabilnu platformu koja se dobiva uz *Microsoft Azure*, jedna od stavki koja se može dobiti uz cijenu jest server održavan od strane *Azure-a*. *Azure* virtualna mašina također je opcija za skalabilnost, ali treba imati na umu da je takvo rješenje potrebno održavati sa sigurnosnim zakrpama i nadogradnjama, slično kao i vlastiti fizički server koji bi se pokrenuo lokalno ili u nekom podatkovnom centru. *Azure cloud service* korak je dalje od virtualne mašine. To je visoko skalabilna platforma na koju se je i dalje moguće prijaviti sa nekog udaljenog mjesta kako bi se napravile željene promjene. Međutim, kod odabira takvog rješenja treba imati na umu da to nije dosljedan server tj. održavan je od strane *Microsoft-a* što znači da određene instance servera mogu biti uništene ili ponovno rekreirane u bilo kojem trenutku od strane *Azure-a*. To ujedno znači da se ne bi trebali spremati nikakvi trajno namijenjeni podaci na sustav datoteka na toj instanci. Takav servis u oblaku odličan je izbor ako je nužno prepustiti održavanje servera pružatelju usluga uz zadržavanje mogućnosti instaliranja dodatnog softvera. Dodatan softver može se instalirati pomoću skripti u trenutku kada se stvara server. *Azure* Internet aplikacije s druge su strane virtualnih mašina. *Azure* Internet aplikacije visoko su skalabilna rješenja koja su u potpunosti održavana od strane *Microsoft-a*. Na takva rješenja nije se moguće spojiti pomoću *Remote desktop*²⁵ aplikacija. Ovo je odličan izbor ako aplikacija ne zahtjeva nikakav dodatan softver koji je potrebno pokrenuti na serveru. Pokretanje aplikacija na ovakvom resursu može se napraviti unutar nekoliko minuta. Sve što je potrebno napraviti kako bi se kreirala Internet aplikacija na *Azure* platformi jest pozicionirati se na administracijskom sučelju aplikacije unutar New -> Web + Mobile -> Web App, popuniti željene podatke za aplikaciju te klikom na gumb *Create*, *Azure* će stvoriti novu Internet aplikaciju.

²⁵ Remote Desktop - <https://support.microsoft.com/en-us/help/17463/windows-7-connect-to-another-computer-remote-desktop-connection>



Slika 19 - Azure, kreiranje nove Internet aplikacije

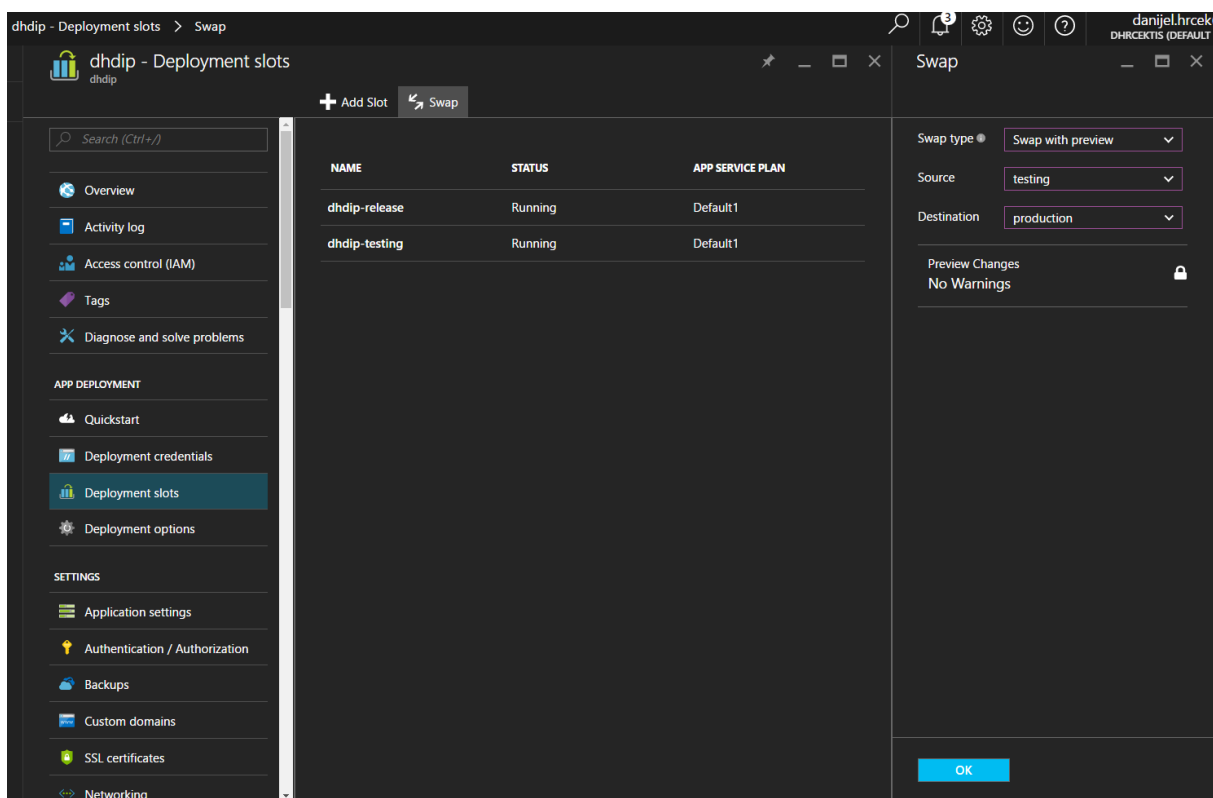
Također je moguće vrlo lako podesiti SSL²⁶ za testiranje bez potrebe za procesom kreiranja i prijenosa testnog certifikata na *Azure*. *Azure* također nudi mogućnost pregleda svih serverskih logova unutar zasebne konzole.

Još jedna od prednosti korištenja *Azure* Internet aplikacija ili servisa u oblaku jest da se sve daljnje nadogradnje aplikacije mogu izvesti bez ikakvog prestanka ili zastoja u radu. Ovo je moguće izvesti pomoću *Deployment slots*²⁷ mogućnosti. Tako se naime mogu definirati različita okruženja (engl. Slot) sukladno potrebama testiranja

²⁶ SSL - <https://hr.wikipedia.org/wiki/SSL>

²⁷ Deployment slot - <https://azure.microsoft.com/en-us/documentation/articles/web-sites-staged-publishing/>

ili razvoja. Aplikacije koja sadrži potrebne nadogradnje može se prebaciti na okruženje za testiranje i nakon što se zaključi da su sve nadogradnje ispravne, okruženja se mogu jednostavno zamijeniti i sav promet će se početi odvijati kroz okruženje za testiranje, koji je samim time zapravo postao produkcija. Sve to odvija se bez da korisnici uopće znaju da se ikakva promjena dogodila. Ova mogućnost odvija se klikom na *Swap* gumb unutar administracijskog sučelja.



Slika 20 - Azure deployment slot

Isto se događa i kod skaliranja aplikacije na viši ili manji servisni plan, aplikacija je i dalje pokrenuta bez ikakvih vidljivih promjena. *Microsoft Azure* nudi mogućnost odabira cjenovnog ranga aplikacije podijeljenog u 4 glavne skupine

- ▶ Besplatan i dijeljen(engl. Free and Shared service plan)
- ▶ Osnovan(engl. Basic service plan)
- ▶ Standardan (engl. Standard service plan)
- ▶ Premium (engl. Premium service plan)

Besplatan servisni plan idealan je za eksperimentiranje sa platformom kao i za razvoj i testne scenarije. Dijeljeni servisni plan dodatno sadrži i značajke za SSL i domenska imena. Važno je napomenuti i da se metrike za ova dva plana također odvijaju po aplikaciji a ne po instanci aplikacije, za razliku od ostalih triju planova. Osnovan servisni plan namijenjen je aplikacijama koje imaju nizak nivo prometa i kod kojih nisu potrebne napredne mogućnosti automatskog skaliranja i praćenja prometa. Standardan servisni plan namijenjen je za produkcijske Internet i mobilne aplikacije i servise. Ovaj plan uključuje ugrađenu opciju za automatsko skaliranje kao i ugrađenu opciju za stvaranje sigurnosnih kopija. Premium servisni plan namijenjen je za produkcijske aplikacije koje trebaju podržavati velik broj korisnika. Ovaj plan podržava velik broj instanci namijenjenih za skaliranje, dodatne konekcije, *BizTalk*²⁸ mogućnosti zajedno sa svim mogućnostima sa standardnog servisnog plana i mnoge druge.

²⁸ BizTalk - <https://www.microsoft.com/en-us/cloud-platform/biztalk>

dh dip > Choose your pricing tier

dh dip

Choose your pricing tier

Browse the available plans and their features

App Service Environments are available in the Premium tier. They offer even greater scale options, private access, and more. [Learn more](#)

P1 Premium	P2 Premium	P3 Premium
1 Core	2 Core	4 Core
1.75 GB RAM	3.5 GB RAM	7 GB RAM
BizTalk Services	BizTalk Services	BizTalk Services
250 GB Storage	250 GB Storage	250 GB Storage
Up to 20 instances * Subject to availability	Up to 20 instances * Subject to availability	Up to 20 instances * Subject to availability
20 slots Web app staging	20 slots Web app staging	20 slots Web app staging
50 times daily Backup	50 times daily Backup	50 times daily Backup
Traffic Manager Geo availability	Traffic Manager Geo availability	Traffic Manager Geo availability
207.05 EUR/MONTH (ESTIMATED)	414.09 EUR/MONTH (ESTIMATED)	828.19 EUR/MONTH (ESTIMATED)

S1 Standard	S2 Standard	S3 Standard
1 Core	2 Core	4 Core
1.75 GB RAM	3.5 GB RAM	7 GB RAM
50 GB Storage	50 GB Storage	50 GB Storage
5 SNI, 1 IP Custom domains / SSL	5 SNI, 1 IP Custom domains / SSL	5 SNI, 1 IP Custom domains / SSL
Up to 10 instances Auto scale	Up to 10 instances Auto scale	Up to 10 instances Auto scale
Daily Backup	Daily Backup	Daily Backup
5 slots Web app staging	5 slots Web app staging	5 slots Web app staging
Traffic Manager Geo availability	Traffic Manager Geo availability	Traffic Manager Geo availability
37.64 EUR/MONTH (ESTIMATED)	75.29 EUR/MONTH (ESTIMATED)	150.58 EUR/MONTH (ESTIMATED)

B1 Basic	B2 Basic	B3 Basic
1 Core	2 Core	4 Core
1.75 GB RAM	3.5 GB RAM	7 GB RAM
10 GB Storage	10 GB Storage	10 GB Storage
Custom domains	Custom domains	Custom domains
Up to 3 instances Manual scale	Up to 3 instances Manual scale	Up to 3 instances Manual scale

Select

Deployment credentials

Deployment slots

Deployment options

SETTINGS

Application settings

Authentication / Authorization

Backups

Custom domains

SSL certificates

Networking

Scale up (App Service plan)

Scale out (App Service plan)

Security Scanning

WebJobs

MySQL In App (preview)

Properties

Locks

Automation script

APP SERVICE PLAN

App Service plan

Quotas

Change App Service plan

DEVELOPMENT TOOLS

Clone app

Console

Advanced Tools

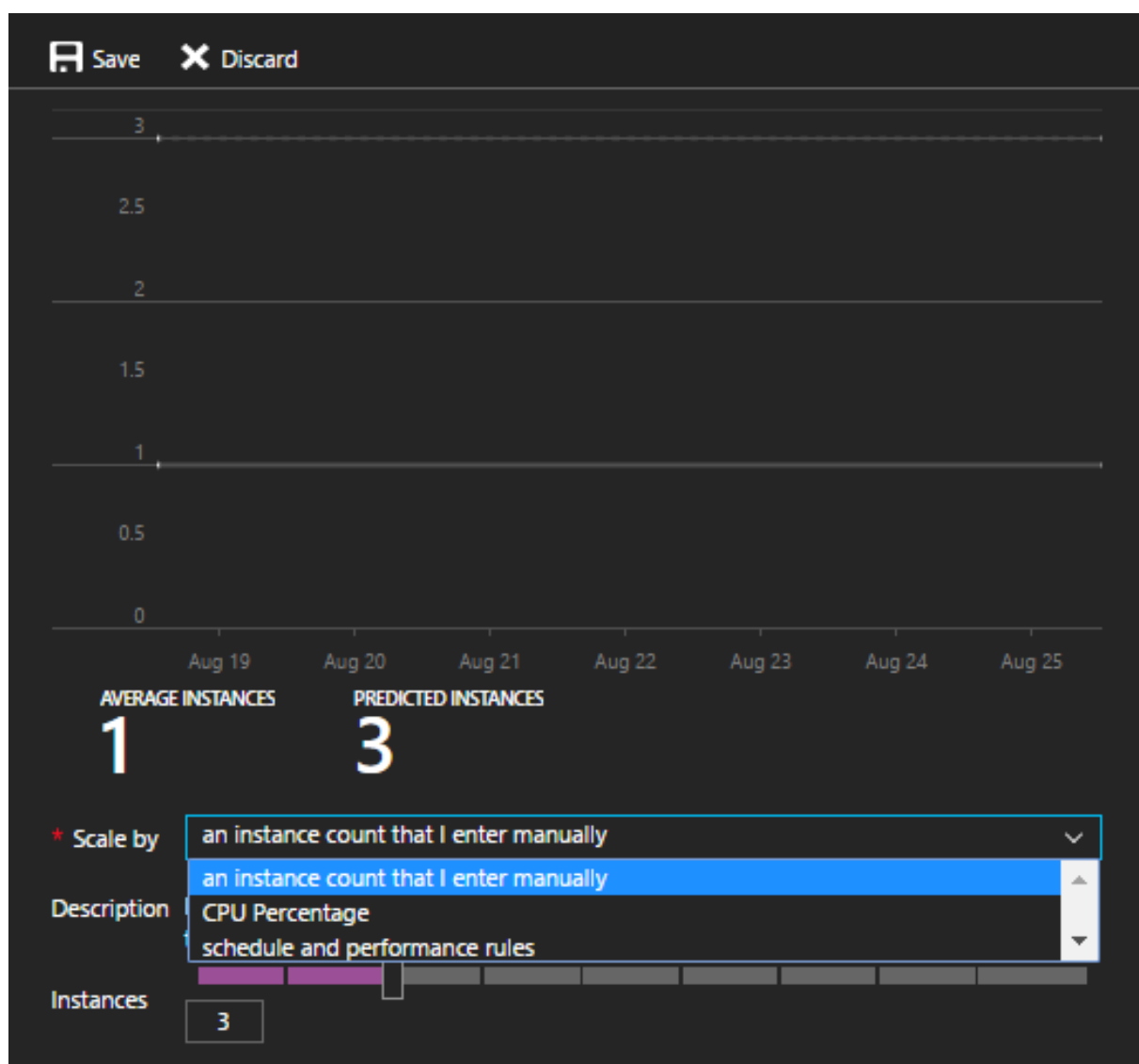
Slika 21 - Azure cjenovni rang Internet aplikacija

Dodatna mogućnost koju Azure nudi kod izrade skalabilnih aplikacija jest mogućnost skaliranja dodatnih instanci aplikacije. Ova mogućnost nalazi se pod *Scale out* resursom na administracijskom sučelju aplikacije. Za potrebe ovog primjera,

odabran je Standardni servisni plan koji ima mogućnost skaliranja za do 10 instanci po aplikaciji. Aplikacije je moguće skalirati na jedan od sljedeća 3 načina

- ▶ Po broju instanci koje se ručno odaberu
- ▶ Po postotku zauzeća procesora
- ▶ Po vlastitim definiranim pravilima

Na administracijskom sučelju ovo izgleda ovako



Slika 22 Azure, skaliranje instanci

Primjerice, ukoliko se odabere postotak zauzeća procesora moguće je definirati sljedeće pravilo. Na administracijskom sučelju moguće je odabrati broj instanci aplikacija za koje želimo skalirati. Ovo pravilo izvršit će se kada dođe do prekoračenja vrijednosti koje smo zadali. Nadalje, potrebno je odabrati raspon zauzeća procesora u postocima za koji želimo da ova pravila vrijede.

Average Instances

1

* Scale by CPU Percentage

Description Automatically scale up or down based on CPU Percentage. Choose an average value you want to target.

Instances 2 10 35

Target range 85 85

Notifications for Scale Actions

☒ Email Administrator and CoAdministrators

Additional email(s) danijelhdev@gmail.com

Webhook ☐ HTTP or HTTPS endpoint to route autoscale notifications to.

[Learn more about configuring webhooks for autoscale notifications](#)

Slika 23 - Azure, skaliranje internet aplikacije cpu

Sa vrijednostima odabranim kao na slici 23, aplikacija će se skalirati na više instanci ukoliko prosječno zauzeće procesora bude veće od vrijednosti koje smo definirali pravilom. U ovom slučaju ta vrijednost iznosi 85%. Nadalje, skaliranje na manje instanci dogodit će se ukoliko prosječno zauzeće procesora padne ispod 35%. Važno je napomenuti da se prosječna vrijednost zauzeća procesora mjeri kroz sve instance

aplikacije. Opcija za automatsko skaliranje aplikacija svakih par minuta provjerava da li prosječna vrijednost instanci aplikacije spada unutar raspona koji smo mu zadali. Ukoliko ne, automatski će prilagoditi aplikaciju prema zadanim pravilima. Na ovaj način ukoliko se dogodi da na Internet aplikaciji poraste broj korisnika te se aplikacija sporije odaziva na zahtjeve korisnika, *Azure* će se sam pobrinuti kako bi performanse ostale dosljedne.

Osim skaliranja Internet aplikacija koristeći direktno *Microsoft Azure* i mogućnosti *Web App* servisa, bilo povećavajući cjenovni rang i mogućnosti aplikacije ili širenjem broja instanci aplikacije, postoji još par koraka u razvoju aplikacije koji se mogu poduzeti kako bi se omogućio nesmetan rad.

- ▶ Zaključavanje kod baza podataka(engl. Storage lock)
- ▶ *Caching*
- ▶ Asinkrone operacije(engl. Asynchronous operation)
- ▶ Redudancija(engl. Redundancy)

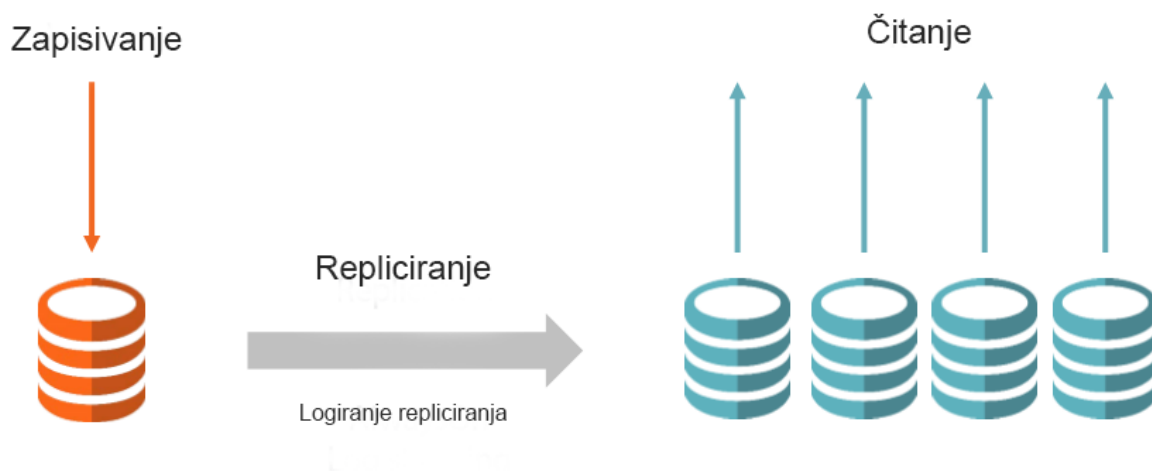
3.1. Zaključavanje kod baza podataka

Gotovo sve Internet aplikacije koriste neki od permanentnih načina pohrane podataka. Relacijske baze podataka jedan su od najčešćih. Relacijske baze podataka zasnivaju se na spremanju kolekcije podataka organiziranih u strukturiranim tablicama iz kojih se može pristupiti podacima na više različitih načina, bez potrebe za reorganizacijom tablica²⁹. Baze podataka imaju značajan utjecaj na performanse aplikacija. Potreba za spremanjem permanentnih podataka očita je. Ako se na primjer korisnik odjavi sa neke aplikacije, očekuje se da se kod sljedeće prijave svi njegovi podaci nalaze na istom mjestu, kao da se nije ni odjavio. Baze podataka posjeduju 4 glavna svojstva. Valentnost(engl. Atomicity) svojstvo omogućava bazama podataka obavljanje više operacija kao zasebne, individualne operacije, koje mogu biti ili uspješne ili neuspješne. Ako operacija ne uspije, sve se operacije vraćaju na prethodno stanje prije prve operacije. Ovo svojstvo sprečava parcijalne promjene na

²⁹ Margaret Rouse, whatts, <http://searchsqlserver.techtarget.com/definition/relational-database> (25.08.2016)

bazama podataka kada dođe do greške. Drugo svojstvo jest Konzistentnost(engl. Consistency). Konzistentna baza podataka osigurava da će jedna transakcija ili promjena promijeniti podatke u bazama podataka sa jednog valjanog stanja u drugo. Sprečava nekonzistentne rezultate kao na primjer prikazivanje polovice podataka promijenjenim a drugu polovicu još nepromijenjenom. Treba imati na umu da validan podatak u ovom smislu ne znači semantički validan, nego validan sa strane jezika baze podataka. Izolacijsko(engl. Isolation) svojstvo osigurava da sa strane aplikacije, sve operacije koje se događaju, događaju se kao da je aplikacija jedini proces koji pristupa bazi podataka. U stvarnosti ima više procesa koji koriste, pišu i čitaju u bazu podataka, ali svaki od njih ponaša se kao da je jedini. Drugim riječima, baza podataka štiti pojedini proces od utjecaja drugi procesa, štiteći tako i podatke koji se dijele između više procesa. Izdržljivost (engl. Durability) posljednje je svojstvo a kod baza podataka koje omogućava da kada operacija ili transakcija baze podataka završi, ostat će zabilježena bez obzira ako se baza podataka ugasi, nestane napajanja ili slično. Ova 4 svojstva vrlo su korisna u mnogim scenarijima, ali glavni problem kod izgradnje skalabilnih sustava jest što se dva od ova četiri svojstva kose sa principima skalabilnosti. Ta dva svojstva jesu Konzistentnost i Izolacija. Ova dva svojstva kod baza podataka osiguravaju se najviše zaključavanjem tablica kod akcija pisanja i čitanja iz baze podataka. Kod pisanja u bazu podataka, proces koji piše zaključava pristup tablici za to vrijeme. Time se osigurava konzistentnost podataka ali se ujedno i blokira pristup drugim procesima dok trenutni proces ne završi. Kada se koristi zaključavanje, samo jedan proces može promijeniti podatke u jedinici vremena. Na nekim razinama, ponekad je blokirano i čitanje. Ovo je prvi problem koji se može javiti vezano uz performanse, a može se dogoditi kada imamo čitanje i pisanje iz iste tablice u bazi podataka u istom vremenu. Kada se dogodi da se izdaje operacija za čitanjem i pisanjem u bazu istovremenom, postoji više scenarija koji se mogu dogoditi. To određuju izolacijske razine. Transakcijska razina sa najvećom izolacijom zove se *Serializable*. Kod ove razine, sve operacije izvode se jedna za drugom bez ikakvih paralelnih akcija. *Repeatable read* izolacijska razina zaključava čitanje i pisanje tijekom jedne transakcije. Ovo se radi na način da se zaključaju samo redovi koji su vezani uz sljedeću operaciju, što znači da je istovremen pristup toj tablici dopušten, ali ne i retku koji je u toku transakcije. Fantomski upiti su međutim dopušteni, kao na primjer odabir broja redaka unutar tablice. *Read committed* izolacijska razina omogućava čitanje samo onih podataka koji su potvrđeni ali ne i završeni, što znači da se njome mogu

čitati ne potvrđeni podaci ili podaci u tijeku transakcije. Zapisivanje u bazu je međutim još uvijek dopušteno ali bez paralelnih akcija. *Read uncommitted* dopušta čitanje i pisanje podataka u bilo kojem trenutku. Mogu se čitati ne potvrđeni podaci i time se dobiva maksimalna paralelnost. Jedino ograničenje jest da se ne može zapisivati isti podatak u isto vrijeme. Izolacijske razine utječu na broj paralelnih akcija koje se mogu odvijati na bazi podataka, pa samim time utječu i na performanse aplikacija. Međutim, nepravilno korištenje razina može dovesti do narušavanja integriteta podataka unutar baze podataka, pa je samim time preporučljivo koristiti razine sa oprezom. Jedno od rješenja jest također razdvajanje izvora podataka za pisanje i čitanje. Jedan od najčešćih pristupa sprečavanju zaključavanja tablica jest replikacija baza. Ovo se radi na način da se kopira glavna ili *master* baza na više kopija tj. replikacija u čestim vremenskim intervalima. U tom slučaju, glavna baza može se koristiti za zapisivanje u bazu podataka dok se njezine replikacije koriste za čitanje i pristup podacima. Model podataka tako ostaje isti, ali se tablice ne zaključavaju pošto je izvor podataka različit. Važno je napomenuti kako je potrebno osigurati što je kraći mogući period replikacije kako bi se minimizirala razlika između svježih i repliciranih podataka.



Slika 24 - SQL, repliciranje baze podataka

Moguće je također i koristiti različite modele za zapisivanje, a različite za čitanje podataka. Na ovaj način, model za zapisivanje podataka biti će optimiziran za serijske transakcije, koje se izvode jedna za drugom dok će model za čitanje iz baze biti

optimiziran za vrlo brza čitanja podataka. Ova specifična arhitektura poznata je pod imenom *CQRS*³⁰ (engl. Common query responsibility segregation). U relacijskim bazama podataka, relacije između modela podataka jako su bitne. Veze između modela obično se povezuju sa stranim ključem kako bi se pohranila veza. U tom slučaju, potrebna je samo jedna dodatna kolona u tablici kako bi se zapisala veza između dva entiteta. Veze osiguravaju da se podatak o entitetu sprema samo jedanput u svojoj vlastitoj tablici, nema ponavljanja istih vrijednosti što je pogodno za konzistentnost baze. Međutim, ono što je pogodno za konzistentnost baze loše je za performanse upita i čitanja iz baze. Naime, ako želimo dohvatiti podatke iz pojedinog entiteta zajedno sa entitetom na koji je vezan, potrebno je napraviti spajanje(engl. Join) tih dviju tablica. Spajanje tablica sa strane performansi jedna je od skupljih operacija. Kada pričamo o visokoj skalabilnosti, čin spajanja dviju ili više tablica između više zahtjeva na aplikaciju podiže više zaključavanja tablica i zapisa za vrijeme trajanja upita. Alternativa razdvajanju podataka jest spremanje vezanih podataka unutar jedne iste tablice. Taj proces zove se denormalizacija baze podataka. Denormalizacija zamjenjuje konzistentnost baze podataka za smanjeno vrijeme upita tj. čitanja iz baze podataka. Na ovaj način, podaci se dupliciraju ali je kod čitanja potreban jedan upit i jedna tablica kod dohvaćanja zapisa. Na ovaj način smanjili smo konzistentnost baze i uveli ponavljanje podataka kako bismo dobili brža vremena upita. Denormalizirane tablice preporučeno je koristiti kod modela za čitanje podataka. Međutim, ako nam već nisu potrebne relacije kod modela za čitanje podataka, možemo otići i korak dalje te iskoristiti prednosti ne relacijskih baza podataka. Na *Microsoft Azure* platformi postoje više opcija vezane uz ne relacijske baze podataka. Jedna od takvih opcija jest *Azure Table storage*. Takav način pohrane podataka predstavljen je glavnim ključem retka(engl. Row key) i particijskim ključem (engl. Partition key). Scenarij kada bi se trebalo razmisliti o uvođenju *Azure Table storage* opcije jest kada broj podataka unutar tablice naraste do te mjere da rezultat čitanja iz baze podataka jednostavno postane prespor. Dodatan razlog tome jest što je manipuliranje podataka u ogromnim tablicama spor proces. Često je slučaj i da se unutar tablice koja sadrži milijune zapisa, nalaze i podaci koji više nisu aktivni ali se ipak iz određenih razloga moraju sačuvati. Rješenje ovog problema zove se particioniranje(engl. Partition ili database shard) tablica. Particioniranje tablica radi na način da razdvaja tablicu po redovima u zasebne baze

³⁰ CQRS - <https://msdn.microsoft.com/en-us/library/dn568103.aspx>

podataka ovisno o particijskom ključu. To znači da će svi podaci sa jednakim ključem ići će u jednaku bazu. *Microsoft Azure* pruža i mogućnost skaliranja relacijskih baza podataka pomoću alata za elastične baze podataka(engl. Elastic Database tools³¹).

3.2. Caching

Caching jest jedan od najjednostavnijih poboljšanja koje se mogu uvesti kako bi se performanse aplikacija dodatno ubrzale. *Cache* je vrlo jednostavan ali efikasan mehanizam kojim se mogu uštedjeti nepotrebni pristupi bazi podataka. Loša strana ovog mehanizma jest što se uvodi dodatna mogućnost ustajalih podataka. Princip je vrlo jednostavan. Nakon što se svježi podaci iščitaju iz baze podataka, spremaju se u obližnji sustav za pohranu, najčešće memoriju. U budućim zahtjevima na aplikaciji, podaci se poslužuju direktno iz tog sustava za pohranu tj. *cache*-a. Na ovaj način, aplikacija će pristupiti bazi podataka jedino u slučaju ako ne postoje adekvatni podaci u *Cache* sustavu. Efikasno korištenje *Cache* mehanizma može uštedjeti ogroman broj pristupa bazi podataka. Jezgra *Caching* mehanizma jest minimiziranje količine posla koju server mora obaviti, pošto je poznato kako su pristupi bazi podataka najskuplje operacije. Kod implementacije, potrebno je voditi računa o ustajalim podacima. Ustajali podaci jesu podaci koji više ne predstavljaju stvarno stanje iz baze podataka. To znači da je moguće da se dogodi da su se podaci u bazi podataka u međuvremenu promijenili, dok su u *Cache* sustavu još uvijek ostali podaci iz prethodnog stanja. Ukoliko rješenje ili aplikacija koja se razvija ne zahtjeva podatke u stvarnom vremenu, može se izaći na kraj sa ustajalim podacima podešavajući vrijeme isteka *Cache* podataka(engl. Cache Expiration time). Vrijeme isteka predstavlja životni vijek podatka unutar *Cache* sustava. Kada vrijeme istekne, podatak se uklanja iz *Cache* memorije tako da sljedeći zahtjev koji dolazi na aplikaciju neće pronaći traženi podatak u zapisan u *Cache*-u. Aplikacija bi tada trebala dohvatiti svježije podatke iz baze podataka te ih ponovno zapisati u *Cache* memoriju, koji će postojati tamo dok im valjanost ponovno ne istekne. Vrijeme isteka trebalo bi biti dovoljno dugo da se dobije korist od ne dohvaćanja istih podataka iz baze podataka, ali i dovoljno kratko da se minimizira

³¹ Elastic database tools - <https://azure.microsoft.com/en-us/documentation/articles/sql-database-elastic-scale-get-started/>

ustajalost podataka. Ne postoji univerzalan recept za određivanje vremena isteka, već je to potrebno odrediti individualno za aplikaciju i njezine potrebe i scenarije. Ono što je međutim sigurno jest da bi vrijeme isteka trebalo biti što kraće za podatke koji se često mijenjaju, a po potrebi duže za statičke podatke koji se rijetko mijenjaju. Ukoliko mehanizam za *Cache* ne podržava automatsko čitanje i pisanje podataka na osnovu izvora ili u ovom slučaju baze podataka, postoji *Cache-Aside* uzorak koji je moguće iskoristiti za efektivno osvježavanje podataka u *Cache* memoriji. Scenarij ovog pristupa je sljedeći. Ukoliko kod zahtjeva na aplikaciju, podatak ne postoji u *Cache* memoriji, taj podatak se tada zapisuje kako bi bio dostupan kod sljedećeg zahtjeva. Kako bi se spriječilo dohvaćanje podataka koji su se u međuvremenu promijenili u bazi podataka, kod osvježavanja podatka potrebno je provjeriti da li taj podatak postoji u *Cache* memoriji te ako da, prisilno ga ukloniti. Postoje dvije vrste *Cache* mehanizma

- ▶ Lokalni(engl. Local cache)
- ▶ Distribuirani(engl. Distributed cache)

Lokalni mehanizam najčešći je i najjednostavniji oblik skladištenja podataka. Njegovo je svojstvo što je jedinstven za svaku instancu aplikacije tj. servera, pošto se podaci pohranjuju direktno u memoriju servera. To znači da se njegov sadržaj osvježava i čita samo ako se pristupa toj instanci aplikacije. Lokalni mehanizam predstavlja najbrži način skladištenja podataka, ali je iz razloga što se sprema u memoriju instance servera manje pogodan za skalabilne aplikacije. Iz tog razloga, pomoću lokalnog *Cache* mehanizma trebali bi se spremati samo podaci koji se jako rijetko osvježavaju. Distribuirani sustav s druge je strane mehanizam koji je zajednički za sve instance aplikacije. Prava vrijednost iz izrade skalabilnih sustava i korištenja *Cache* mehanizma dolazi ako se uspije implementirati za dinamičke podatke i podatke koji se često mijenjaju, na primjer svakih nekoliko minuta. Takve podatke vrijedi također spremati u *Cache* memoriju, jer ako na primjer aplikaciji pristupa tisuću korisnika unutar te minute, uštedjeli smo jednak taj broj pristupa bazi podataka u tom vremenskom periodu. Još jedna od prednosti korištenja distribuiranih sustava za skladištenje podataka jest skalabilnost koju ono može pružiti. Mnogi sustavi za distribuirano skladištenje podataka koriste *clustering*³². Instanca aplikacije pošalje zahtjev na distribuirani sustav za skladištenje, a infrastruktura ispod tog sustava zadužena je za određivanje lokacije

³² Clustering - <https://en.wikipedia.org/wiki/Clustering>

podataka unutar *cluster*-a. *Cache* se lako može skalirati dodajući više servera ili povećavajući otpornost i dostupnost. Negativna strana distribuiranog sustava za skladištenje podataka jest sporiji pristup, pošto se ne nalazi u lokalnoj memoriji instance servera. Međutim, takav *Cache* iako je sporiji od lokalnog i dalje je puno brži od pristupanja bazi podataka. *Microsoft Azure* pruža tri mogućnosti za implementaciju *Cache* mehanizma.

- ▶ *Azure In-Role Cache*
- ▶ *Azure Managed Cache*
- ▶ *Azure Redis*³³ *Cache*

Međutim, preporučena opcija za implementaciju *Cache* mehanizma jest *Azure Redis distributed cache*.

3.3. Asinkrone operacije

Kao što je već spomenuto, sinkrone radnje kod izgradnje skalabilnih sustava mogu imati krajnje negativan utjecaj na performanse aplikacija. Koristeći sinkrone operacije, najčešće se blokira neki drugi dio operacija. Poziv iz koda aplikacije na neki drugi dio aplikacije, najčešće nije poziv koji blokira određenu radnju. Međutim, velika većina koda u interakciji je sa operativnim sustavom, lokalnim ili udaljenim servisima. Svaki puta kada se kroz kod otvara datoteka, čitaju podaci sa *Rest* servisa ili se spremaju podaci u bazu podataka, postoji određen vremenski period kada se čeka na odgovor akcije. Kada se neka akcija pokrene sinkrono, prije nego se nastavi dalje mora se pričekati da ta akcija završi. Kada se akcija pokrene asinkrono, ukoliko postoji period u kojem je nit(engl. Thread³⁴) u stanju čekanja, drugi zadatak(engl. Task) može nastaviti rad sa svojom operacijom. Asinkroni poziv može se podijeliti na dva dijela, na početni zahtjev(engl. Initial request) i odgovor(engl. Callback). Početni zahtjev prosljeđuje se komponentni zajedno za povratnom adresom za odgovor. Na ovaj način, daje se do znanja komponentni gdje da se dostavi rezultat operacije. Kada je rezultat spreman, *Callback* pokreće event i nit koja je započela radnju čita rezultat i

³³ Azure redis cache - <https://azure.microsoft.com/en-us/services/cache/>

³⁴ Thread - [https://en.wikipedia.org/wiki/Thread_\(computing\)](https://en.wikipedia.org/wiki/Thread_(computing))

nastavlja dalje svojim tokom. Za vrijeme čekanja na odgovor od komponente, nit je u stanju čekanja i u mogućnosti je obraditi neki drugi zahtjev ili operaciju. Iako su asinkroni pozivi od velike koristi, također imaju svoje negativne strane. Asinkron kod je veće složenosti i teže ga je razumjeti od sinkronoga. Mehanizmi za asinkrone pozive također zahtijevaju dodatnu memoriju kako bi pohranili odgovore tj. *callbacks*. Asinkronost dolazi najviše do izražaja kada operaciju koju je potrebno obaviti obavlja neka druga, vanjska komponenta kao što je to operativni sustav ili baza podataka.

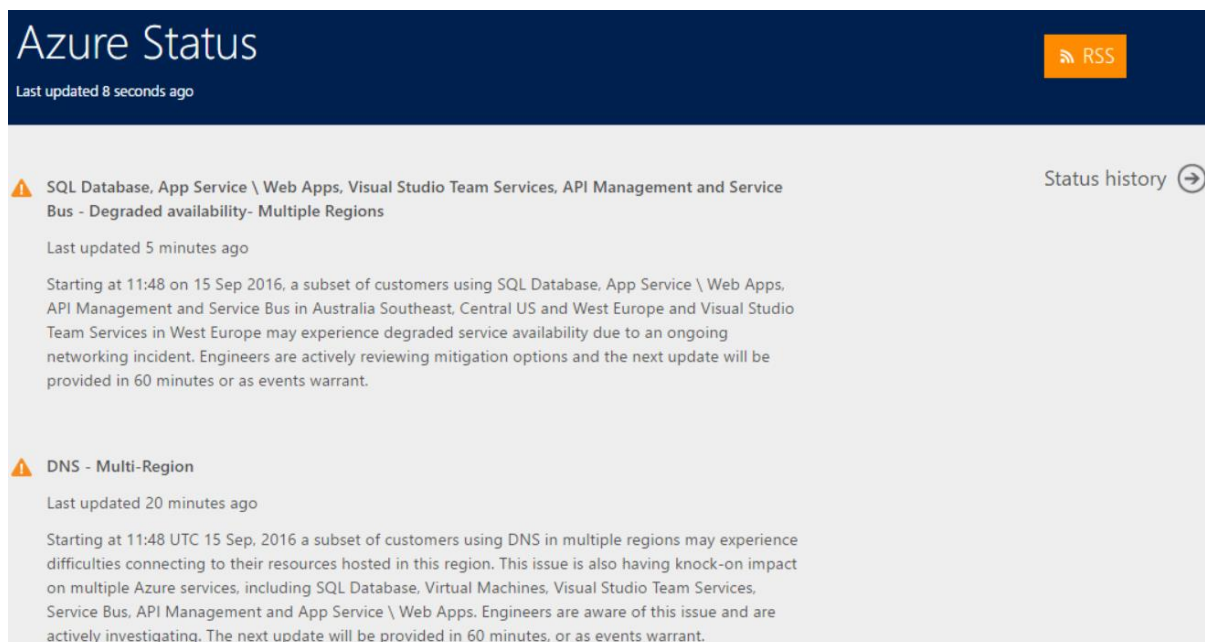
3.4. Redundancija

Još jedan od elemenata skalabilnih sustava jest koncept redundancije i otpornosti na greške. Otpornost na greške u aplikacijama postiže se upravo dizajniranjem dijelova aplikacije kao zasebne redundantne cjeline eliminirajući tako jedinstvenu točku(engl. Single point of failure) pogreške u aplikaciji. Jedinstvena točka pogreške ili iznimke odnosi se na situaciju kada bilo koja komponenta unutar aplikacije ili sustava uzrokuje pad ili nepredviđeno ponašanje aplikacije. Ovakva greška također podrazumijeva da ne postoji alternativa kada se dogodi ovakva greška. Kako bismo uspješno obradili ovakve tipove pogrešaka, aplikaciju je potrebno promijeniti i dizajnirati na način da može nastaviti raditi čak i sa pogreškama. Potrebno je napomenuti i da izrada aplikacija da budu otporne na greške dodaje još više složenosti na postojeću arhitekturu. Pogreške bismo mogli podijeliti na dva tipa

- ▶ Tranzijentne(engl. Transient)
- ▶ Dosljedne(engl. Persistent)

Dosljedne pogreške odnose se na greške u aplikaciji koje sama aplikacija ne može prebroditi. Primjer takve pogreške bila bi neka od prirodnih katastrofa ili pogreška u održavanju, na primjer u podatkovnom centru gdje je aplikacija podignuta. Bilo koji od ovih scenarija može izazvati nedostupnost aplikacija. U ovakvim slučajevima, potrebno je osigurati dobre mehanizme za bilježenje događaja kako bi se brzo otkrio uzrok problema i minimiziralo vrijeme koje je aplikacija bila nedostupna. Kod ovakvih pogrešaka najčešće je potrebna ljudska intervencija te se ne mogu spriječiti samim dizajnom arhitekture aplikacije.

Tranzijentne pogreške odnose se na greške u aplikaciji koje mogu biti uzorkovane privremenim gubitkom Internet ili neke druge veze ili opterećenjem servera. Ovakvi tipovi pogrešaka većinom se mogu samo zaliječiti što znači da se mogu zaobići bez ljudske intervencije, većinom ponovnim pokušajem spajanja ili prebacivanjem na alternativne izvore. Pogreške u aplikaciji mogu biti različitog podrijetla. Najjednostavnije od njih jesu mehaničkog podrijetla. Otpornost na pogreške počinje samim hardverom. Memorijski čipovi i tvrdi diskovi imaju ugrađene mehanizme za otkrivanje i uklanjanje pogrešaka. Na *Microsoft Azure* platformi, većina mehaničkih kvarova može biti ispravljena pomoću redudancije tj. pomoću pokretanja više instanci aplikacije na više servera. Osim samog servera na kojem je pokrenuta, mnoge aplikacije ili servisi konzumiraju i vanjske servise. Primjer takvih servisa bio bi *Azure Table storage*, *Redis cache* mehanizam, DNS i mnogi drugi. Moguć scenarij jest i da jedan od vanjskih servisa bude nedostupan na neko vrijeme. U tom slučaju, potrebno je imati odgovarajuće mehanizme za bilježenje pogrešaka te korisniku prikazati ispravnu poruku. Moguće je postupiti i na način, na primjer u slučaju pada baze podataka, prikazati podatke iz *cache* mehanizma iako mogu biti potencijalno zastarjeli. Primjer jednog od takvih dana kada su bili zahvaćeni Azure servisi bio je i 15.09.2016.



The screenshot shows the Azure Status page with a dark blue header. The header contains the text "Azure Status" and "Last updated 8 seconds ago". There is an orange RSS button in the top right corner. The main content area is light gray and contains two sections of service status updates, each preceded by a yellow warning icon. The first section is titled "SQL Database, App Service \ Web Apps, Visual Studio Team Services, API Management and Service Bus - Degraded availability- Multiple Regions" and was last updated 5 minutes ago. It describes a networking incident starting at 11:48 on 15 Sep 2016 affecting a subset of customers in Australia Southeast, Central US, and West Europe. The second section is titled "DNS - Multi-Region" and was last updated 20 minutes ago. It describes difficulties connecting to DNS resources in multiple regions, also starting at 11:48 UTC 15 Sep, 2016, and affecting multiple Azure services. A "Status history" link with a right arrow icon is located to the right of the first section.

Azure Status
Last updated 8 seconds ago

SQL Database, App Service \ Web Apps, Visual Studio Team Services, API Management and Service Bus - Degraded availability- Multiple Regions
Last updated 5 minutes ago

Starting at 11:48 on 15 Sep 2016, a subset of customers using SQL Database, App Service \ Web Apps, API Management and Service Bus in Australia Southeast, Central US and West Europe and Visual Studio Team Services in West Europe may experience degraded service availability due to an ongoing networking incident. Engineers are actively reviewing mitigation options and the next update will be provided in 60 minutes or as events warrant.

DNS - Multi-Region
Last updated 20 minutes ago

Starting at 11:48 UTC 15 Sep, 2016 a subset of customers using DNS in multiple regions may experience difficulties connecting to their resources hosted in this region. This issue is also having knock-on impact on multiple Azure services, including SQL Database, Virtual Machines, Visual Studio Team Services, Service Bus, API Management and App Service \ Web Apps. Engineers are aware of this issue and are actively investigating. The next update will be provided in 60 minutes, or as events warrant.

Status history →

Slika 25 - Microsoft Azure status dostupnosti

Tranzijetne pogreške kao što je spor odaziv ili gubitak povezanosti mogu se dogoditi u bilo kojem trenutku. Kao što je već spomenuto, najčešće rješenje jest predvidjeti takav tip pogreške te prikazati korisniku odgovarajuću poruku. Međutim, većina takvih pogrešaka može se ublažiti ili razriješiti jednostavno ponovnim izvođenjem operacije i pristupanjem servisu. Logika iza ovoga jest da je ponekad bolje pokušati ponovno uspostaviti vezu i dohvatiti podatke, imajući tako sporiji odaziv aplikacije, nego direktno prikazati korisniku poruku o pogrešci. U većini slučajeva, ponovni pokušaj će biti uspješan i korisnik neće biti svjestan da se u pozadini uopće dogodio problem. Međutim, moguć je i scenarij da i nakon nekoliko pokušaja, vanjski servis se i dalje ne odaziva. Ne rijetko je slučaj i da isti servis poslužuje više aplikacija, koje bi u ovom scenariju sve pokušavale ponovno uspostaviti vezu istovremeno. Zajedničko opterećenje od previše korisnika koji pokušavaju ponovno uspostaviti vezu sa već nedostupnim servisom može dodatno degradirati njegove performanse. Šaljući ponovne zahtjeve prečesto, aplikacije mogu ograničiti dostupne niti servera te onemogućiti odaziv servera na zahtjeve koji bi inače bili uspješni. Jedna od najbolji praksa kod korištenja politike ponovnog pokušaja(engl. Retry policy) jest korištenje eksponencijalnog odstupanja.



Slika 26 - Eksponencijalna politika ponovnog pokušaja

Zasniva se na eksponencijalnom povećanju vremena između svakog ponovnog pokušaja. Tako na primjer, ukoliko je od prošlog pokušaja prošlo 2 sekunde, do sljedećeg će proći 4. Sljedeći nakon toga bit će još više udaljen tj. za 8 sekundi. Čekajući više i više između ponovnih pokušaja, može se smanjiti trenutno opterećenje sa servisa. Ovo se naravno ne može nastaviti unedogled, pošto korisnici čekaju na odaziv aplikacije.

Postoji nekoliko dostupnih mehanizma za implementiranje politike ponovnog pokušaja unutar .NET framework-a.

- ▶ Azure klijent za pohranu(engl. Azure storage client³⁵)
- ▶ Biblioteka za rukovanje tranzijentnim pogreškama(engl. Enterprise library transient fault handling block³⁶)
- ▶ Polly³⁷

Način kako postići otpornost na greške jest čineći sve komponente redundantnima. Najjednostavniji način postizanja redundancije jest postavljanje aplikacija na više instanci i servera unutar pojedinog podatkovnog centra. Tako na primjer na *Microsoft Azure* platformi, moguće je odabrati broj instanci internet aplikacije. *Microsoft Azure* ne samo da dozvoljava broj instanci aplikacije, već je moguće kontrolirati i nivo redundancije ostalih *Azure* servisa. Tako na primjer *Azure storage* dozvoljava korištenje lokalne, zonske ili geografske redundancije. *Azure Redis cache* mehanizam podržava replikaciju servisa tako da ako primarni mehanizam zataji, onaj u pozadini može preuzeti i samostalno izdržati pod opterećenjem korisnika. Drugi način bio bi postavljanje aplikacije na više različitih podatkovnih centara. Ako jedan od centara nije dostupan ili je neočekivano spor, moguće je prebaciti se na drugi. Ovo zahtjeva da se replicira kompletno postavljanje u drugom geografskom području, što je na *Microsoft Azure* platformi izrazito lako za postići. Da bi se ovo postiglo, potrebno je pogoniti

³⁵ Azure storage client - <https://msdn.microsoft.com/en-us/library/azure/mt347887.aspx>

³⁶ Enterprise transient fault handling - [https://msdn.microsoft.com/en-us/library/hh680934\(v=pandp.50\).aspx](https://msdn.microsoft.com/en-us/library/hh680934(v=pandp.50).aspx)

³⁷ Polly - <https://github.com/App-vNext/Polly>

servis za upravljanje prometom(engl. Traffic management service³⁸) . Ovaj servis omogućava kontrolu distribucije korisničkog prometa na određene krajnje adrese što može uključivati azure servise u oblaku, Internet aplikacije, stranice i druge servise.

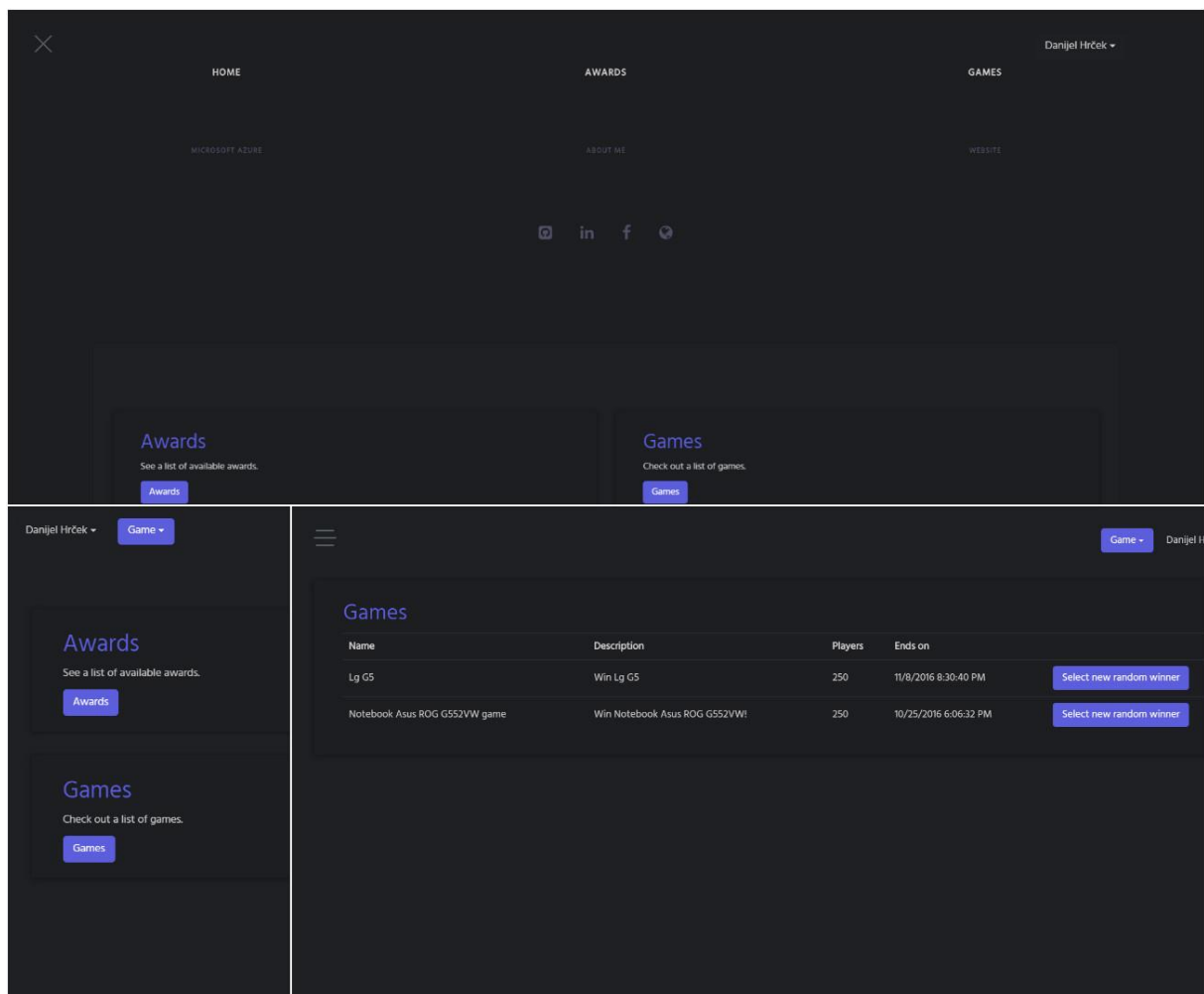
³⁸ Azure traffic manager - <https://azure.microsoft.com/en-us/documentation/articles/traffic-manager-overview/>

4. Izrada i testiranje skalabilne internet aplikacije

Za potrebe testiranja i izrade skalabilne Internet aplikacije, razvojnu platformu čine:

- ▶ Bootstrap 3.3.7 za prilagodbu sučelja
- ▶ ASP.Net MVC 5.2 za korisničko i administracijsko sučelje
- ▶ Entity Framework 6.1.3 za O-R mapiranje
- ▶ ASP.Net Framework 4.6.2 kao izvršna platforma
- ▶ Microsoft Azure SQL Server verzije 12(2014)
- ▶ Azure Redis Cache

Službeni jezik projekta je engleski. Svrha ogledne aplikacije jest testirati njezino ponašanje pod opterećenjem na *Microsoft Azure* platformi. Aplikacija je zamišljena kao neprekidna Internet nagradna igra u kojoj se stalno traži slučajni pobjednik. Sastoji se od 4 ekrana. Početni ekran služi kao navigacijski ekran koji vodi na sljedeća dva ekrana. To su ekran sa popisom nagrada i ekran sa popisom igara. Ekran sa popisom nagrada sadrži popis svih nagrada iz kojih se može kreirati nagradna igra. Za potrebe ovog testa, u bazu podataka unesene su dvije nagrade. Ekran sa popisom igara sadrži dvije kreirane igre iz prihodno navedenih dviju nagrada. Ekran također sadrži i gumb koji pokreće algoritam za slučajan odabir pobjednika, te prikazuje ekran sa dobitnikom. Za potrebe testa u bazu je uneseno 250 korisničkih računa među kojima jedan od njih postaje pobjednik. Primjer izgleda Internet aplikacije u nastavku.



Slika 27 - Izgled aplikacije

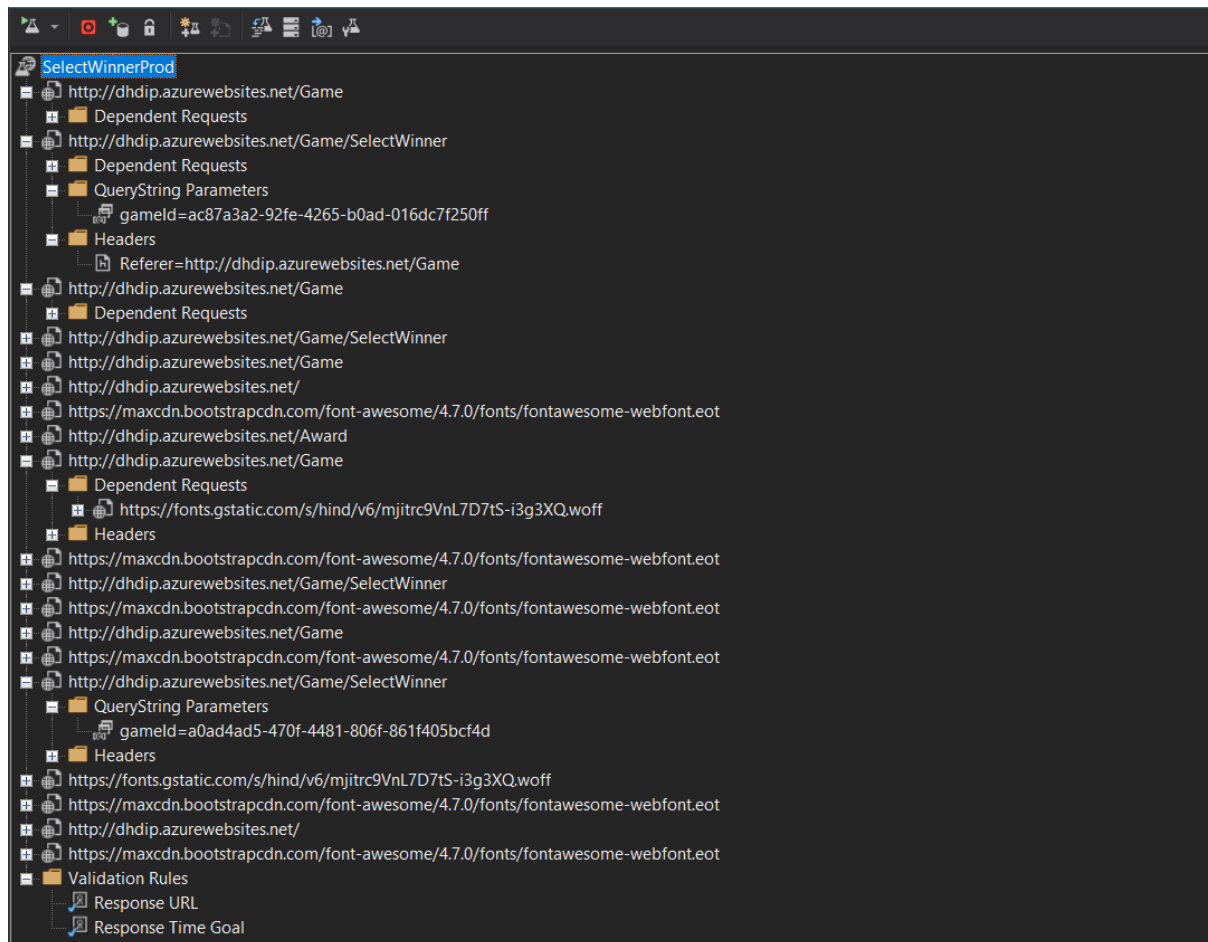
Za testiranje skalabilnosti i performansi Internet aplikacije koristi se *Microsoft Azure* resurs za testiranje performansi(engl. performance test). Test se može napraviti na dva načina.

- ▶ Korištenjem isključivo Microsoft Azure
- ▶ Korištenjem Visual studio testne datoteke

Prvi način jest koristeći isključivo *Microsoft Azure* i jednu Internet adresu koja će se koristiti za vrijeme testiranja kao jedinstvena točka pristupa. Drugi način jest koristeći *Visual studio*³⁹ datoteke za testiranje. Pomoću drugog načina moguće je simulirati

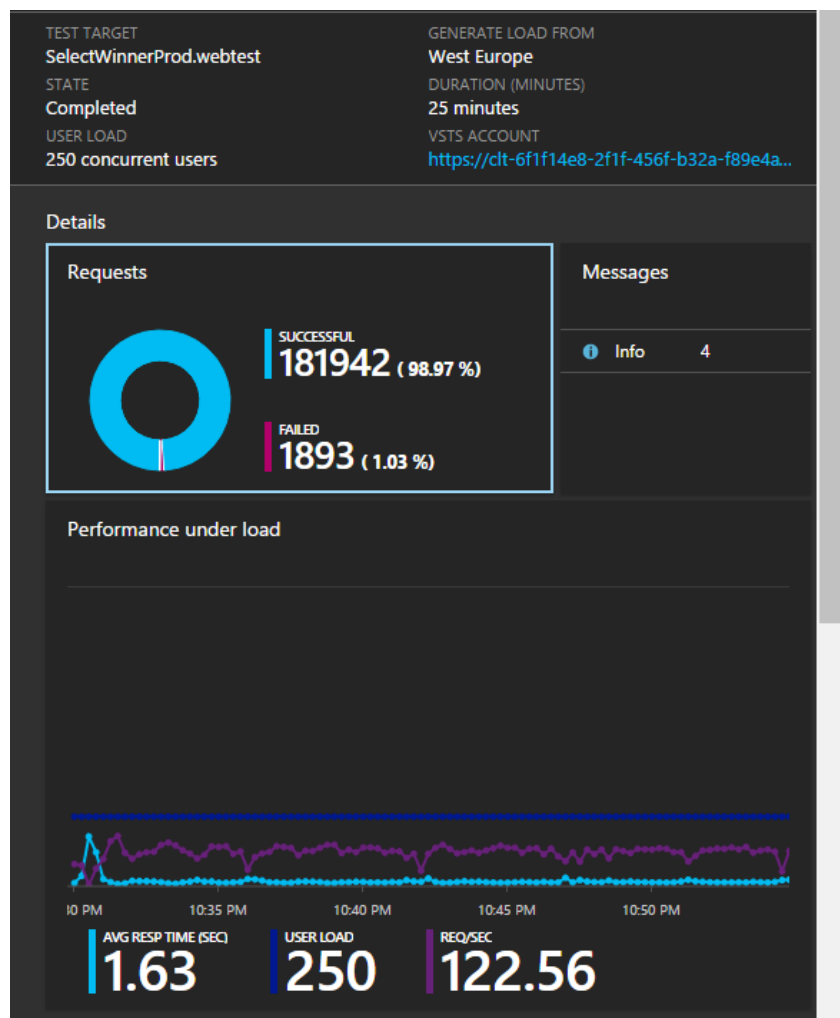
³⁹ Visual studio - <https://www.visualstudio.com/>

kompletno kretanje korisnika kroz aplikaciju. Za potrebe ovog testa koristi se drugi način. Kada se takav test zabilježi kroz *Visual studio*, to izgleda ovako

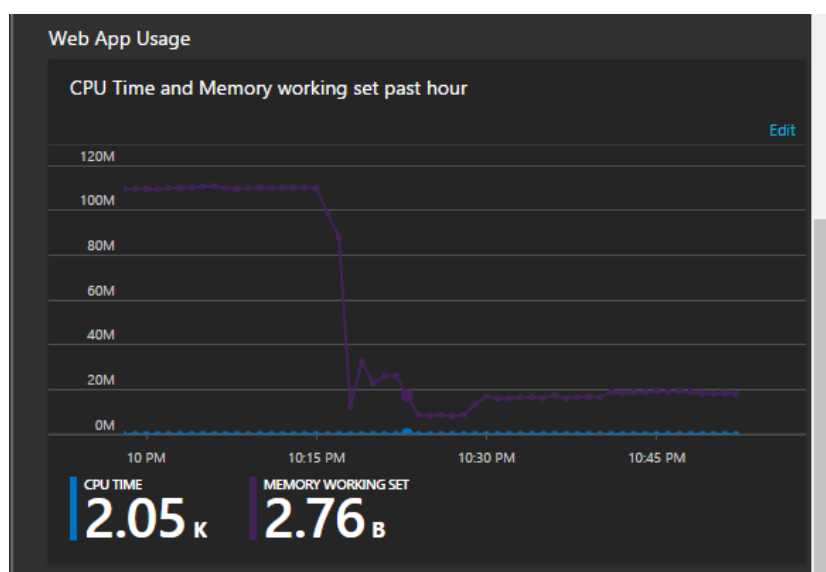


Slika 28 - Visual studio test

Ovaj test simulira ponašanje korisnika na oglednoj Internet aplikaciji. Na *Microsoft Azure* platformi kreiran je test sa iznad navedenom testnom datotekom, 250 istovremenih korisnika te u trajanju od 25 minuta. Koristeći samo jednu instancu aplikacije zabilježeni su sljedeći rezultati testiranja.



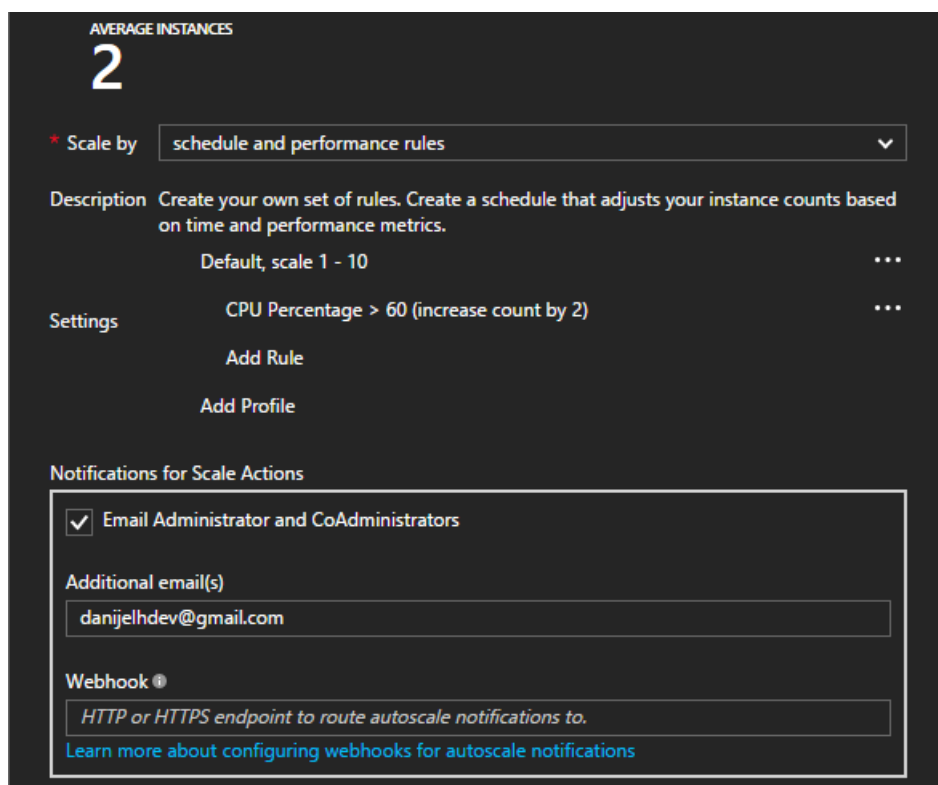
Slika 29 - Rezultati testa, skaliranje isključeno



Slika 30 - Rezultati testa, skaliranje isključeno

Test je završio sa ukupno 183 835 zahtjeva upućenih na Internet aplikaciji unutar 25 minuta. Od toga 181 942 zahtjeva bila su uspješna dok je ostalih 1893 rezultiralo nekakvom pogreškom. Prosječno to iznosi 122.56 zahtjeva u sekundi.

Drugi test izveden je pomoću uključene opcije za skaliranje i pomoću *Azure Redis* cache baze. Aplikacija je podešena da reagira na sljedeći način



AVERAGE INSTANCES

2

* Scale by schedule and performance rules

Description Create your own set of rules. Create a schedule that adjusts your instance counts based on time and performance metrics.

Default, scale 1 - 10

Settings CPU Percentage > 60 (increase count by 2)

Add Rule

Add Profile

Notifications for Scale Actions

☒ Email Administrator and CoAdministrators

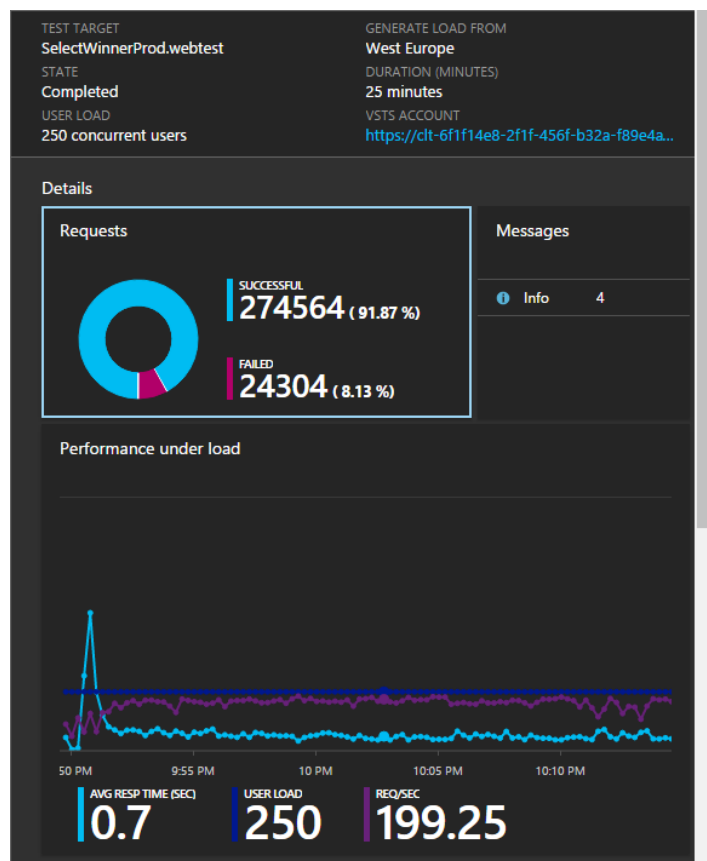
Additional email(s)

Webhook ⓘ

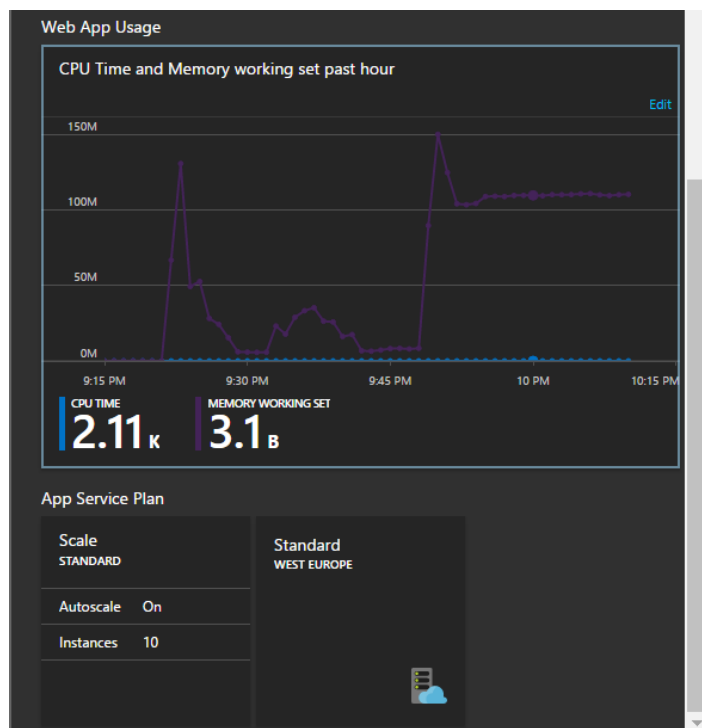
[Learn more about configuring webhooks for autoscale notifications](#)

Slika 31 - Postavke skaliranja

Aplikacija je postavljena da se automatski preraspodjeli na više instanci ukoliko prosječno opterećenje procesora bude veće od 60% unutar vremenskog perioda od 5 minuta. Kakav utjecaj ovo ima na rezultate ponovnog testa slijedi u nastavku.



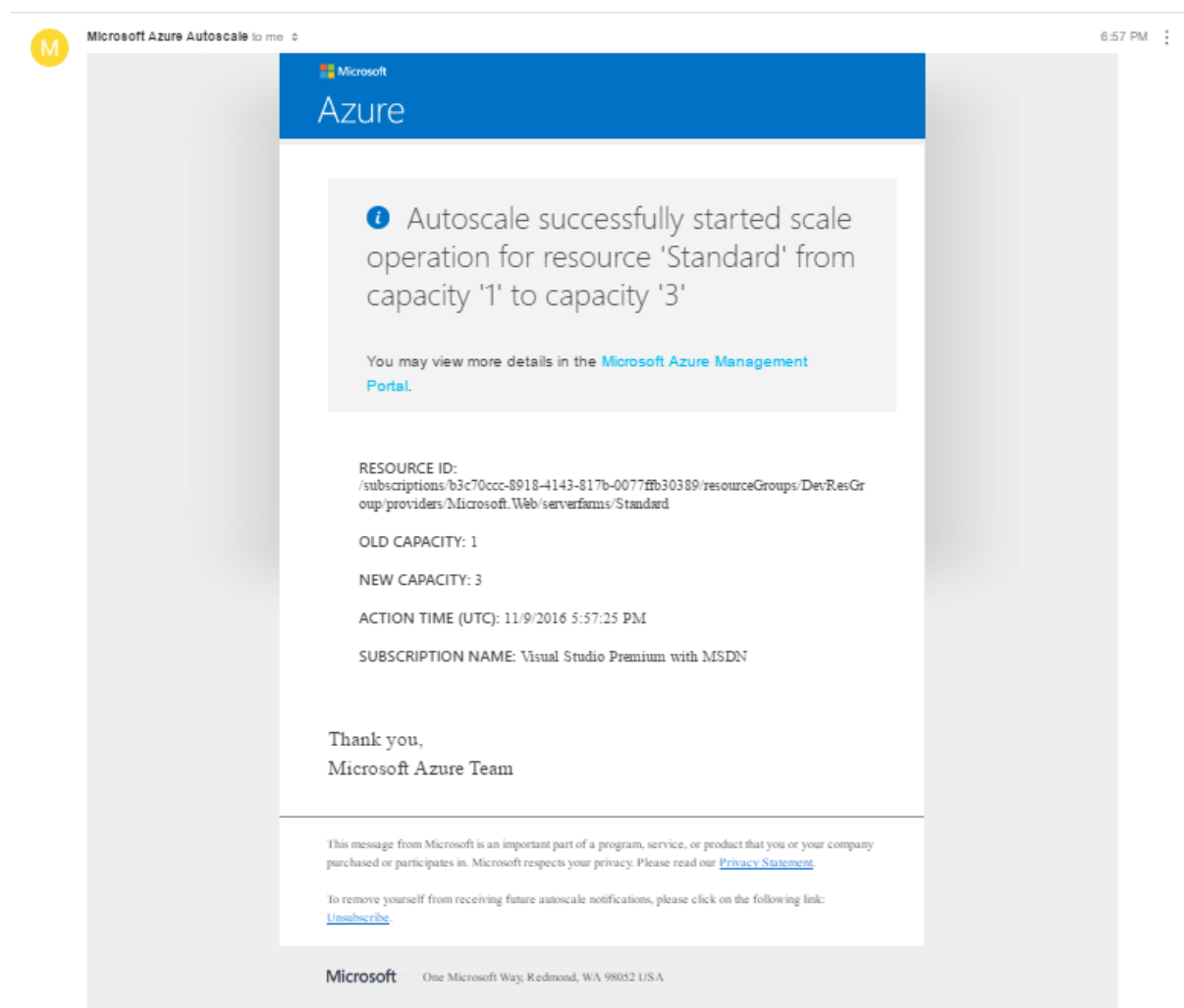
Slika 32 - Rezultati testa, skaliranje uključeno



Slika 33 - Rezultati testa, skaliranje uključeno

Test je završio sa ukupno 298 868 zahtjeva upućenih na Internet aplikaciji unutar 25 minuta. Od toga 274 564 zahtjeva bila su uspješna dok je ostalih 24 304 rezultiralo nekakvog pogreškom. Prosječno to iznosi 199.25 zahtjeva u sekundi. Ovdje se može primijetiti da je postotak neuspjelih zahtjeva veći nego u prijašnjom testu. Ovo se dogodilo iz razloga što je za potrebe ovog testa korištena jeftinija instanca *Microsoft Azure Redis cache-a* koja nije namijenjena da služi u bilo kakvom produkcijskom okruženju.

Da je skaliranje aplikacije bilo uspješno, osim rezultata to pokazuje i obavijest mailom.



Slika 34 - Skaliranje, potvrda o preraspodjeli instanci

Iz ovih rezultata moguće je zaključiti kako skaliranje aplikacija u oblaku koristeći *Microsoft Azure* platformu može uvelike olakšati upravljanje Internet aplikacijama i servisima u slučaju povećanja broja korisnika. Drugi test sa uključenim skaliranjem i *redis cache* optimizacijom uspješno je zaprimio 92 622 zahtjeva više i to sa odazivom u duplo kraćem vremenu. Sve što je bilo potrebno za postizanje ovih performansi jest definirati skup pravila koja određuju kako će se naša aplikacija ponašati pod opterećenjem.

5. Zaključak

Korištenjem rješenja u oblaku, klijenti, korisnici, razvojni inženjeri i drugi više ne moraju brinuti o ulaganju sredstava u izgradnju informacijskih sustava već istu iznajmljuju od pružatelja usluga. Na taj način fokus se prebacuje isključivo na razvoj rješenja bez potrebe za konstantnom izgradnjom i potporom hardverskih zahtjeva. U današnje vrijeme, mnoge uspješne Internet aplikacije i stranice dosegnu trenutak kada zbog povećanog broja korisnika performanse rješenja jednostavno padnu. Neki od simptoma mogu biti sporije učitavanje stranica, pucanje konekcija prema serveru i stalna visoka iskorištenost serverskih resursa.

U ovom radu prikazano je kako je koristeći *Microsoft Azure* platformu u oblaku aplikacije vrlo lako skalirati prema vlastitim zahtjevima i opterećenju bez ikakve potrebe za potporom, promjenom ili nadogradnjom hardvera. Izgradnja skalabilnih rješenja može unijeti dodatnu kompleksnost i apstrakciju u aplikacije, što čini dodavanje novih mogućnosti i testiranje koda dužim i kompliciranijim. Međutim, prednosti mogu biti znatne. Kroz rad prikazani su rezultati koji pokazuju kako je aplikacija koja je podešena da se prilagođava opterećenju na aplikaciji znatno fleksibilnija te može uspješno i u manjem vremenu obraditi znatno veći broj istovremenih zahtjeva.

U izgradnji vlastitih rješenja potrebno je odvagati prednosti unošenja dodatne složenosti i cijene u svrhu fleksibilnosti rješenja. *Microsoft Azure* tu pruža mnoge značajke te može svakako pomoći.

Ime Prezime

Literatura i izvori

1. Microsoft Azure - <https://azure.microsoft.com/>
2. Kyle Burns, Windows Azure Websites Succintly, 2014
3. Parikshit Asvjani, SQL on Azure, 2015
4. Haishi Bai, Zen of Cloud: Learning Cloud Computing by Examples on Microsoft Azure, 2014
5. Stack overflow, <http://stackoverflow.com/questions/240455/what-is-windows-azure>, 2016
6. Edin Kavic, Building Highly Scalable Web Applications in Azure, 2015
7. Michael S. Collier, Robin E. Shahan, Microsoft Azure Essentials: Fundamentals of Azure, Second Edition, 2016
8. Rick Rainey, Microsoft Azure Essentials: Azure Web Apps for Developers, 2015
9. Bob Familiar, Microservices, IoT and Azure, 2015
10. Henry Li, Introducing Windows Azure, 2009