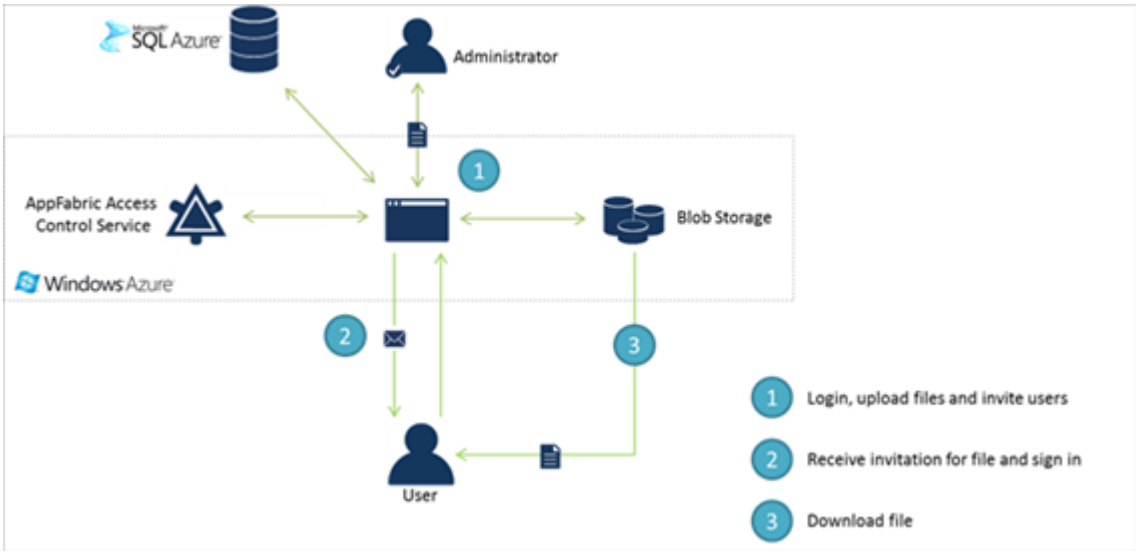
Arhitektura Windows Azure aplikacija

ZADATAK 1

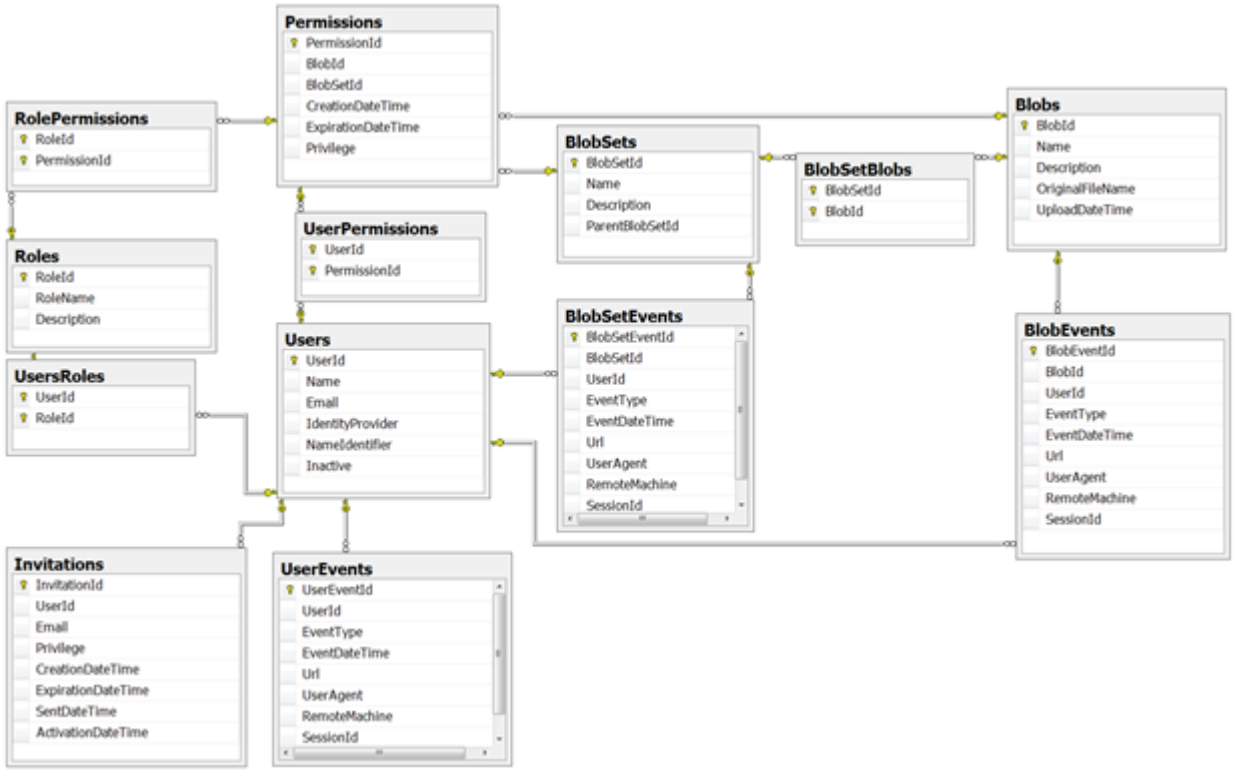
Potrebno je rešiti problem kontrolisanog deljenja fajlova na Internetu, pod kontrolom jednog administratora, koji ima punu kontrolu nad dodeljivanjem privilegija. Administrator upload-uje fajlove i deli ih sa korisnicima šaljući pozivnice elektronskom poštom. Administrator takođe dodeljuje uloge korisnicima, daje dozvole za pojedine korisničke naloge ili uloge, i ispituje kompletan trag revizije svih aktivnosti korisnika. Pozvani korisnici mogu se autentifikovati kod provajdera socijalnog identiteta kao što su Windows Live Id, Facebook, Google+, da bi dobili pristup deljenim fajlovima.

Rešenje

MVC aplikacija nalazi se iznad skladišta fajlova i služi za prihvatanje korisnika sa aktivnim korisničkim nalozima, koji se čuvaju u SQL Azure bazi podataka o korisničkim profilima i ulogama. Lokalni korisnički profil i uloga tog korisnika koriste se da utvrde da li taj korisnik ima pristup do određenog fajla koji traži. Svaki fajl treba da ima pojedinačnu URL adresu. Kada korisnik pokuša da dođe do jednog fajla, aplikacija će ga uputiti kroz proces provere autentičnosti i ako se ispostavi da korisnik nema dozvolu za određeni fajl, neće dobiti pristup tom fajlu.



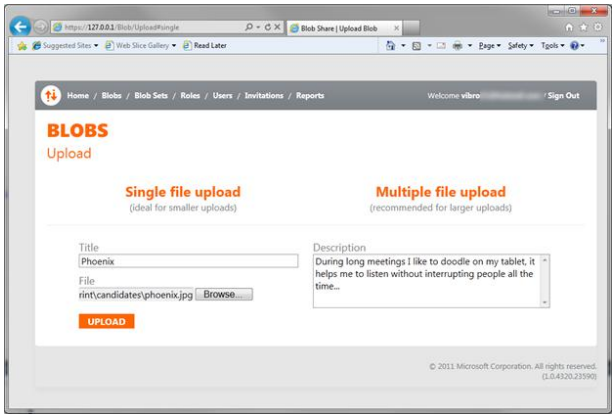
Model SQL Azure baze podataka dat je u nastavku:

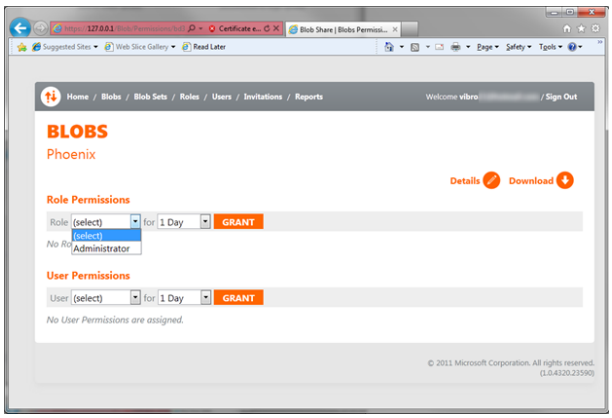


Kao što možete da primetite sva prava pristupa čuvaju se u bazi podataka. Kada se korisnik autentifikuje, aplikacija očekuje odgovor sa identifikatorom korisnika i identifikatorom provajdera. Ova dva podatka se koriste da bi se svaki korisnik jedinstveno identifikovao u bazi podataka ovog sistema. Ako dolazni par podataka IdentifikatorKorisnika-IdentifikatorProvajdera nije prisutan u bazi podataka, pristup neće biti dozvoljen.

Analizirajući kontrolu pristupa, postoje četiri vrste zahteva koje su interesantne za ispitivanje:

- Dodavanje novog fajla, od strane administratora
- Proces prijavljivanja novog korisnika (koji je dobio pozivnicu mejlom)
- Korisnik pristupa URL adresi koja ukazuje na fajl
- Korisnik se prijavljuje na sistem



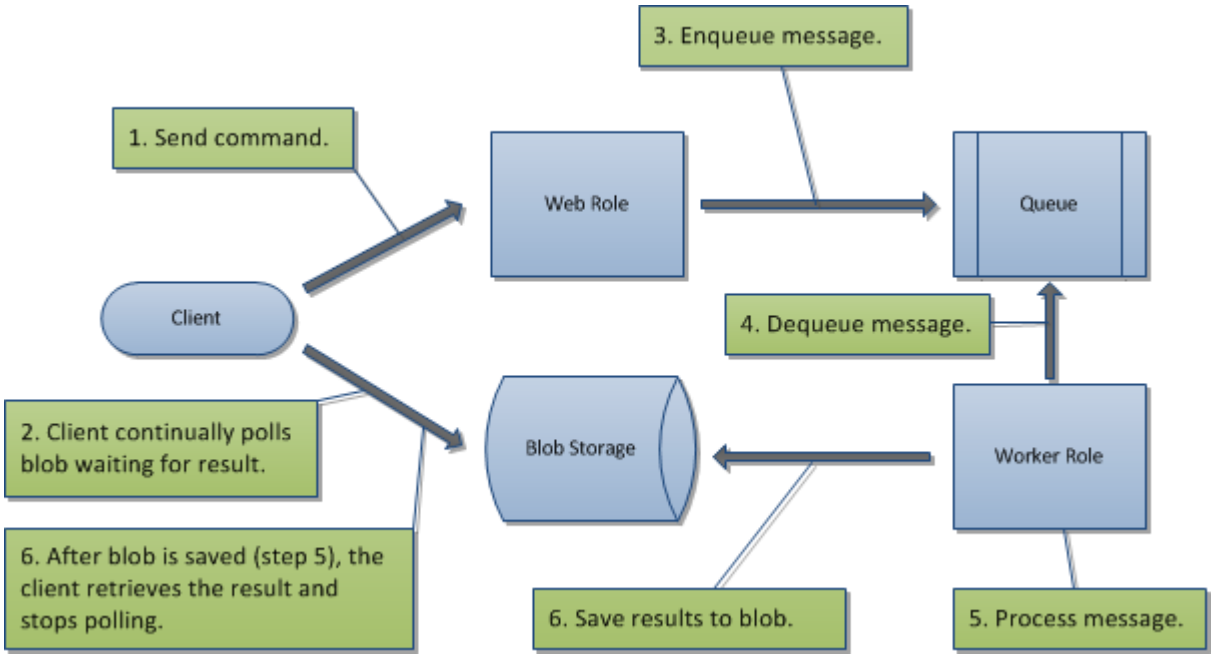


ZADATAK 2

Potrebno je osmisliti arhitekturu socijalne web igre za dva igrača na Azure oblaku (tic-tac-toe), koja može istovremeno da podrži više hiljada konkurentnih korisnika. Scenario igranja odvija se na sledeći način: igrač unosi svoje ime i pritiska dugme *Join*, čekajući da mu sistem dodeli (na slučajan način) drugog igrača. Zatim se igra odvija tako što korisnici naizmenično razmenjuju poteze, dok se igra ne završi pobedom jednog od igrača ili nerešeno. Vodi se statistika koliko je koji igrač ostvario procentualno pobeda u igrama koje je igrao i prikazuje top lista najboljih igrača.

Rešenje

Sledeći dijagram ilustruje korake izvršavanja komandi u ovom sistemu.



Na početku ovog procesa (korak 1), klijent šalje komandu Web ulozi. Ova komanda može biti bilo koji zahtev korisnika od priključivanja određenoj partiji do snimanja korisničkog profila. Posle slanja komande, treba napomenuti da se rezultat ne vraća.

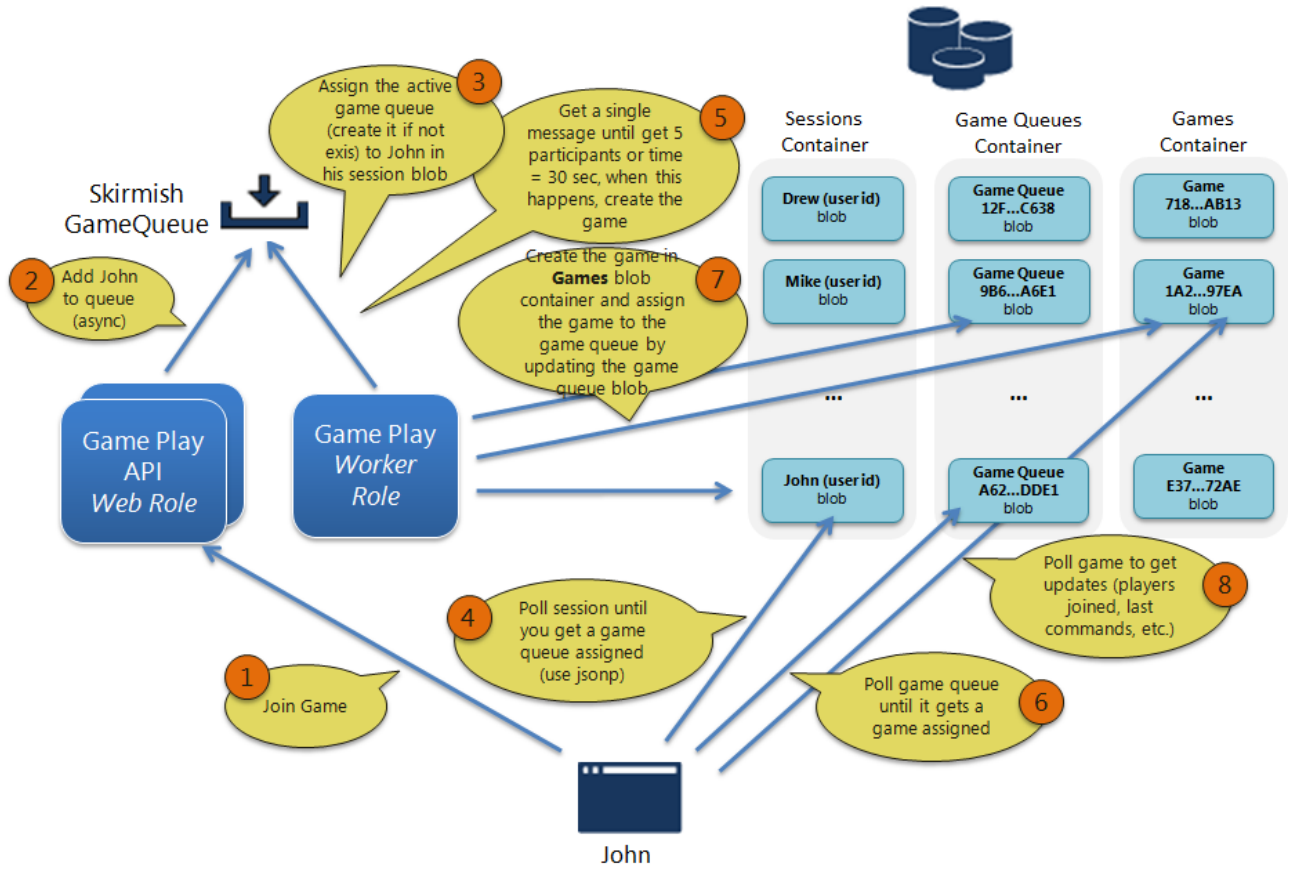
Nakon što klijent pošalje komandu, ona se unosi u Windows Azure skladište u iščekivanju rezultata (korak 2). U koraku 3, Web uloga će uzeti tu komandu i ubaciti podatke u red Windors Azure skladišta. Uloga Worker je da stalno ispituje taj red i proverava da li postoje nove poruke. Kada nova poruka stigne, Worker uloga uklanja iz reda tu poruku (korak 4). Nakon što Worker uloga dobije poruku, ona obrađuje podatke (korak 5). Ova uloga može da radi ažuriranje baze podataka, izvršavanje nekih proračuna,… itd. Nakon što se process završi, Worker uloga čuva rezultat ili poruku o grešci (korak 6).

Arhitektura socijalne igre
(Izvor: http://blog.ntotten.com/2011/07/25/architecture-of-tankster-introduction-to-game-play-part-1/)

Web uloge prihvataju HTTP komande sa klijentskog pregledača i stavljaju u Azure red čekanja. Worker uloge izvlače zahteve iz Azure reda i ažuriraju neki od blob objekata. Postoje sledeće kolekcije blobova:

1. Session Containter sadrži po jedan blob za svakog korisnika (njegovu sesiju): UserID, ActiveGameQueueId
2. GameQueues Containter sadrži za svaku instancu igre po jedan blob za listu igrača koji igraju tu instancu (na ovaj način lako može da se podrži N igrača): GameId, Users[].
3. Games Containter sadrži po jedan blob za svaku započetu instancu ove igre: Users[], ActiveUser, Board, Status.

Statistika igre sakuplja se u SQL azure bazi (UserId, UserName, GamesPlayed, Victories).



Scenario odvijanja igre je sledeći:

1. Klikom korisnika na *Join* dugme u pregledaču rezultuje u slanju HTTP zahteva na server.
2. Web uloga pretvara ovaj zahtev u poruku koju stavlja u Azure red čekanja za igru. Ovaj red odnosi se na zahteve korisnika da se uključe u igru. Odgovor Web uloge klijentu na ovaj zahtev ne sadrži nikakvu informaciju.
3. Worker čita Azure red čekanja za igru, kada se prvi korisnik prijavi, formira blob objekat (u kontejneru QameQueues) i u korisnikov blob sesije upiše njegov ActiveGameQueueId.
4. Klijent odmah počinje da repetitivno iščitava blob sesije korisnika, koji je jedinstven za datog korisnika. Ako je ActiveGameQueueId jednak nula, znači da instanca igre još nije formirana, pa klijent ponavlja korak 4.
5. Worker čita Azure red čekanja za igru, sledećeg prijavljenog korisnika dodaje u ovaj formirani blob objekat i u njegovu sesiju upisuje isti ActiveGameQueueId.
6. Klijent repetitivno čita GameQueue blob, dok god je GameId u njemu jednak 0.

7. Kada se prijavi dovoljno korisnika (ili istekne timeout za prijavu), formira se blob igre. Id ovog bloba (GameId) postavlja se u blob GameQueueId.
8. Klijent dok se igra odvija stalno čita blob Game, da bi znao da li je na potezu, izgled igračkog polja i da li je igra završena i ažurira korisnički interfejs. Potezi se sa klijenta šalju u Azure red, odakle ih čita Worker uloga i ažurira Game blob.
9. Na kraju igre, Worker uloga ažurira statistiku. Klijent može preko posebnog PHP modula u Workeru da po potrebi pročita tabelu najboljih skorova i prikaže korisniku.

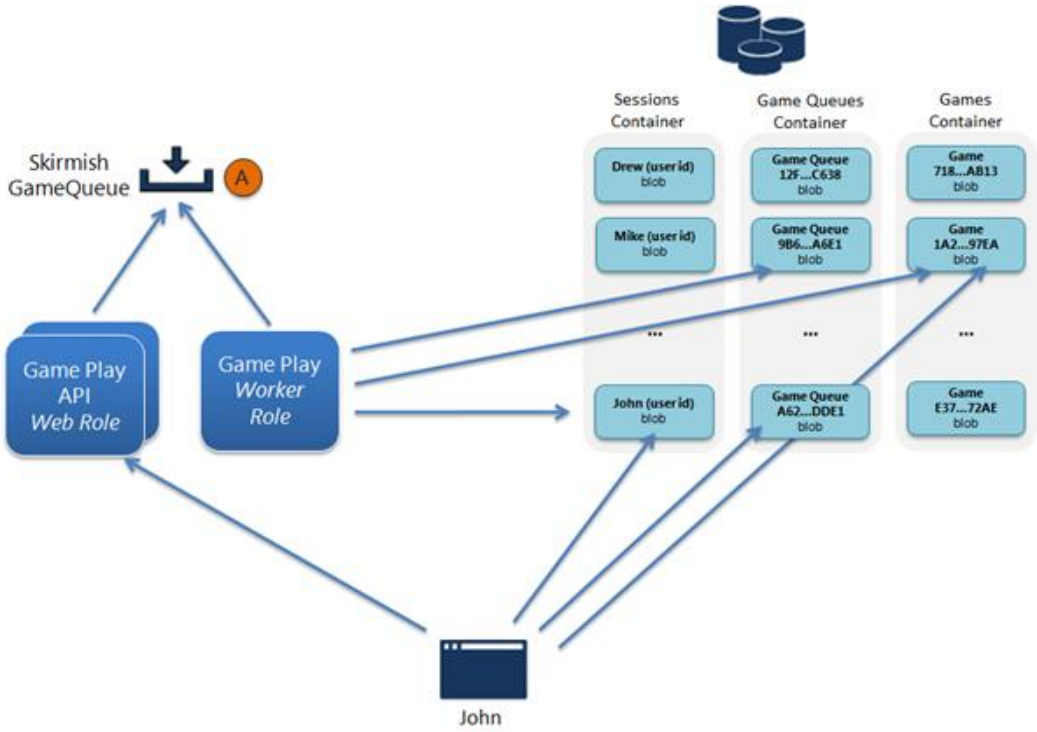
Skalabilnost ove arhitekture:

Mi znamo da u zavisnosti od korišćenjam možemo skalirati Web uloge i Worker uloge, onoliko koliko nam je potrebno, ali možemo kontrolisati i Windows Azure skladište. Možemo povećavati skladište do maksimalne granice.

Najviši nivo skladištenja u Windows Azure zavisi od tipa naloga koji koristimo. Neka je dato jedno Windows Azure skladište sa sledećim nivoem skalabilnosti:

Kapacitet:	do 100 TBs
Transakcije:	do 5000 lica/poruka/blobova po sekundi
Propusni opseg:	do 3 gigabita po sekundi

Pored toga, Windows Azure red može da obradi do 500 poruka u sekundi, a jedan blob može da obradi do 60 megabajta u sekundi. Sa trenutnom arhitekturom za ovu igru, koristimo samo jedan nalog za skladištenje i koristimo po jedan red u zavisnosti od tipa poruke. To znači da je naša sadašnja arhitektura ograničena. Apsolutni, maksimalni, broj transakcija koje možemo da obradi skladište je 5000 po sekundi. Svaki korisnik igre verovatno izaziva 2-3 transakcije u sekundi, što znači da naša igra može da obradi oko 1700 korisnika istovremeno. Očigledno, to nije velika cifra za igru.

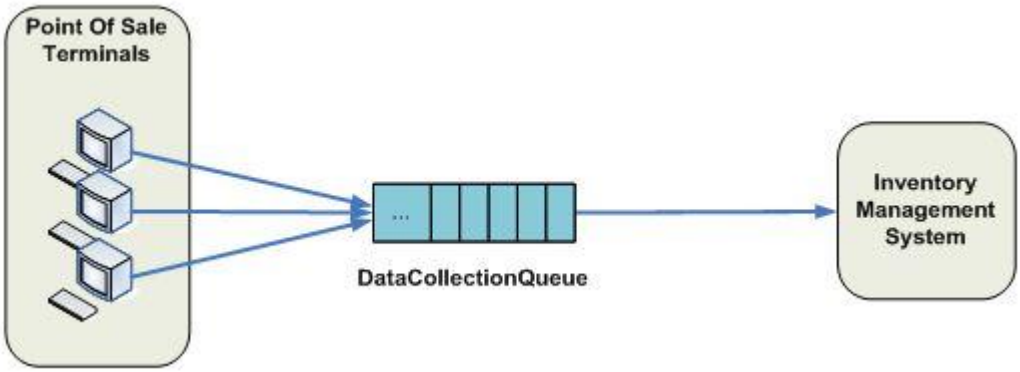


Prvi korak kojim možemo ovo da popravimo je da koristimo više Windows Azure skladišta. Potrebno je samo da konfigurišemo opcije za dodatne naloge i da kreiramo jednostavan algoritam koji definiše koji su redovi i blobovi i tabele na svakom od tih naloga.

Trenutno, svaki tip poruke koristi samo jedan red. To znači da smo ograničeni na 500 zahteva u sekundi na tom redu. Podelićemo zato ove redove u N redova i distribuirati poruke preko njih. Ovaj proces može da se radi na nekoliko načina. Prvi način bi bio definisanjem broja na osnovu neke formule i dodavanje tog ograničenja, koji govori o distribuiranju svakog tipa poruke u redovima. Drugi pristup bi skupljao broj redova na osnovu opterećenja.

ZADATAK 3 - APPFABRIC SERVICE BUS

Razmatramo scenario maloprodajnih objekata nekog trgovinskog lanca marketa, u kome se podaci iz svakog objekta usmeravaju ka jednom centralnom sistemu za upravljanje zalihama, koji koristi podatke da bi utvrdio da li neki artikli treba da budu još uvek na akciji ili ne.



POS = Point Of Sale

Svaki POS terminal prenosi podatke o prodaji slanjem poruk u red **DataCollectionQueue**. Ove poruke ostaju u tom redu dok ne budu preuzete od strane sistema za upravljanje zalihama. Ovaj model često nazivamo *asinhrona razmena poruka*, zato što POS terminal ne treba da čeka odgovor iz sistema za upravljanje zalihama da bi nastavio obradu.

Potrebno je razviti i novi scenario dodavanjem novog zahteva u sistem: svaki vlasnik želi da kontroliše rad svake svoje prodavnice u realnom-vremenu.

Rešenje

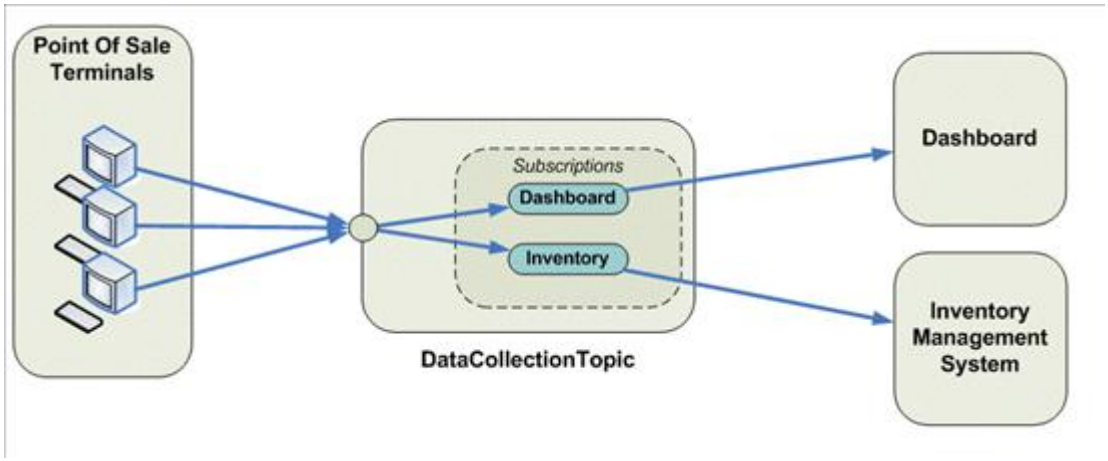
Windows Azure redovi, koji su deo Windows Azure Storage infrastrukture, imaju jednostavan Rest-base Get/Put/Peek interfejs, koji obezbeđuje pouzdanu i prezistentnu razmenu poruka, unutar i između servisa. Service Bus redovi su deo šireg Windows Azure sistema za poruke, koja podržava redove, kao i pretplatu, udaljeni pristup veb servisu i integraciju uzoraka.

Da bismo rešili ovaj zahtev sistem mora da prati tok podataka prodaje. Mi i dalje želimo da šaljemo svaku poruku od POS terminal na sistem za upravljanje zalihama, ali pre toga želimo da još jednu kopiju takve poruke da koristimo da prezentujemo na komandnu-tablu vlasniku marketa.

U situaciji kao što je ova, u kojoj je potrebno da svaka poruka bude korišćenja od više strana, možete koristiti Service Bus sa temama. Teme obezbeđuju pretplatu u kome je svaka objavljena poruka stavljena na raspolaganje jednog ili više registrovanih pretplatnika na tu temu. Sa druge strane, kod redova, svaka poruka je primljena samo od strane jednog korisnika.

Poruke se šalju na temu, na isti način kao što se šalju u red. Međutim, poruke se ne primaju direktno od teme, već se primaju od pretplatnika. Možete zamisliti pretplatu na temu kao virtuelni red koji prima kopije poruka, koje se šalju na tu temu. Poruke se primaju od pretplate na isti način kao što su bile primljene u redu.

U cilju podrške kontrolnoj tabli, moramo da formiramo drugu pretplatu na temu, kao što je prikazano na slici:



Ovakvom konfiguracijom, svaka poruka sa POS terminal je dostupna i kontrolnoj table i pretplatnicima sistema za zalihe.

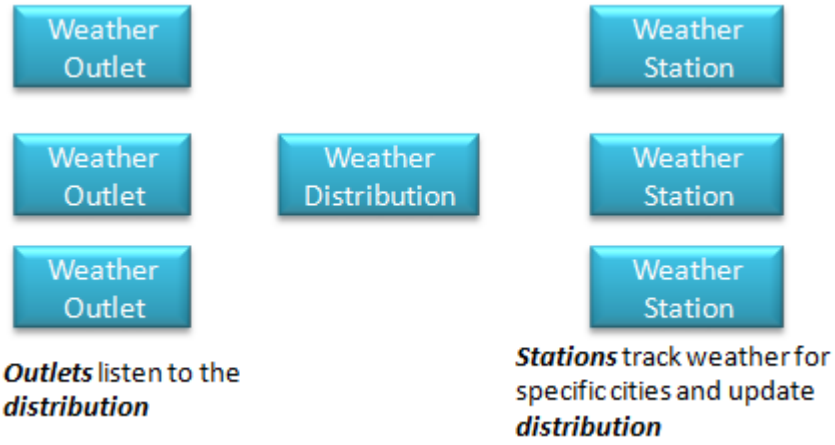
ZADATAK 4 - APPFABRIC SERVICEBUS RELAY

Metereološka stanica

Ova usluga koristi uslugu Service bus i nije potrebno da bude razvijena za Service bus.

Ovde se jednostavno koristi AppFabric kao relej za objaviljivače koji komuniciraju sa potrošačima. Demonstriraćemo AppFabric Service Bus na jedan način - koristićemo podršku za multicast događaje i omogućiti dozvolu pristupa za N objavljivača događaja i M potrošača događaja, koji se sastaju na istoj krajnjoj tački.

U ovom primeru, mnogo objavljivača dobija vreme na osnovu PTT broja određenog mesta i objavljuje ga tako da informaciju dobija više korisnika. Kada korisnik želi informaciju o vremenu, on će otvoriti prozor sa servisom koji pruža vremensku prognozu.



Ova aplikacija dobija vreme od veb servisa Nacionalnog zavoda za metereologiju i jednostavno stiže informacija o vremenu do slušaoca.

Rešenje