

Message Queue

Šta je message queue?

- ▶ Predstavljaju način komunikacije između aplikacija ili komponenti slanjem poruka
- ▶ Najbolju primenu imaju kod asinhronne obrade velike količine podataka, što predstavlja realan problem kada se govori o web aplikacijama
- ▶ Sa redom poruka možemo efikasno podržati znatno veću količinu konkurentnih poruka, poruke se razmenjuju u realnom vremenu umesto periodičnog pull-ovanja, poruke se automatski čiste nakon što su primljene i ne moramo brinuti o deadlock-u

MQ

- ▶ MQ obično (ali ne uvek) predstavljaju brokere koji olakšavaju razmenu poruka tako što oboezbeđuju protokole ili interfejse kojima drugi servisi mogu pristupati
- ▶ Ovaj interfejs povezuje **producer-a** koji stvara poruku, i **consumer-a** koji zatim obrađuje tu poruku
- ▶ U kontekstu web aplikacija, producer je klijentska aplikacija koja kreira poruke bazirane na komunikaciji sa korisnikom, dok je consumer neki uglavnom daemon (pozadinski) proces

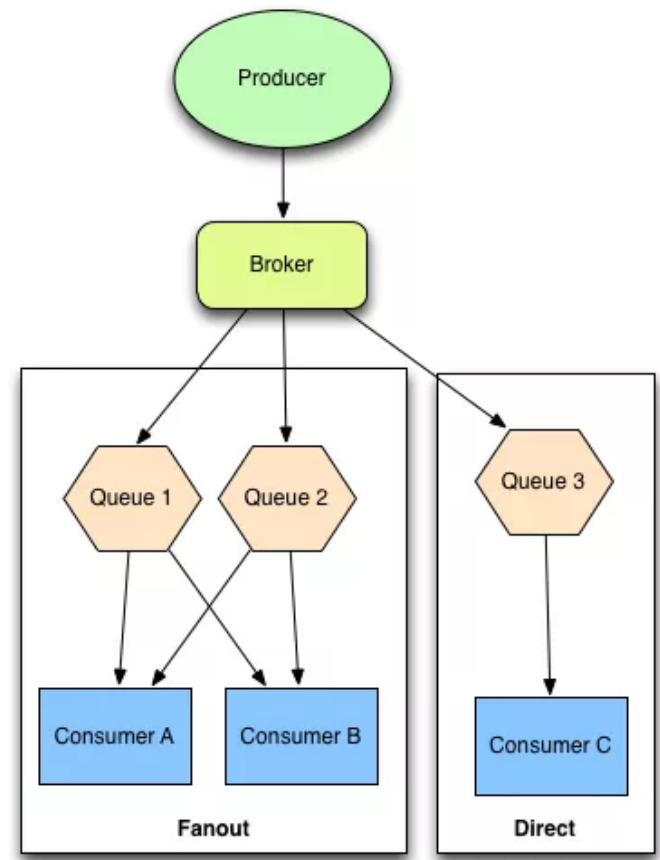


Broker i njegova uloga

- ▶ Broker primi poruku od publisher-a (aplikacija koja šalje poruke, još se naziva i producer) i prosleđuje ih consumer-ima (aplikacija koja obrađuje poruke)
- ▶ Pošto je ovo mrežni protokol, brokeri, publisher-i i consumer-i mogu da budu na različitim računarima
- ▶ Svi podaci koji stižu sa porukom su skroz transparentni, i mogu biti vidljivi samo u aplikaciji koja treba da je obradi. Međutim, moguće je definisati neke metapodatke za same brokere

Tipovi

- ▶ **Direct** – poruka ide od jednog producer do tačno jednog consumer-a
- ▶ **Fanout** – poruka se šalje na više različitih consumer-a

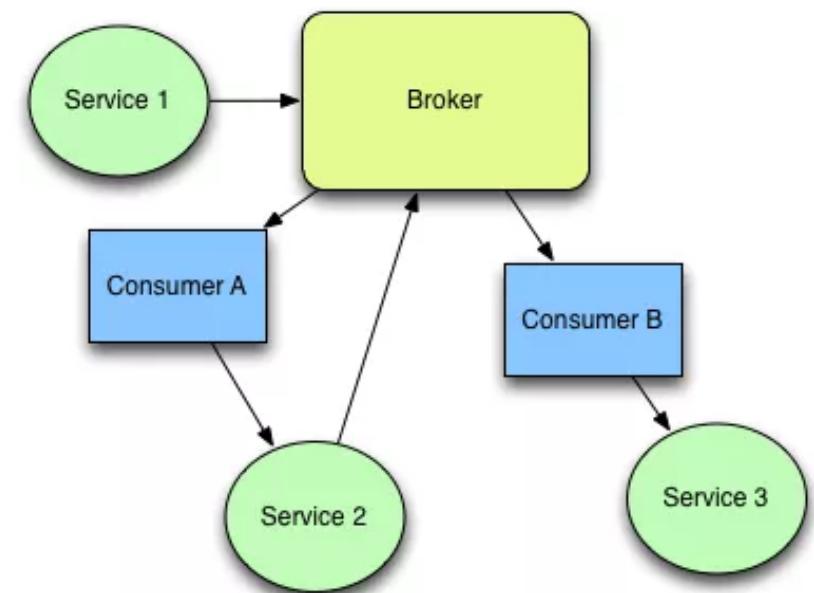


Perzistentnost

- ▶ Jedna od bitnih stvari MQ jesu robusnost i pouzdanost koje se postižu strategijama perzistentnosti
- ▶ Da bi pouzdanost poruka ostala visoka, većina MQ čuva sve pristigle poruke na disku, do trenutka kada ne budu primljene i obrađene od strane consumer-a
- ▶ Čak i ako se aplikacija sruši, ili sam MQ, sve pristigle poruke se čuvaju do trenutka kada sistem opet bude operabilan. Ovo je ključna razlika u odnosu na web aplikacije koje ne koriste MQ, svi zahtevi se čuvaju u memoriji, i ako bilo šta podje po zlu, ostaju zauvek izgubljeni

Servis-orientisana arhitektura sa porukama

- ▶ Prednost ovakve arhitekture sistema jeste što je jedan veliki sistem čitave aplikacije podeljen na više manjih podsistema što rezultuje time da je sa manjim celinama lakše upravljati, testirati i debagovati
- ▶ Takođe, ovo omogućava i da na razvoju aplikacije radi više timova, koji rade nazavisno od drugih. Unutar tima je moguće čak i koristiti različite alate, sve dokle god su jasno definisane ulazne i izlazne tačke za svaki od podsistema



Mane MQ

- ▶ Podešavanje i konfiguriranje MQ (pogotovo složenijih) može dodati mnogo dodatnih delova aplikaciji
- ▶ Rukovanje greškama zahteva dodatni ručni napor
- ▶ Komunikacija sa MQ dodaje određenu složenost u logiki aplikacije
- ▶ Za većinu aplikacija mora se ručno definisati *state machine* za poruke. Na primer, imamo poruku koju želimo da prosledimo na tri različita servisa. Na nama je da ručno definisemo workflow reda da bi ovo bilo moguće

Prednosti MQ

1. Decoupling (razdvajanje) – na samom početku rada na projektu je veoma teško predvideti sve buduće potrebe aplikacije. Uvođenjem sloja između procesa, MQ implicitno uvode data-based interfejs koji oba procesa moraju da implementiraju. Ovo omogućava da se procesi nezavisno proširuju i menjaju, dokle god poštjuju interfejse koje je MQ propisao
2. Redudantnost – Neki procesi se neuspešno završe prilikom obrade poruka. Kod klasičnih web aplikacija, u ovim situacijama poruke i podaci koje one nose se zauvek gube. MQ rešava ovo tako što čuva podatke dokle god se poruka uspešno ne obradi. Većina MQ ne dozvoljava da poruka bude uklonjena iz reda, dokle god se proces koji je obrađivao tu poruku uspešno ne završi. Na ovaj način, svi podaci ostaju sigurni dokle god su potrebni

Prednosti MQ

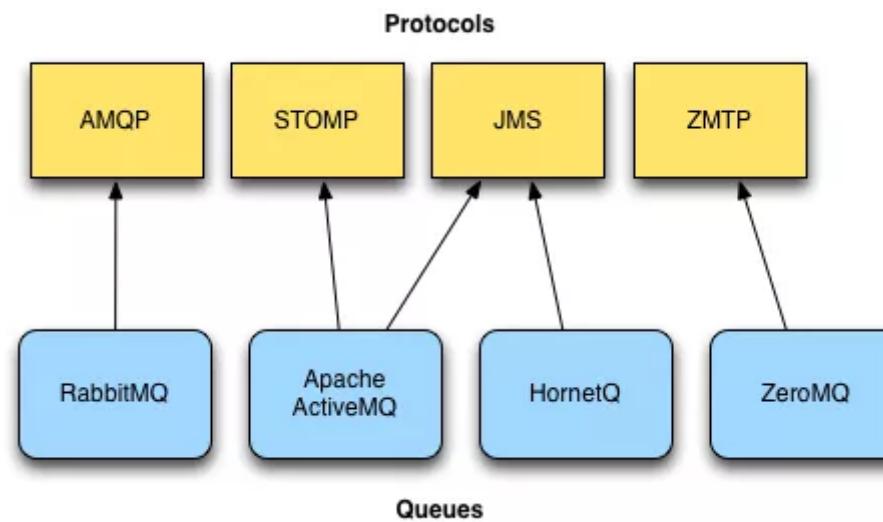
3. Skalabilnost - pošto MQ razdvajaju procese, lako je povećati brzinu kojom se poruke dodaju u red i obrađuju: samo se doda novi proces. Nikakav kod se ne mora menjati i nikakva konfiguracija se ne mora menjati
4. Nemamo problem ukoliko se naglo poveća količina saobraćaja koji pristiže na server. Sve poruke se samo dodaju u red i budu obrađene u trenutku kada dođu na red
5. Pošto MQ razdvajaju arhitekturu sistema na više manjih, nezavisnih podsistema, nemamo problem ako jedan servis padne da će se srušiti ceo sistem. Poruke koje stignu i koje treba da obradi servis koji trenutno nije u funkciji se isto dodaju u red i bivaju obrađene kada servis ponovo bude funkcionalan

Prednosti MQ

6. Garantovana isporuka – redundancija garantuje da će poruka koja se nalazi u redu biti kad tad obrađena tačno jednom. Bez obzira koliko različitih procesa istovremeno čitaju poruke iz istog reda, svaka poruka može biti obrađena samo jednom
7. Garantovan redosled – poruke se u MQ obrađuju po FIFO principu, odnosno poruka koja prva dođe u red će biti prva i obrađena. U nekim sistemima je ovaj redosled veoma važan
8. Asinhrona komunikacija – MQ podržavaju asinhronu obradu poruka, pošto postoje realne situacije kada imamo neku poruku, ali ona ne treba da se odmah obradi. Zato se ovakve poruke samo stave u red i budu obrađene nekad kasnije kada bude bilo potrebno

Vrste MQ

- ▶ Postoje različite implementacije MQ, svaki ima svoje prednosti i mane
- ▶ Generic MQ koji su danas najpopularniji i koji se najviše koriste su RabbitMQ, Apache ActiveMQ i ZeroMQ



ZeroMQ

- ▶ Za razliku od drugih, ZeroMQ se ponaša kao framework za razvoj MQ
- ▶ Glavni fokus je na tome da se pristigle poruke samo što pre rasporede preko mreže, dok se recimo RabbitMQ ponaša kao pravi broker i vodi računa o persistenciji, filtriranju i nadgledanju poruka
- ▶ Nema ugrađenog brokera, što znači da ne postoji centralni dispečer koji će da upravlja porukama i zapravo nije „pravi MQ servis“
- ▶ Ukoliko samo želimo da se poruke efikasno razmenjuju, a sav posao da se ručno uradi, ZeroMQ predstavlja odličan izbor
- ▶ Ukoliko želimo da koristimo MQ u tipičnim slučajevima korišćenja, ZeroMQ nikako nije dobar izbor i trebalo bi razmisliti o ostalim implementacijama

RabbitMQ and ActiveMQ

- ▶ Suština ove dve implementacije je ista, razlika je samo u nekim detaljima
- ▶ ActiveMQ je pisan u Java programskom jeziku i podržava JMS i STOMP protokole. JMS protokol omogućava da se ActiveMQ koristi u aplikacijama koje su pisane u Java, Scala, Clojure (JVM jezici) programskim jezicima, a STOMP pruža podršku za Ruby, PHP, i Python
- ▶ RabbitMQ je pisan u Erlang-u i koristi AMQP protokol
- ▶ Protokoli koji se koriste (JMS i AMQP) imaju određene razlike. Jedna od ključnih jeste ta što kod AMQP protokola producer šalje poruku brokeru bez da zna strategiju distribucije koja se koristi, dok kod JMS protokola producer je svestan strategije koja se eksplicitno koristi



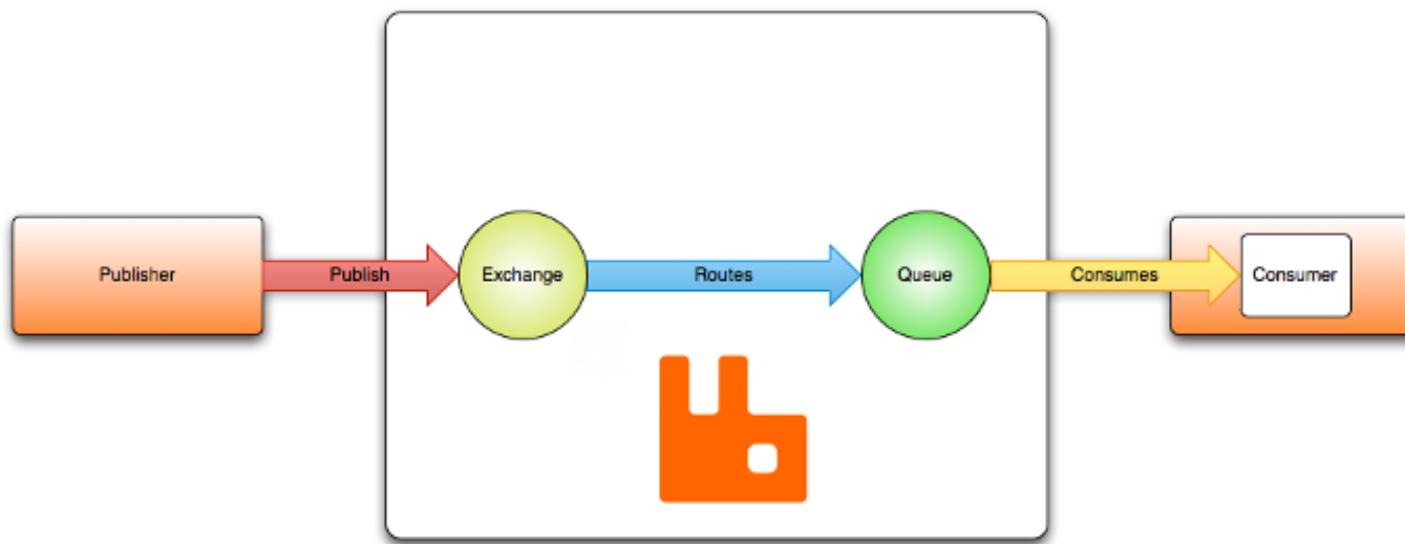
RabbitMQ

AMQP

- ▶ Advanced Message Queuing Protocol
- ▶ Protokol koji omogućava usklađenim klijentskim aplikacijama da komuniciraju sa usklađenim brokerima posrednicima u razmeni poruka

AMQP model

- ▶ Publisher prvo šalje poruke na Exchange, koji se često poredi sa poštanskim sandučićima. Exchange zatim kopiju poruke prosleđuje u redove koristeći pravila koji se zovu *bindings*. Zatim AMPQ brokeri ili proslede poruku consumer-ima koji su se preplatili na odgovarajući red ili consumer sam šalje zahtev da pull-uje prisigle poruke iz reda



dead letter queue

- ▶ U nekim situacijama poruka ne može biti ispravno prosleđena. Tada se ona ili odbacuje, ili vraća publisher-u ili broker implementira proširenje – dead letter u queue
- ▶ U ovom redu se nalaze poruke koje nisu uspešno prosleđene na odgovarajuća mesta, i na publisheru je da sam odredi kako će rukovati sa njima, tako što ih ponovo publish-uju koristeći određene parametre

Tipovi Exchanges-a

- ▶ Exchanges su AMPQ komponente na koje se šalju poruke. Prime poruku i dalje je proslede na 0 ili više redova. Algoritam za rutiranje zavise od tipa razmene i od pravila koja se zovu bindings
- ▶ AMPQ brokeri omogućavaju četiri tipa razmene:
 1. Direct
 2. Fanout
 3. Topic
 4. Headers

Atributi Exchanges-a

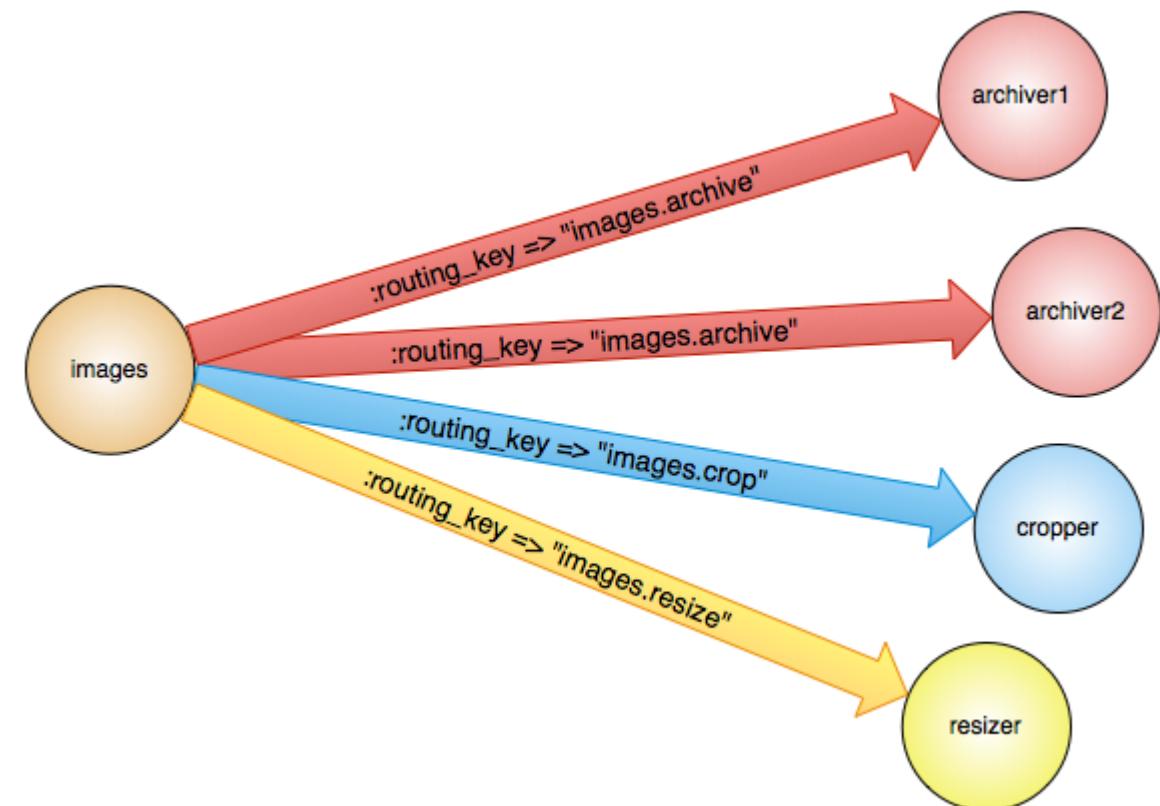
- ▶ Osim tipa, svaki Exchange je specificran određenim atributima, od kojih su najvažniji sledeći:
 1. Name
 2. Durability – Exchange preživljava ukoliko se broker restartuje
 3. Auto Delete – Exchange se briše kada se poslednji red otkači od njega
 4. Arguments – opcioni, koriste ze plugin-e i za polja koja se tiču samog brokera

Default Exchange

- ▶ Direktna razmena, bez imena (atribut name ima vrednost empty-string)
- ▶ Predefinisan
- ▶ Svaki red koji se automatski kreira se vezuje za ovaj Exchange i ima vrednost za routing key ime kreiranog reda

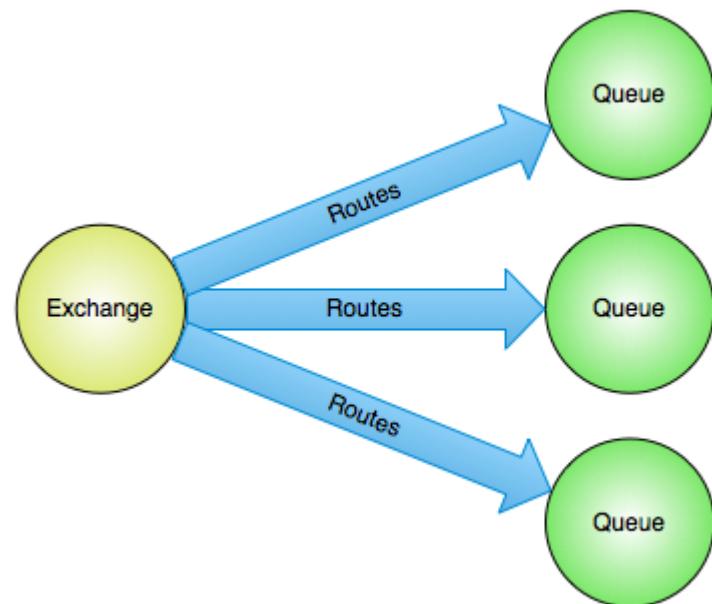
Direct Exchange

- ▶ Poruke se prosleđuju na osnovu vrednosti *routing_key*
- ▶ Princip rada:
 1. Red se *bind-uje* na Exchange sa *routing_key* K
 2. Kada stigne nova poruka koja ima *routing_key* R, Exchange je prosleđuje u red ako je K = R



Fanout Exchange

- ▶ Prosleđuje kopiju poruke na sve redove koji su povezani na njega
- ▶ Vrednost za *routing_key* se ignoriše



Topic Exchange

- ▶ Prosleđuje poruku na jedan ili više redova kojima se poklapa vrednost *routing_key* i pattern koji je korišćen prilikom bind-ovanja reda za Exchange
- ▶ Primeri upotrebe:
 1. Distribucija podataka specifičnih za određeno geo-područje
 2. Orkestracija procesa različitih vrsta

Headers Exchange

- ▶ Dizajnirano za rutiranje koje zavisi od više atributa, a ne samo od vrednosti *routing_key*
- ▶ Ove atribute je lakše izraziti kao zaglavlja poruke, nego kao *routing_key*
- ▶ Ignoriše *routing_key*
- ▶ Naziva se još i "direct exchanges on steroids". Pošto se poruka rutira na osnovu vrednosti atributa u zaglavljtu, mogu se koristiti i kao *direct Exchange* gde *routing_key* ne mora biti striktno tipa string

Redovi

- ▶ Redovi u AMQP modelu su veoma slični redovima i u ostalim MQ sistemima
- ▶ Čuvaju poruke koje aplikacije kasnije koriste
- ▶ Pre nego što se red koristi, on mora biti deklarisan. Deklaracija će izazvati kreiranje novog reda ukoliko on ne postoji. Deklaracija nema smisla ukoliko se navede red sa imenom koje već i postoji i ako ta dva reda imaju iste vrednosti svih atributa – tada se baca Exception

Atributi redova

1. Name – UTF8 karakteri, do 255 bajtova dugački. Ukoliko se za vrednost ovog atributa postavi prazan string, AMPQ će da sam generiše jedinstveno ime za ovaj red. Ne mogu počinjati sa „amq“ zato što je rezervisano za interna imena unutar samog brokera
2. Durable - red se čuva na disku i preživljava ukoliko se broker restartuje. Suprotni redovi se zovu *transient*. Poruke iz reda će preživeti ukoliko su persistirane na disku, u suprotnom se ne mogu oporaviti
3. Exclusive – koristi se za samo jednu konekciju i briše se kada se ta konekcija zatvori
4. Auto-delete - red se briše ukoliko ima samo jednog consumera i on se unsubscribiuje
5. Arguments - opcioni, koriste se za plugin-e i za polja koja se tiču samog brokera

Bindings

- ▶ Pravila koja Exchange koristi za rutiranje poruka
- ▶ Recimo imamo Exchange E koji treba da prosledi poruku na red Q. Q mora biti vezan (*bound*) za E
- ▶ Mogu da imaju opcioni *routing_key* koji koriste neki tipovi exchanges-a i on se ponaša kao filter kome se poruka šalje
- ▶ Imajući ovaj sloj indirekcije, omogućavaju se scenariji rutiranja koji su nemogući ili vrlo teški za implementaciju, koristeći direktno objavljuvanje u redovima
- ▶ Takođe eliminišu određenu količinu dupliranog posla koji bi developeri trebali da urade

Consumers

- ▶ Da bi čuvanje poruka u redovima imalo smisla, mora da postoji aplikacija koje će da ih koriste
- ▶ U AMPQ modelu postoje dva načina da aplikacija ovo može da uradi:
 1. Push API – poruka se prosleđuje consumer-u čim stigne
 2. Pull API – consumer sam zatraži poruke kada mu trebaju