High Availability u SQL Server DBMS-u

Seminarski rad iz predmeta

*Sistemi za upravljanje bazama podataka*

na master akademskim studijama

Mentor:

dr Aleksandar Stanimirović

Student:

Arsenije Arsenijević

br. ind. 793

# Sadržaj

[Uvod 1](#_Toc44035101)

[Modeli redundansi 2](#_Toc44035102)

[Redundansa procesa 2](#_Toc44035103)

[Redundansa podataka 3](#_Toc44035104)

[Fizička redundansa podataka 3](#_Toc44035105)

[Logička redundansa podataka koristeći replikaciju 3](#_Toc44035106)

[Modeli redundansi podataka 4](#_Toc44035107)

[Fragmenti baze, particionisanje i replikacija fragmenata 4](#_Toc44035108)

[Veze kardinalnosti između primarnih i sekundarnih replika 5](#_Toc44035109)

[1\*Primar/1\*Sekundar 5](#_Toc44035110)

[1\*Primar/Y\*Sekundar 5](#_Toc44035111)

[1\*Primar 5](#_Toc44035112)

[X\*Primar 6](#_Toc44035113)

[Veze između baze podataka i fragmenata 6](#_Toc44035114)

[Neparcionisana replicirana baza podataka 6](#_Toc44035115)

[Particionisana replicirana baza podataka 6](#_Toc44035116)

[Mešani replicirani fragmenti 6](#_Toc44035117)

[Modeli redundanse procesa 7](#_Toc44035118)

[Aktivan/Na čekanju (2N) 7](#_Toc44035119)

[Hot Standby 8](#_Toc44035120)

[Warm Standby 8](#_Toc44035121)

[Aktivan/S\*Rezerva 8](#_Toc44035122)

[N\*Aktivan 9](#_Toc44035123)

[N\*Aktivan/S\*Rezerva 11](#_Toc44035124)

[Drugi modeli redundanse 11](#_Toc44035125)

[Pogled aplikacije 12](#_Toc44035126)

[SQL Server Always On Availability Groups 13](#_Toc44035128)

[Korišćeno pri implementaciji 13](#_Toc44035129)

[Tipovi replika 15](#_Toc44035130)

[Modovi dostupnosti 16](#_Toc44035131)

[Latencija podataka na sekundarnim replikama 16](#_Toc44035132)

[Tipovi failover-a 17](#_Toc44035133)

[Konekcije klijenata 17](#_Toc44035134)

[Period isteka timeout-a sesije 18](#_Toc44035135)

[Automatsko popravljanje stranica 18](#_Toc44035136)

[Oporavak replika 19](#_Toc44035137)

[Licence 19](#_Toc44035138)

[Implementacija 20](#_Toc44035139)

[Podešavanje okruženja 21](#_Toc44035140)

[Kreiranje nove AG 24](#_Toc44035141)

[Failover 28](#_Toc44035142)

[Zaključak 34](#_Toc44035143)

[Literatura 35](#_Toc44035144)

# Uvod

Baze podataka se aktivno koriste u sistemima raznih poslovnih namena. Postoji mnogo sistema/aplikacija koje zavise umnogo od dostupnosti baze podataka i tu u igru dolaze baze podataka sa visokom dostupnošću (eng. *High-Availability - HA*). HA sistemi baza podataka se koriste kada je potrebno da baza bude dostupna skoro 100% vremena. Ako se potrebna dostupnost definiše kao ’5 devetke’ odnosno 99.999% ili ’6 devetke’ - 99.9999%, sve komponente u sistemu moraju da zadovolje pooštrene uslove kako bi se taj nivo dostupnosti ostvario. Uzmimo u obzir da dostupnost ’6 devetke’ dozvoljava da sistem ne bude dostupan samo 32 sekunde godišnje. Lako je razumljivo da pod takvim strogim zahtevima sve akcije skrivanja kvarova moraju trajati najviše sekundu. Ovakve baze podataka se nazivaju Visoko Dostupne Baze Podataka (eng. *Highly Available Databases*), a sistemi kojima se ostvaruju takve baze se nazivaju Sistemi za upravljanje bazama podataka visoke dostupnosti (eng. *Highly Available Database Management Systems – HA-DBMS*).

Svi HA-DBMS rade slično HA aplikacijama: visoka dostupnost se postiže *redundantnošću procesa* – više instanci procesa su pokrenuti u isto vreme, tipično na hardveru u više-čvornom klasteru. Pored jednog ili više *aktivnih procesa* koji rade kao servis, postoje i *procesi na čekanju*, ili redundantni aktivni procesi, koji su pokrenuti na drugim računarskim čvorovima i spremni su da preuzmu operaciju (i nastave servis), ako bi aktivni proces na neki način bio ugrožen. Procesi baza podataka uključuju podatke čije su stanje i dostupnost krucijalni za uspešan nastavak servisa. Odavde govorimo o *redundansi podataka*, koja ima za cilj da omogući dostupnost podataka u slučaju da neke komponente koje drže podatke zakažu.

Microsoft ima svoje rešenje za HA-DBMS, a to je *Always On Availability Groups*.

# Modeli redundansi

HA sistemi baza podataka koriste više koncepata redundantnosti. Svi HA-DBMS sistemi se zasnivaju na postojanju redundantnih procesa baze podataka. Kada proces baze podataka zakaže (bilo hardverski ili softverski problem), drugi proces baze podataka preuzima odgovornost opsluživanja servisa koji je prvi proces prethodno kvaru opsluživao. Da bi se pružila korektnost/tačnost podataka, neophodno je da svaki redundantni proces vidi isti set izmena nad bazom podataka. U suštini postoje dva načina dase ovo osigura: jedna tehnika, replikacija, oslanja se na procese baza podataka koji eksplicitno razmenjuju izmene između sebe. U zavisnosti od implementacije, svaka replika može da čuva svoju kopiju u glavnoj memoriji (RAM) ili na disku. Replikacija se ne radi ekskluzivno između pojedinačnih baza. U distribuiranim sistemima baza podataka, jedna baza podataka biva upravljana preko više procesa baza podataka na različitim čvorivima, sa mogućim internim replikacijama između sebe (eng. *intra-database*). Alternativa, druga tehnika, je osigurati da svi redundantni procesi vide isti set podataka tako što stvarno ,,gledaju’’ isti set podataka. Ovo je moguće postognuti deljivim diskovima (eng. *Shared-Disks*) tako što svi procesi pristupaju istom setu diskova. Kako svi procesi pristupaju istom setu diskova, različiti procesi baza podataka ne trebaju eksplicitno razmenjivati izmene. Umesto toga, svi procesi imaju jedan, koherentan pogled na podatke. Primetimo da deljivi disk takodje može imati redundansu, ali ovo je redundansa na nižim nivoima – npr. RAID ili kopiranje preko mreže (eng. *Network-Based Mirroring*) – i proces je često apsolutno transparentan DBMS-u.

Dva pristupa pomenuta iznad mogu se mapirati na dve dobro poznate DBMS arhitekture: *deljivo-ništa*  i *deljivi-disk* [2], respektivno. U ovom radu se fokusiramo na sama sredstva kojima dolazimo do visoke dostupnosti.

Više modela redundantnosti je moguće u HA-DBMS sistemu i oni su definisani ispod. Razlikujemo *redundantnost procesa* koja definiše dostupnost procesa baza podataka i *redundansu podataka* koja specificira, za rešenja zasnovana na replikaciji, broj kopija podataka koje se eksplicitno održavaju. Oba tipa redundantnosti su potrebna da bi se pružio Visoko dostupan servis baze podatka (eng. *High Availability Database Service*).

## Redundansa procesa

Redundansa procesa u HA-DBMS-u dozvoljava da DBMS nastavi svoju operaciju u prisustvu zakaza procesa. Kao što ćemo reći kasnije, većina modela redundanse procesa može biti implementirana i preko tehnologije deljivog diska i replikacije.

Proces koji je u *aktivnom* (eng. *Active*) stanju je onaj koji trenutno pruža (ili je osposobljen da pruža) servis baze podataka. Proces koji je u stanju *na čekanju* (eng. *Standby*) trenutno ne pruža servis ali je spreman da preuzme ulogu davanja servisa na veoma brz način, ako trenutni aktivan proces zakaže. Ovo se naziva (eng.) *failover*. U nekim slučajevima, novi tip procesa, *rezervni* (eng. *Spare*) proces se može koristiti. Rezervni proces može biti implementiran ili kao pokrenuta komponenta kojoj nije zadat nikakav izvršni zadatak (eng. *workload*), ili kao komponenta koja je definisana ali nije instancirana. *Rezerva* može biti uzdignuta na nivo *aktivna* ili *na čekanju* nakon odgovarajuće inicijalizacije.

Redundansa procesa donosi pitanje kako (ili da li) će transparentnost redundanse biti održana u HA-DBMS-u. Od svih pokrenutih procesa, neki mogu biti aktivni a neki ne. U slučaju failover-a aktivni procesi mogu biti promenjeni. Zadatak nalaženja relevantnih aktivnih procesa može biti odgovornost ili aplikacije ili posebnog softverskog sloja (eng. *Software Layer*) koji je zadužen za transparenciju redundanse.

## Redundansa podataka

Redundansa podataka je takođe neophodna za visoku dostupnost (HA). U suprotnom, gubitak jedne jedine kopije podataka bi rezultiralo time da baza postaje nedostupna kao takva. Redundansa podataka može biti pružena na fizičkom ili logičkom nivou.

### Fizička redundansa podataka

Fizička redunsansa podataka se odnosi na oslanjanje na softver i/ili hardver ispod baze podataka za održavanje više fizičkih kopija podataka. Iz perspektive baze podataka, čini se da postoji samo jedna kopija podataka. Neki primeri fizičke redundanse podataka uključuju: replikacija diska, RAID, udaljena replikacija diska, replicirani fajl sistemi...

Sve ove tehnologije dele zajednički atribut da održavaju odvojenu fizičku kopiju na moguće različitoj geografskoj lokaciji. Kada je primarna kopija podataka izgubljena, procesi baze podataka koriste drugu kopiju podataka. Ove tehnologije se mogu razlikovati u pogledu transparentnosti zakaza koji je podržan. Repliciranje diska (eng. *Disk Mirroring*) i RAID, npr. , čine fizičku redundansu apsolutno transparentnom bazi podataka.

Fizička redundansa podataka se često kombinuje sa redundansom procesa koristeći SAN (eng. *Storage Area Network*). Ovo dozvoljava da više čvorova ima pristup istoj fizičkoj bazi podataka. Ako jedan server baze zakaže (usled softverske ili hardverske greške), baza podataka je i dalje dostupna kroz druge čvorove. Ovi drugi čvorovi nakon toga mogu da nastave servis.

### Logička redundansa podataka koristeći replikaciju

Logička redundansa podataka se odnosi na situaciju gde baza podataka eksplicitno održava višestruke kopije podataka. Transakcije koje se primenjuju nad primarnom bazom D se repliciraju nad sekundarnom bazom D’ koja je manje-više ažurirana u odnosu na primarnu bazu, a u zavisnosti od tipa sinhronizacije replikacionog protokola u HA bazi podataka. Dodatno replikaciji između baza podataka (eng. *inter-database*), replikacija u samoj bazi podataka (eng. *intra-database*) se koristi u distribuiranim sistemima da bi se postigla visoka dostupnost koristeći samo jednu bazu podataka. Primetimo da govorimo o replikaciji u generalizovanom obliku kako je šema replikacije zavisna od dobavljača/pružaoca/tvorca konkretne tehnologije baze podataka (pretpostavljajući da baza podataka i replikacija dolaze od istog dobavljača). Replikacija može biti *sinhrona* (eng. *Synchronous*) i *asinhrona* (eng. Asynchronous), može biti bazirana na prosleđivanju logova ili direktne replikacije kao deo transakcije, transakcije mogu biti grupisane (eng. *Batched*) i moguće kombinovane grupom izvršenja/posvete/komita (eng. *Commits*). Odabrana metoda zavisi od samog produkta baze podataka i potrebnog nivoa *sigurnosti* (eng. *Safeness*) [3]. Kod (eng.) *1-safe* replikacije (,,asinhrona replikacija’’) transakcije se repliciraju nakon što se *izvrše/komituju* nad primarnom kopijom. Kod (eng.) *2-safe* replikacije (,,sinhrona replikacija’’) transakcije se repliciraju u sekundarnoj kopiji, ali se ne komituju pre nego što se potvrdi komit na primarnoj kopiji. Kod (eng.) *2-safe committed* replikacije transakcije se repliciraju i komituju na sekundaru pre nego što se potvrdi komit na primaru. Kod (eng.) *very safe* replikacije sve operacije osim čitanja se isključuju ako bilo primarna ili sekundarna replika postane nedostupna. Iako se većina metoda pružanja sigurnosti odnosi na kontekst udaljenih rezervi, rezultati su važeći i za (eng.) *in-cluster* operacije takođe.

# Modeli redundansi podataka

Za trajanje ostatka ovog prvog, teorijskog dela, redundansa podataka se odnosi na *logičku* redundansu podataka. Redundansa podataka predstavlja broj samostalnih kopija podataka koje se održavaju od strane procesa baze podataka preko replikacije. Ne računaju se kopije koje se možda održavaju od strane nekog fizičkog modela redundanse. Npr. dve kopije podataka koje se održavaju od strane niza diskova (eng. *Disk Array*) ili od strane menadžera particija se računaju kao jedna u kontekstu ove diskusije, dok se dve kopije podataka koje se održavaju od strane baze podataka računaju kao dve. Primetimo da u oba slučaja, gubitak jedne kopije se može obraditi transparentno bez gubitka dostupnosti.

Prvo raspravljamo modele redundantnosti podataka u detalje zato što je ovo oblast koja je poprilično jedinstvena za DBMS-ove visoke dostupnosti (HA-DBMS-ovi).

## Fragmenti baze, particionisanje i replikacija fragmenata

Da bismo definisali modele redundantnosti podataka moramo definisati šta zapravo repliciramo, odnosno *fragmente baze podataka* (fragment je ovde generalizacija ustaljene definicije fragmentacije tabele u relacionoj bazi podataka). *Fragmentacija* baze podataka je dekompozicija baze podataka D u fragmente P1, P2... Pn koja mora ispuniti sledeće uslove:

1. *Kompletnost* – svi podaci koji postoje u bazi podataka moraju postojati u nekom fragmentu.
2. *Rekonstrukcija* – treba biti moguće rekonstruisati kompletnu bazu podataka od fragmenata.
3. *Odvojenost* – bilo koji podatak sadžan u jednom fragmentu ne sme postojati ni u jednom drugom fragmentu (ovo normalno važi za horizontalnu fragmentaciju, ali ne isključuje vertikalnu fragmentaciju ako uzmemo u obzir replicirani primarni ključ može biti identifikator podataka umesto samih podataka).

Granularnost fragmenta se tipično izražava u kontekstu modela podataka koji se koristi. U relacionim bazama podataka, fragmenti mogu biti povezani sa kompletnim SQL *šemama* (takođe nazivanim *katalozima*) ili setovima njihovih tabela. Najmanja postignuta granularnost se obično naziva *horizontalnom* ili *vertikalnom* fragmentacijom gde ,,horizontalno’’ označava deljenje tabele po redovima a ,,vertikalno’’ – po kolonama. Primetimo da ova definicija fragmentacije ne isključuje posmatranje baze podataka kao jedan entitet ako je ovo potreban logički pogled na bazu podataka.

Nereplicirana, *particionisana* baza podataka sadrži fragmente koji su alocirani procesima baze podataka, normalno na odvojenim čvorovima klastera, sa samo jednom kopijom bilo kog fragmenta na celom klasteru. Takva šema nema jake HA (*High Availability*) sposobnosti. Da bi se postigla visoka dostupnost podataka, *replikacija fragmenata baze podataka* se koristi da bi se dozvolilo smeštanje i pristup podacima u veše od jednog čvora. U *potpuno repliciranoj* bazi podataka sama baza podataka postoji u svojoj celosti u svakom procesu baze podataka. U *parcijalno repliciranoj* bazi podataka fragmenti baze podataka su distribuirani kroz procese baze podataka na takav način da kopije fragmenta, odsad referencirane kao *replike*, mogu postojati u većem broju procesa baze podataka.

U replikaciji podataka, fragmenti se mogu klasifikovati kao *primarne* replike (Primari (eng. *Primary*)) ili kao *sekundarne* replike (Sekundari (eng. *Secondaries*)). Primarne replike predstavljaju *pravi* fragment podataka (ako primarna replika nije dostupna onda fragment nije dostupan, stoga baza podataka nije dostupna) i mogu se čitati u menjati. Sekundarne replike se uglavnom mogu samo čitati (eng. *read-only*) i manje-više su ažurirani u odnosu sa primarnom replikom. Sekundarne replike se mogu unaprediti u primarne replike u toku procesa (eng.) *failover*.

## Veze kardinalnosti između primarnih i sekundarnih replika

### 1\*Primar/1\*Sekundar

Ovde svaki fragment ima tačno jednu primarnu repliku koja je replicirana na tačno jednu sekundarnu repliku. Ovo je veoma čest model redundanse pošto je uviđeno da su dve replike adekvatne za postizanje visoke dostupnosti u većini slučajeva.

### 1\*Primar/Y\*Sekundar

Ovde svaki fragment ima tačno jednu primarnu repliku koja je replicirana u veći broj sekundarnih replika. Ovaj model pruža veću dostupnost of *1\*Primar/1\*Sekundar* i dozvoljava veću pristupačnost radi čitanja ako je dozvoljeno čitanje sa sekundarnih replika.

### 1\*Primar

Ovde svaki fragment postoji u tačno jednoj primarnoj replici. Ovaj model ne pruža nikakvu redundansu na nivou baze podataka. Redundansa je pružena bazi podataka od strane nižih slojeva (skladištenja – eng. *storage*).Koristi se u sistemima sa deljenim diskom i takođe u centralizovanim ili particionisanim bazama podataka.

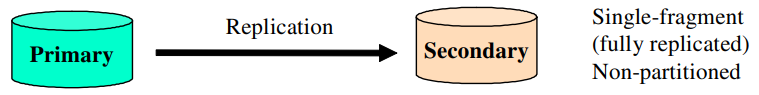
### X\*Primar

Ovde svaki fragment ima veći broj primarnih replika i koristi se *N\*Aktivan* model redundanse procesa (može se nazvati i *multi-master*). Ovaj model dozvoljava veći broj pristupa radi čitanja i upisa od *1\*Primar* ako se pokušaj upisa u isti fragment ne desi u paraleli (jer bi ovo dovelo do konflikta upisa).

## Veze između baze podataka i fragmenata

### Neparticionisana replicirana baza podataka

Najčešći slučaj je kada su baza i fragment jedno isto. Posledično, cela baza se replicira u sekundarnu lokaciju (Ilustracija 1). Primetimo: Svi slučajevi u ovoj podsekciji su ilustrovani pretpostavljajući kardinalnost *1\*Primar/1\*Sekundar*.

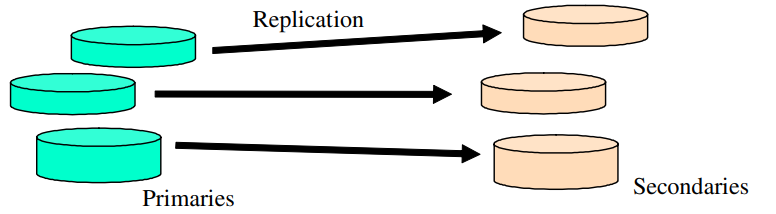


Ilustracija 1. Neparticionisana baza podataka

### Particionisana replicirana baza podataka

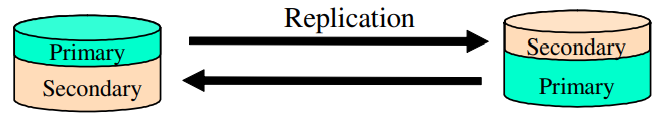
U ovom modelu, postoje fragmenti čija je svrha biti alociran na različitim čvorovima ili biti repliciran na različitim čvorovima (Ilustracija 2).

### Mešani replicirani fragmenti



Ilustracija 2. Particionisana baza podataka

Specijalan slučaj particionisane baze podataka je baza podataka sa mešanim (eng. *Mixed*) particijama gde jedna baza može sadržati (hostovati – eng. *Host*) i primarne i sekundarne fragmente. Specijalan slučaj su dve baze podataka sa simetričnim fragmentima (Ilustracija 3).

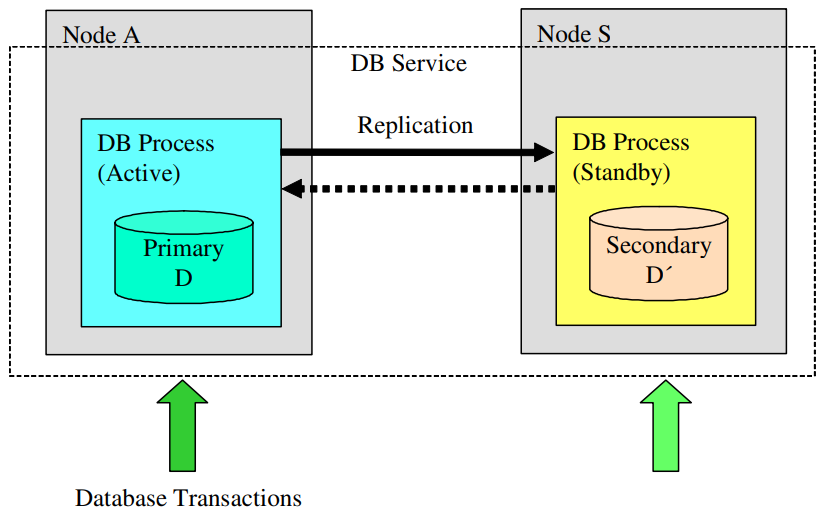


Ilustracija 3. Dve baze podataka sa simetričnim fragmentima

# Modeli redundanse procesa

## Aktivan/Na čekanju (2N)

*Aktivan/Na čekanju* (ponekad referenciran kao 2N) je model redundantnosti procesa za HA-DBMS-ove koji je podržan od strane i *sistema replikacije* i *sistema deljenog diska*. Svaki *aktivan* proces baze podataka se *bekapuje* (čuva – eng. *Back up*) od strane procesa baze na drugom čvoru koji je *na čekanju*. U ilustraciji 4, prikazan je primer sistema replikacije, a u ilustraciji 5 je prikazan primer sistema deljenog diska. Sve izmene (eng. *Updates*) se moraju obaviti nad procesom aktivne baze podataka; one će biti propagirane procesom replikacije, ili preko deljenog diska, procesu baze podataka koja je *na čekanju*.

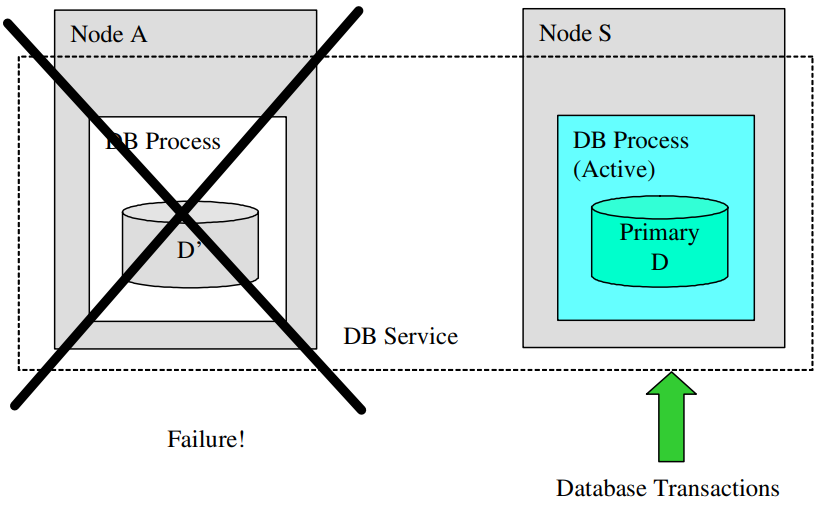


Ilustracija 4. Aktivan/Na čekanju model redundanse koristeći replikaciju

U slučaju da zakaže *aktivan* proces baze podataka (iz bilo kog razloga kao što je softverska greška na serveru baze podataka ili hardverska greška u čvoru – računaru) proces baze podataka koji je *na čekanju* će preuzeti i postaće novi aktivni proces baze podataka (Ilustracija 6). Ako se prethodno *aktivan* proces baze podataka koji je zakazao oporavi od problema, on će postati novi proces baze podataka koji je *na čekanju* i procesi baze podataka će potpuno promeniti uloge (Ilustracija 7). Ako HA baza podataka ima *preferiran aktivan* (eng. *preffered active*) proces baze podataka, može se kasnije prebaciti na originalnu konfiguraciju.



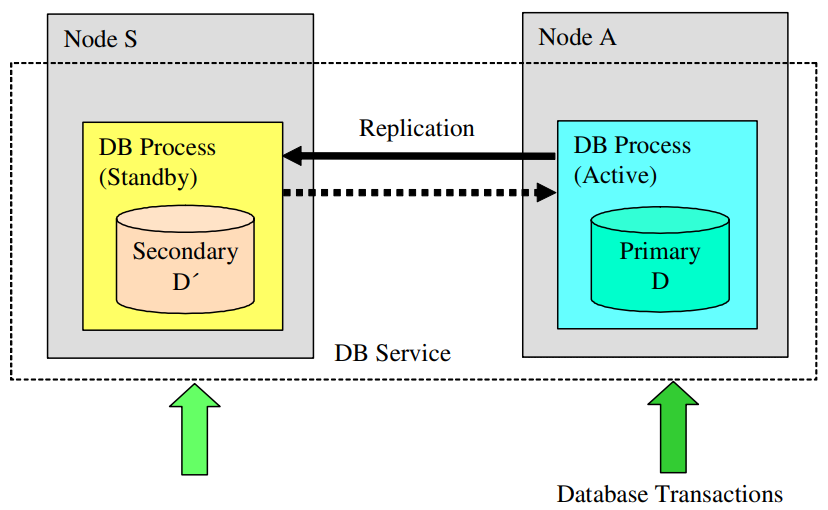
Ilustracija 5. Aktivan/Na čekanju model redundanse koristeći deljeni disk



Ilustracija 6. Zakaz na aktivnom primaru, prelazak sekundara u ulogu primara

Proces baze podataka koji je *na čekanju* će biti definisan kao manje-više spreman da preuzme u zavisnosti od odabranog nivoa sigurnosti (eng. *Safeness Level*) i HA zahteva aplikacija. Da bi se klasifikovali procesi baze podataka koji nisu *aktivni* pravimo razliku između *vruće na čekanju* (eng. *Hot Standby*) i *toplo na čekanju* (eng. *Warm Standby*).

### Hot Standby



Ilustracija 7. Obrnute uloge

Jedan proces baze podataka se *bekapuje* od strane procesa baze podataka koji je *na čekanju* koji je spreman manje-više odmah (vreme ~ ispod jedne sekunde) da preuzme u slučaju da aktivna baza podataka zakaže. Aplikacije već mogu biti konektovane na proces baze podataka koji je *na čekanju* ili trebaju biti ponovo konektovane na proces koji je *na čekanju* (sada već *aktivan*).

### Warm Standby

Jedan proces baze podataka se *bekapuje* od strane procesa baze podataka koji je *na čekanju* koji je spreman da preuzme nakon određene sinhronizacije/rekonektovanja sa aplikacijama u slučaju da *aktivna* baza podataka zakaže. U ovom slučaju, *failover* može trajati nekoliko desetina sekundi do nekoliko minuta.

U sledećoj sekciji ćemo uključiti *rezerve* (eng. *Spares*) i napravićemo razliku između procesa koji su *na čekanju* i *rezervnih* procesa, kako je moguće imati model *Aktivan/Na čekanju/Rezerva*.

## Aktivan/S\*Rezerva

*Aktivan/S\*Rezerva* (jedan *aktivan* i S *rezervi*) je konfiguracija u kojoj su nekoliko *rezervi* procesa baze podataka unapred konfigurisani na nekim čvorovima. Podržani su i sistemi sa replikacijom i sistemi sa deljenim diskom. Primer arhitekture sa *rezervom* na deljenom disku dat je ispod (Ilustracija 8).

U bazi podatka sa deljenim diskom, ako aktivan proces zakaže, deljeni disk koji sadrži datoteke baze podataka se *dodeljuje* (eng. *Mount*) čvoru koji hostuje *rezervu* (rezervni čvor), *rezerva* biva inicijalizovana bazom podataka, i baza podataka postaje aktivna na tom čvoru. Ako se čvor koji je zakazao restartuje on će postati novi *rezervni* čvor. Čvorovi su ovde znači potpuno zamenili uloge. Ako HA baza podataka ima *preferiran aktivan* proces baze podataka, kasnije će se prebaciti na originalnu konfiguraciju.



Ilustracija 8. Aktivan/S\*Rezerva model redundanse koristeći deljeni disk

U HA-DBMS-u sa replikacijom, *rezerva* dobija bazu podataka pre nego što postane *aktivna*. Nivo dostupnosti koji pruža ovaj model je niži od onog koji pruža model *Aktivan/Na čekanju* zbog dodatnog vremena potrebnog za inicijalizaciju *rezerve*.

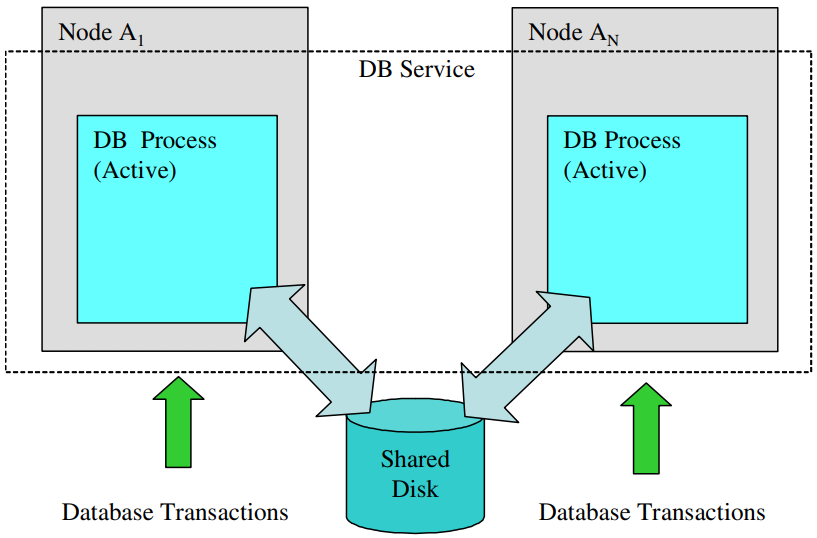
## N\*Aktivan

U većim klasterima, sistem baze podataka može iskorisitti više od dva čvora da bi bolje koristio moć procesiranja, memoriju, lokalne diskove i ostale resurse čvorova, da bi balansirao opterećenje na najbolji mogući način. U *N\*Aktivan* (referenciran i kao *N-Way Active*) modelu redundanse procesa, N procesa baza podataka su aktivne i aplikacije mogu pokretati transakcije na bilo kom procesu. Ovde svi procesi podržavaju jedni druge u slučaju zakaza i svaki može manje-više momentalno preuzeti. Sve *komitovane* (eng. *Committed*) izmene urađene nad jednim procesom baze podataka su dostupne drugim procesima i obrnuto. U sistemima sa deljenim diskom, svi procesi baze podataka vide jedan isti set izmena. U replikacionom sistemu, sve izmene se repliciraju na sve procese.

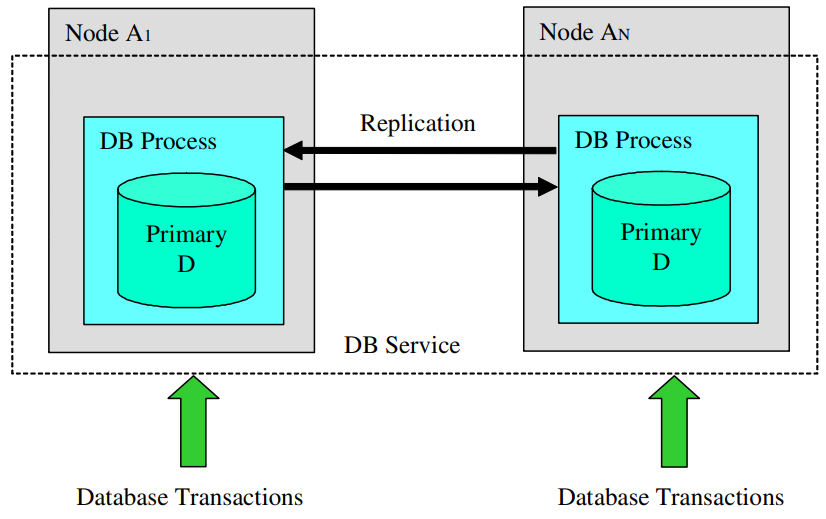
Baza podataka je potpuno dostupna iako svi procesi baze podataka i svi rekordi mogu biti ažurirani u svim procesima. U slučaju istovremenog konflikta izmena, konzistencija (verodostojnost – eng. *Consistency*) kopija može biti ugrožena, u sistemima sa replikacijom. Ovo se prevazilazi koristeći (eng.) *distributed concurrency control system* ili (eng.) *copy update reconciliation method*. U sistemu sa deljenim diskom, infrastruktura baze podataka može biti prostija zato što objekti podataka nisu replicirani. Baza podataka interno implementira (eng.) *lock manager* kako bisprečila konfliktna ažuriranja istih podataka. Ilustracija 9 prikazuje *N\*Aktivan* model zasnovan na deljenom disku. Primetimo da koristimo 2 čvora kao primer iako model podržava više od 2 čvora. Ilustracija 10 prikazuje *N\*Aktivan* model zasnovan na replikaciji sa 2 čvora.

Ako baza podataka nije potpuno replicirana i postoje mešani fragmenti u svim bazama podataka, model procesa je uvek *N\*Aktivan*. Npr. u Ilustracija 11, HA-DBMS sa *2\*Aktivan* modelom je prikazan kako koristi simetričnu replikaciju. Sa simetričnom replikacijom, problemi sa kontrolom konkurencije se izbegavaju a prednost nivelisanja opterećenja je zadržana.

Za razliku od većine *N\*Aktivan* okruženja, u Ilustracija 11, šema particionisanja je vidljiva aplikacijama i često je zasnovna na particionisanju opsega primarnih ključeva (eng. *Primary Key range*). Aplikacije su odgovorno za pristupanje pravom (potrebnom) aktivnom procesu. Inter-particijske (eng. *Inter-partition*) transakcije nisu podržane u normalnom slučaju.

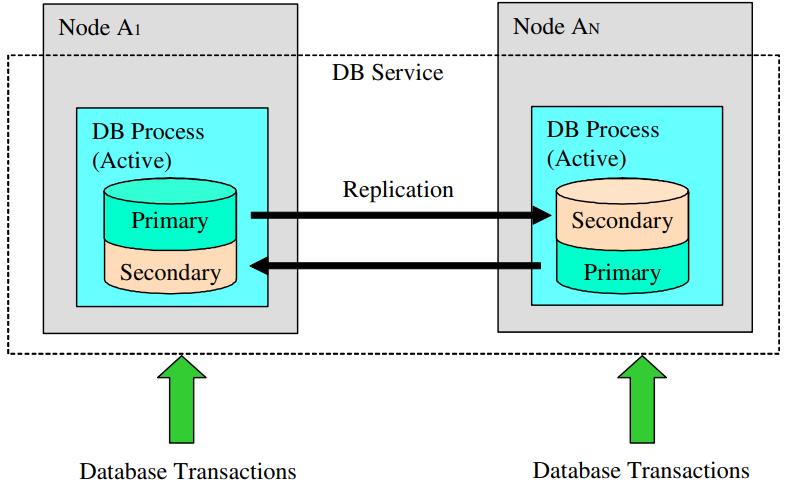


Ilustracija 9. N\*Aktivan, deljeni disk



Ilustracija 10. N\*Aktivan, potpuna replikacija

## N\*Aktivan/S\*Rezerva

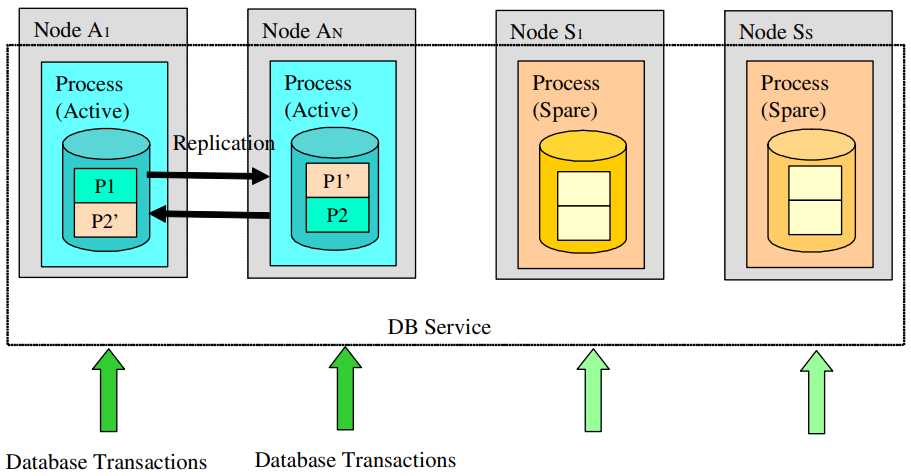


Ilustracija 11. 2\*Aktivan sistem sa simetričnom replikacijom baze podataka

*N\*Aktivan/S\*Rezerva* (eng. *N\*Active/S\*Spare*) – N *aktivnih* i S *rezervnih* procesa – je varijanta *Aktivan/S\*Rezerva* modela kada N *aktivnih* procesa baze podataka pruža dostupnost do particionisane baze podataka. Kao u modelu *N\*Aktivan*, *aktivni* procesi se mogu oslanjati na deljeni disk, mogu koristiti potpuno replicirane baze podataka ili mešane fragmente (parcijalno replicirane baze podataka – eng. *Partially replicated databases*).

Kao primer parcijalno replicirane baze podataka sa *rezervama* data je Ilustracija 12.

Svaki proces baze podataka održava neke fragmente bae podataka i *rezervni* procesi mogu preuzeti u slučaju greške kod *aktivnog* procesa baze podataka. *Rezerva* mora dobiti relevantne fragmente od *aktivnog* procesa prilikom pokretanja.



Ilustracija 12. N\*Aktivan/S\*Rezerva model redundanse, parcijalno replicirana baza podataka

## Drugi modeli redundanse

Neki sistemi kombinuju više modela redundanse da bi postigli drugačije nivoe redundanse podataka i procesa. *M-standby*, *cascading standby* i *geografski replicirani N\*Aktivan klasteri* (eng. *Geographically replicated N\*Active clusters*) su neki primeri. Kako su oni sadržani od drugih modela redundanse koji su ovde već prezentovani, neće biti dalje diskutovani.

# Pogled aplikacije

Aplikacije mogu i ne moraju biti svesne modela redundasi koje koriste razne komponente sistema baza podataka. U *Aktivan/Na čekanju* sistemia, aplikacije obično trebaju biti svesne tako da bi se mogle ponovo konektovati na *aktivnu* instancu. Štaviše, drugačiji aspekti sistema baza podataka mogu ustvari koristiti različite modele redundanse. Na primer, DBMS može imati jedan set procesa koji kontroliše podatke, a drugi set procesa koji izvršava upite i kojima se klijentske aplikacije obraćaju. ovi setovi procesa mogu imati totalno drugačije modele redundanse, i mogu komunicirati međusobno na različite načine.

Srodna tema je šema particionisanja. Generalno, bolje je sakriti particionisanje fragmenata od aplikacije. Ovo dozvoljava da aplikacija ne mora trpeti izmene ako se promeni šema particionisanja zbog planirane ili neplanirane rekonfiguracije. Držanje šeme particionisanja internom serveru baze podataka dozvoljava interno balansiranje opterećenja i reorganizaciju podataka bez uticanja na aplikacije. Iz razloga performansi neki sistemi pružaju neki koncept lokaliteta i podržavaju kolociranje aplikacije i podatke na istom čvoru. Ovo ponekad može biti kontrolisano kroz ,,naznake’’ (eng. *Hints*) od strane aplikacije ka serveru baze podataka o tome gde je najbolje da se podaci efektivno skladište. Šeme logičkog particionisanja za aplikacije i podatke se često kombinuju sa opštim šemama balansiranja opterećenja (eng. *Load balancing*) koje su ugrađene u *stekovima distribuirane komunikacije* (eng. *Distributed communication stacks*).

# SQL Server Always On Availability Groups

SQL Server Always On Availability Groups (skraćenica AG) je Microsoft-ovo rešenje za HA-DBMS ili za skaliranje čitanja (eng. *read-scale*). U ovom radu će se najviše obratiti pažnja na HA-DBMS koncept SQL Server AG.

Funkcionalnost je dostupna od verzije SQL Server 2012 (11.x), pa nadalje, sa unapređenjima u svakoj verziji. Unapređenja u ovom konktekstu mogu biti potpuno nove funkcionalnosti – kao npr. u SQL Server 2019 (za razliku od SQL Server 2017) *Automatic seeding* opcija kao preferenca početne sinhronizacije primarne i sekundarnih baza, ili unapređene funkcionalnosti – kao npr. u SQL Server 2019 (za razliku od SQL Server 2017) povećanje broja mogućih replika koje mogu biti u modu za automatski *failover* sa 3 na 5. Dalja obrada teme će se odnositi na SQL Server 2019 (15.x) Enterprise. Naglašavamo *Enterprise* iz razloga što postoje ograničenja kod Standard verzije; malo o tome kasnije u odeljku *Licence*.

SQL Server AG podržava do 9 replika dostupnosti, od kojih 5 mogu podržavati automatski *failover*. Sve validne replike moraju biti smeštene na pojedinačnom čvoru u jedinstvenom *Windows Server Failover Clustering* (WSFC) klasteru. Moguće je pridružiti i baze koje nisu u WSFC-u, ali one samo mogu služiti za čuvanje konfiguracija; malo o tome kasnije.

U SQL Server AG tehnologiji, podrazumevaju se *Neparticionisana replicirana baza podataka* kao i *1\*Primar/Y\*Sekundar* (ili *1\*Primar/1\*Sekundar* ako uzmemo vrednost 1 za Y) modeli redundanse podataka. Takođe podrazumeva se repliciranje baza podataka na nivou redundanse podataka, mada je moguće koristiti AG tehnologiju u kombinaciji sa *shared-disk* rešenjima.

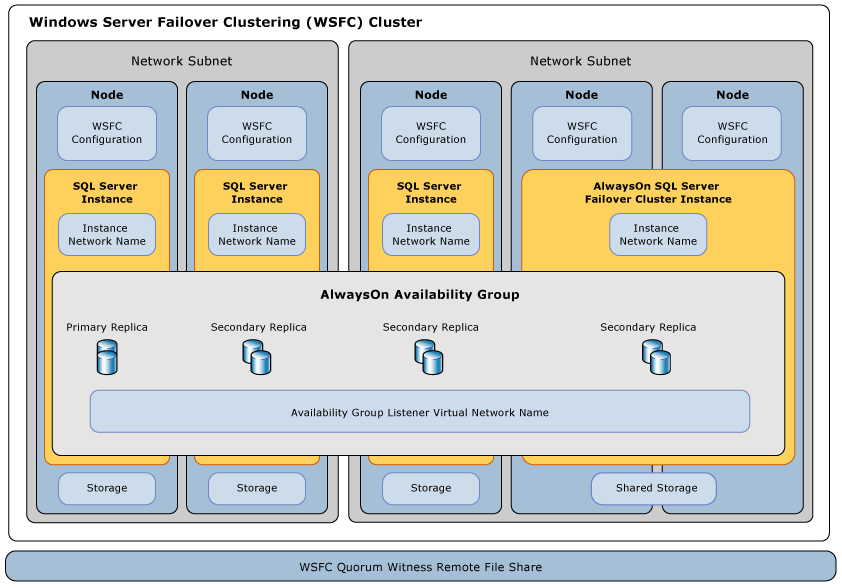
## Korišćeno pri implementaciji

Kako to nije tema ovog seminarskog rada, samo ćemo kratko objasniti tehnologije koje se koriste radi implementiranja *SQL Server Always On Availability Groups* funkcionalnosti.

*Windows Server Failover Cluster* (WSFC) je Microsoft tehnologija klasterovanja većeg broja nezavisnih servera u jedinstveni klaster. Osnovne komponente WSFC-a su:

* čvor (eng. *Node*) – server koji učestvuje u klasteru,
* resurs klastera (eng. *Cluster Resource*) – fizički ili logički entitet koji je u posedstvu čvora, može biti dostupan ili nedostupan, premeštan između čvorova,
* uloga (eng. *Role*) – kolekcija resursa klastera koja se ponaša kao jedan objekat klastera i pruža neku specifičnu funkcionalnost; u našem slučaju *Always On Availability Groups* (AG) ili *Always On Failover Cluster Instance* (FCI):
  + FCI uloga (*rola*) sadrži resurs IP adrese (eng. *IP address*), resurs imena na mreži (eng. *Network Name*) i resurs *SQL Server*-a,
  + AG uloga sadrži resurse: AG resurs i, ako je konfigurisan *listener*, resurse imena na mreži i IP adrese,
* resurs imena na mreži (eng. *Network Name* *Resource*) – logičko ime servera kojim se upravlja kao resursom klastera; mora biti korišćen uz resurs IP adrese; ovaj resurs možda zahteva možda zahteva korišćenje *Active Directory Domain Services* (AD) i/ili *Domain Name Service* (DNS),
* zavisnost resursa (eng. *Resource Dependency*) – resurs od kojeg zavisi neki drugi resurs
* preferiran vlasnik (eng. *Preffered Owner*) – čvor na kojem se preferira da određeni resurs bude pokrenut; svaki resurs ima defiinisanu listu preferiranih vlasnika koji su sortirani po prioritetu; u toku automatskog *failover*-a, grupa resursa se premešta na sledeći preferirani čvor iz liste,
* mogući vlasnik (eng. *Possible Owner*) – sekundarni čvorovi na kojima resurs može biti pokrenut; svaka grupa resursa se vezuje za listu mogućih vlasnika; uloge (*role*) mogu *failover*-ovati samo na čvorove koji su u ovoj listi,
* mod kvoruma (eng. *Quorum Mode*) – konfiguracija kvoruma u *failover* klasteru koji određuje broj neispravnih čvorova koji klaster može da izdrži,
* forsiran kvorum (eng. *Force Quorum*) – proces pokretanja klastera iako je samo manjina elemenata potrebnih za kvorum u komunikaciji.

Ilustracija 13 pokazuje primer izgleda jednog WSFC klastera sa 5 čvorova u dve podmreže i jednim FCI unutar sebe organizovanih kao 4 SQL Server instanci. Takođe je dostupan i *svedok kvoruma* (eng. *Quorum Witness*).



Ilustracija 13. WSFC

Svedok kvoruma (eng. *Quorum Witness*) predstavlja dodatni glas pri glasanju čvorova. Naime, radi donošenja odluke stanja (da li je klaster ispravan ili ne) potrebno je da svi čvorovi glasaju. Ako je broj glasova veći ili jednak modu kvoruma (pomenut ranije), onda je klaster ispravan, a ako je manji onda je klaster neispravan. Odnosno, paran broj (obično 2) nije dobar izbor za broj glasova, jer će biti nemoguće odlučiti stanje. Zato se uvodi treći, svedok, koji će presuditi u glasanju glasajući uvek za jednu stranu (obično pozitivnu – da je klaster u redu). Svedok kvoruma može biti deljeni direktorijum (eng. *Shared Directory*) ili deljeni disk (eng. *Shared Disk*), mada se preporučuje da to bude deljeni direktorijum. U našem slučaju, pratimo preporuku.

*Always On Failover Cluster Instance* (FCI) je grupa resursa, odnosno uloga (*rola*) koja grupiše više servera u jedan klaster. FCI i AG (*Always On Availability Groups*) se obično koriste u paru, ali mogu se koristiti i odvojeno. FCI grupiše više servera u klaster u cilju povećanja dostupnosti ili skalabilnosti sistema.

*Active Directory Domain Services* (AD) se koristi u cilju centralizovanog upraljanja dozvola (eng. *permissions*) na nivou domena, odnosno grupe servera na istom domenu. Sve u svemu, AD dozvoljava da na jednom mestu definišemo sve korisnike (eng. *Users*) i dozvole tih korisnika i koristimo ih na svim serverima koji su pridruženi tom domenu. AD se koristi i kao centralno mesto za DNS u okviru klastera.

Klaster treba biti umrežen. Najjednostavnija topologija je da svi čvorovi budu u jednoj podmreži (eng. *Subnet*) i često se ta najviše i koristi, ali moguće je i podeliti mrežu na više podmreža, a i dalje biti u mogućnosti napraviti klaster – to bi se uradilo, npr., ako su delovi klastera na različitim geografskim lokacijama.

Deljeni direktorijumi se ne koriste, naravno, samo u slučaju svedoka kvoruma. Oni se koriste i interno radi razmene informacija između čvorova klastera. Npr., koriste se pri inicijalizaciji baza sekundarnih čvorova kada se koristi *Full database and log backup* opcija.

Kako bi se koristila *Automatic seeding* opcija pri inicijalizaciji sekundarnih baza, neophodno je podesiti da direktorijumi baza i logova budu identični na svim čvorovima.

## Tipovi replika

Tipovi replika dostupnosti (eng. *Availability Replica Types*) se odnose na dva osnovna tipa replika: primarnu i sekundarnu. Obično u svim klasterovanim sredinama postoji samo jedna primarna replika i jedna ili više sekundarnih replika. AG funkcionalnost prati zadovoljava ovaj standard.

Primarna replika (eng. *Primary Replica*) je ona replika nad kojom se izvršavaju promene, odnosno replika na koju se klijenti (obično) vezuju kada se vezuju na klaster. To znači da ona uvek prva ima najnovije informacije i obično je najugroženija što se problema tiče.

Sekundarna replika (eng. *Secondary Replica*) je ona replika koja, logično, nije primarna, odnosno ona replika koja svoje izmene dobija od primarne replike u vidu sinhronizacije izmena. Sekundarna replika se može naći u dva oblika, kada se radi o AG: pasivna sekundarna replika i aktivna sekundarna replika.

Pasivna sekundarna replika je ona replika koja samo prima izmene od primarne replike i služi samo kao rezerva u slučaju problema na primarnoj replici.

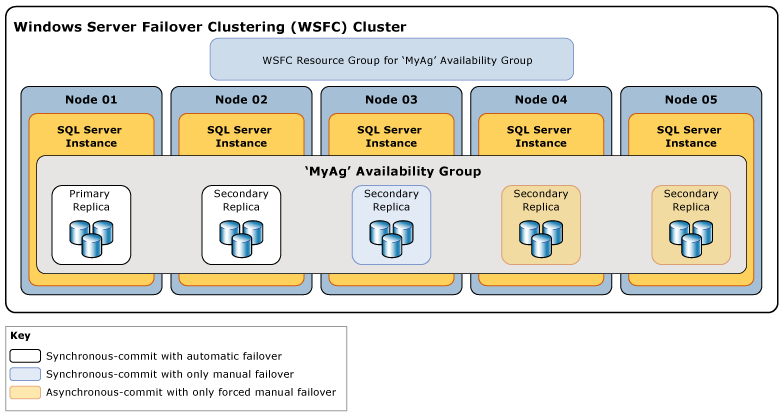
Aktivna sekundarna replika je ona replika koja ima dodatne funkcije osim samo rezerve (kao pasivna). Te dodatne funkcije mogu biti:

* + izvršavanje operacije *backup*-ovanja na sekundarnoj replici – kako se ne bi opterećivao primarni čvor čestim *backup*-ovanjem koje je neophodno raditi,
  + pristup samo radi čitanja (eng. *Read-Only Access*) na sekundarnoj replici – kako bi se paralelizovalo/skaliralo čitanje na svim replikama koje ovo omogućavaju (umesto da svi klijenti i čitaju sa primarne replike gde se pravi usko grlo).

## Modovi dostupnosti

Modovi dostupnosti (eng. *Availability Modes*) predstavljaju tipove dostupnosti koje AG može da pruža. Postoje 3 osnovna tipa/moda dostupnosti:

* *Asynchronous-commit mode* – je opcija koja se koristi za oporavak od katastrofe (eng. *disaster-recovery*); ako je sekundarna replika u ovom modu, primarna replika ne čeka da sekundarna replika potrvrdi da je odradila izmene pre nego što primarna replika potvrdi (eng.  *Commit*) izmene kod sebe; ovo stvara manji *overhead* komunikacije između primarne i sekundarnih replika (u ovom modu), ali dovodi do situacije da izmene u sekundarnim replikama kaskaju/kasne za primarnom. Jedini *failover* koji je podržan u ovom modu je *primoran failover* (eng. *Forced Failover*),
* *Synchronous-commit mode* – je opcija koja se koristi za najveću dostupnost; sekundarne replike u ovom modu ne dozvoljavaju primarnoj replici (odnosno ovaj protokol/opcija joj ne dozvoljava) da svoje izmene trajno potvrdi pre nego što dođe potvrda od svih sekundarnih replika (koje su u ovom modu) da su one potvrdile (obično se kaže i ,,utvrdile log’’ – eng. *Hardened Log*). Tipovi *failover*-a koji su podržani u ovom modu su *automatski failover­* (eng. *Automatic Failover*) i *ručni failover* (eng. *Manual Failover*),
* *Configuration only mode* – je opcija za AG koje nisu u WSFC-u; eplika koja je u ovom modu ne sadrži nikakve korisničke podatke; u konfiguracionom modu *master* baza ove replike samo čuva metapodatke (eng. *Metadata*) u vezi AG.



Ilustracija 14. Modovi dostupnosti

### Latencija podataka na sekundarnim replikama

Implementiranje *read-only* pristupa na sekundarnim replikama je korisno ako ta čitanja mogu da podnesu određenu latenciju/kašnjenje podataka. U nekim slučajevima je kašnjenje neprihvatljivo, pa je bolje koristiti sva čitanja nad primarnom replikom uprkos opterećivanju sistema.

Primarna replika šalje rekorde logova promena (eng. *log records of changes*) na primarnoj bazi podataka sekundarnim replikama. Na svakoj sekundarnoj bazi podataka, specijalno određena nit odrađuje promene koje su pristigle. Na sekundarnoj bazi kod koje je omogućeno čitanje, promene učinjene nad bazom nisu vidljive dok se transakcija ne potvrdi (eng. *Commit*) na primarnoj bazi podataka. Ovo praktično znači da postoji određeno kašnjenje, obično veličine nekoliko sekundi, između primarne i sekundarnih replika. U uobičajenim slučajevima, npr. kada problemi sa mrežom smanje protok podataka, latencija može postati značajnija. Latencija se povećava kada se dese uska grla na U/I i kada podaci ne mogu više da se kreću u sistemu. Da bi se pratili problem kretanja podataka u sistemu, može se koristiti *Always On Dashboard* ili pogled (eng. *View*) *sys.dm\_hadr\_database\_replica\_states\_dynamic\_management*.

## Tipovi failover-a

U SQL Server ­*Always On Availability Groups* postoje 3 tipa *failover*-a:

* **Planiran ručni *failover*** (eng. *Planned Manual Failover*)
  + Ručni *failover* se dešava kada administrator sistema eksplicitno zada *failover* komandu sistemu i izazove da sinhronizovana sekundarna replika promeni svoju ulogu u primarnu repliku, prethodno primarna replika preuzme ulogu sekundarne replike. Ovo se dešava sa garantovanom zaštitom podataka. Ovaj tip *failover*-a zahteva da i primarna i sekundarna replika budu u *synchronous-commit* modu i sekundarna replika mora biti sinhronizovana.
* **Automatski *failover*** (eng. *Automatic Failover*)
  + Automatski *failover* se dešava kao odgovor na problem/zakaz u primarnoj replici i uzrokuje promenu da sekundarna replika pređe u ulogu primarne replike. Kada se prethodna primarna replika oporavi od problema, ona prelazi u ulogu sekundarne replike. Ovo se dešava sa garantovanom zaštitom podataka. Automatski *failover* zahteva da i primarna i sekundarna replika budu u *synchronous-commit* modu i da je tip *failover*-a podešen na ,,*Automatic*’’. Dodatno, sekundarna replika mora biti sinhronizovana, imati WSFC kvorum i podržati fleksibilnu *failover* polisu ove AG.
* **Forsiran *failover*** (eng. *Forced Failover*)
  + Forsiran *failover* je jedini tip *failover*-a dostupan kada je replika u *asynchronous-commit* modu. Pun naziv je forsiran ručni *failover* (eng. *Forced Manual Failover*) i ne daje garanciju na zaštitu svih podataka. Ovaj tip *failover*-a može se desiti samo eksplicitnom naredbom od strane administratora sistema. Korisi se samo za oporavak od katastrofe i jedini je tip *failover*-a koji dozvoljava da sekundarna replika ne bude potpuno sinhronizovana sa primarnom replikom.

## Konekcije klijenata

Da bi se najlakše pružio pristup primarnoj bazi podataka određene AG klijentskim aplikacijama potrebno je kreirati (eng.) *Group Listener* koji će opsluživati zahteve. *Availability group listener* pruža set resursa koji je pridružen datoj AG da usmerava klijentske konekcije odgovarajućoj replici.

*Availability group listener* je asociran unikatnim DNS nazivom koji služi kao *virtual network name* (VNN), jednom ili više virtuelnim IP adresama (VIP) i TCP portom.

U okviru konfiguracije AG moguće je konfigurisati *Read-Only Routing* opciju koja dozvoljava balansiranje opterećenja prilikom čitanja. Ovo koristi osobinu aktivne sekundarne replike da se dozvoli čitanje replike (ako je osobina omogućena) i dozvoljava čitanje sa više lokacija umesto samo sa jedne (primarne) što povećava izlaznu moć čitanja sa klastera.

U nastavku data je tabela () koja pokazuje ishod konekcije ka bazi u zavisnosti na koju se ulogu replike, tip konekcije koji je podržan na replici i namere konekcije (eng. *Connection Intent*).

| **Replica Role** | **Connection Access Supported on Replica** | **Connection Intent** | **Connection-Attempt Result** |
| --- | --- | --- | --- |
| Secondary | All | Read-intent, read-write, or no connection intent specified | Success |
| Secondary | None (This is the default secondary behavior.) | Read-intent, read-write, or no connection intent specified | Failure |
| Secondary | Read-intent only | Read-intent | Success |
| Secondary | Read-intent only | Read-write or no connection intent specified | Failure |
| Primary | All (This is the default primary behavior.) | Read-only, read-write, or no connection intent specified | Success |
| Primary | Read-write | Read-intent only | Failure |
| Primary | Read-write | Read-write or no connection intent specified | Success |

## Period isteka timeout-a sesije

*Timeout* period sesije je osobina replike AG koji određuje koliko dugo konekcija sa drugom replikom može biti neaktivna pre nego što je konekcija zatvorena. Primarna i sekundarne replike se međusobno *ping*-uju signalima da su aktivne i izvršavaju svoje uloge. Prijem *ping*-a označava da je konekcija ,,živa’’ i resetuje se brojač *timeout* perioda sesije.

*Timeout* sprečava beksonačno čekanje na odgovor od druge replike. Kada period istekne, konekcija između te dve replike se označava kao zatvorena i sama replika koja se ,,ne javlja’’ se postavlja u *DISCONECTED* stanje. U tom stanju, čak i ako je replika podešena u *synchronous-commit* modu, transakcije neće čekati da se ta replika sinhronizuje.

Podrazumevana vrednost (eng. *default*) ovog *timeout* perioda sesije je 10 sekundi. Vrednost se može podešavati, s tim što je minimalna vrednost 5 sekundi, a preporučeno je da se nikad ne ide ispod 10 sekundi iz razloga što je ponekad replici (odnosno bazi replike) potrebno malo vremena da obradi neki težak zahtev i u tom slučaju je moguće doći do lažnog otkaza (eng. *false failure*).

## Automatsko popravljanje stranica

Još jedna korist *Always On Availability Groups* je parcijalni oporavak informacija baze. Naime, moguće je da se korumpira samo jedna stranica (eng. *Page*) u memoriji koja čuva neke informacije iz baze podataka – uzrok može biti npr. hardverska nesavršenost diska. U tom slučaju replika kod koje se desio problem će zatražiti od drugih replika samo tu stranicu i popraviti je ili premestiti na drugu lokaciju ako je ta lokacija na disku trajno nedostupna. Primarna replika će tražiti zamensku stranicu od svih sekundarnih replika i primeniće onu kopiju koju prvu dobije. Sekundarne replike će uvek tražiti zamensku stranicu od primarne replike.

## Oporavak replika

Nakon što replika padne (bilo primarna pa se desi *failover* ili jednostavno padne sekundarna replika) postavlja se pitanje kako podići nazad repliku da služi kao ispravna sekundarna replika. Ovo sve zavisi od tipa greške koja se dogodila, stanja u kojem su bile primarna/sekundarna replika pre pada i stanja u kojem je sada aktivna primarna replika. Pretpostavljamo da su sve ostale komponente sistema u redu, tipa hardverske komponente i veze i propratni softver; govorimo samo u kontekstu podataka koje replike poseduju. U slučaju da se pad desio kada je sekundarna replika (bilo ona koja je postala primarna posle pada primarne, ili sama sekundarna replika koja je pala) bila sinhronizovana vrlo je verovatno da će se nakon ponovnog podizanja replike sama sinhronizovati sa akrivnom primarnom replikom tražeći od nje transakcije koje su se možda desile u međuvremenu. Problem u ovom slučaju može da se desi ako je prošlo previše vremena i log transakcija je obrisan; tada je neophodna intervencija administratora. Takođe je najverovatnije potrebna intervencija administratora ako je replika bila nesinhronizovana sa primarnom bazom, odnosno ako se pad desio u nezgodnom trenutku sa stanovišta sinhronizacije podataka između replika.

Ručna intervencija predstavlja ili čišćenje celokupne replike (baza podataka) i ponovnog iniciranja ubacivanja te replike u AG ili jednostavan ručni *backup-restore* proces ako je manji problem.

## Licence

Ovde ćemo kratko napomenuti bitne razlike između SQL Server 2019 (15.x) *Enterprise* i *Standard* licenci. Druge licence nećemo pominjati kako nisu ni namenjene za realna *production* okruženja. *Standard* licenca je izuzetno ograničena što se tiče *Always On Availability Groups* funkcionalnosti u odnosu na *Enterprise* licencu (verziju). Ograničenja *Standard* licence:

1. Limitirana na 1 bazu podataka unutar jedne AG (kod *Enterprise* verzije nema tog ograničenja)
2. Limitirana na 2 replike unutar jedne AG (kod *Enterprise* verzije taj broj je 9)
3. Ne postoji mogućnost *read-only access* na sekundarnoj replici
4. Ne postoji mogućnost *backup-offload* na sekundarnoj replici
5. Nema GUI podešavanja – sve se mora konfigurisati T-SQL naredbama ili PowerShell-om

Ograničenje #1 samo primorava da se mora praviti više AG što ne zvuči previše ograničavajuće, ali zapravo pravi interni *overhead* kako se za svaku AG moraju pokretati posebni interni procesi i vršiti odvojena sinhronizacija.

Ograničenje #2 je zapravo najveće ograničenje što se tiče HA funkcionalnosti. Istina, za veliki broj realnih rešenja više od jedne sekundarne replike nije ni potrebno, ali ponovo smanjuje fleksibilnost umnogo.

Ograničenje #3 sprečava/gasi *load-balancing* funkcionalnost.

Ograničenje #4 ne dozvoljava delegiranje *backup* operacija na sekundarne replike, što može usporavati primarnu repliku.

Ograničenje #5 samo ograničava način rada administratora sa samim sistemom, odnosno ne možemo koristiti grafičke alate unutar *SQL Server Management Studio*-a (SSMS-a). Ono, samo po sebi, predstavlja samo ograničenje fleksibilnosti rada sa sistemom, a ne utiče na funkcionalnosti koje sistem pruža.

## Implementacija

U sklopu izrade ovog seminarskog rada, autor je napravio virtuelni klaster u VirtualBox alatu na svojoj lokalnoj mašini. Izrađen klaster je bio spor iz razloga što *host* mašina ima dvojezgarni procesor i stoga su sve virtuelne mašine delile jedno jezgro, dok je *host* mašina koristila sama drugo jezgro. Svakako je izrađena arhitektura poslužila probnoj i pokaznoj svrsi. Cilj pokusa je bio napraviti jednostavan klaster koji će odraditi automatski *failover* nakon pada jedne instance. Takođe proveriti da li je moguć ručni *failover*.

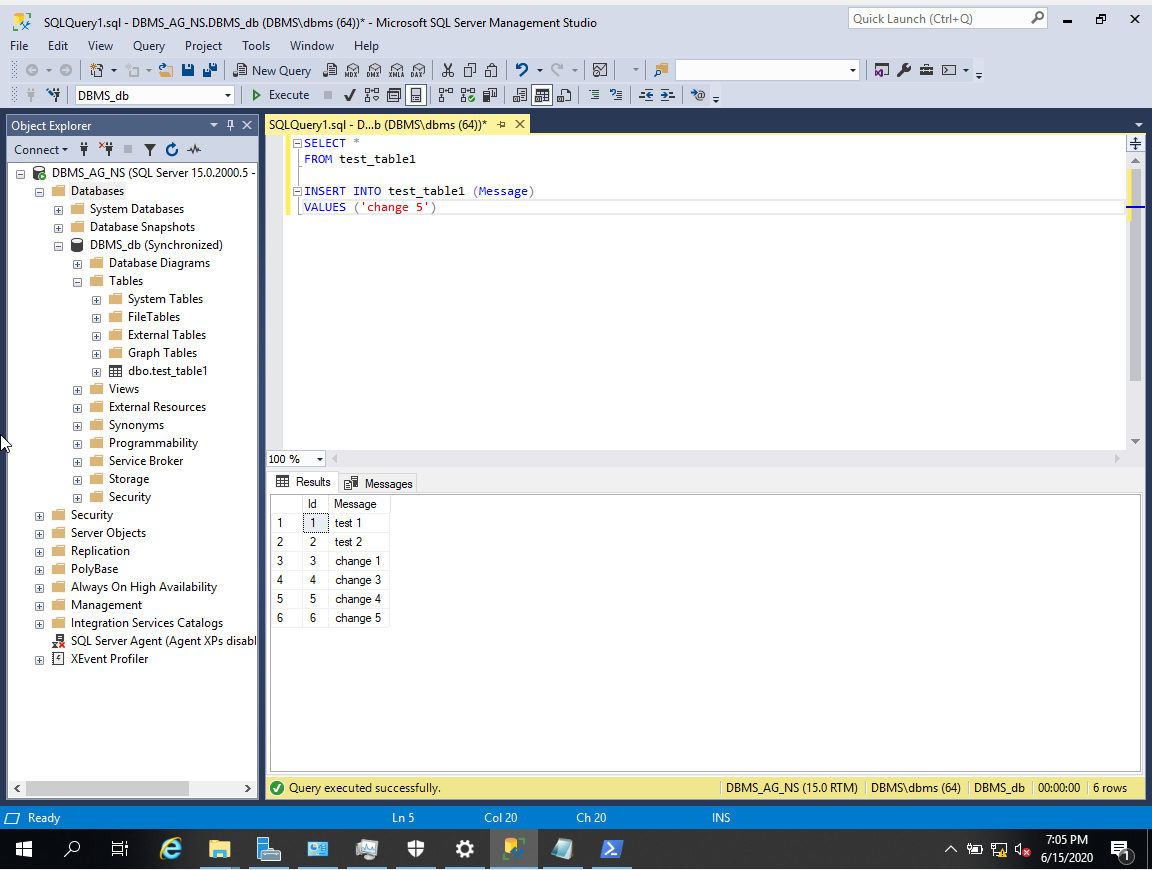
Napravljena arhitektura je sastavljena od 3 servera sa *Windows Server 2019 Enterprise Datacenter* operativnim sistemom.

* **DC1** server služi za hostovanje AD i DNS servisa i na njemu su definisani *quorum witness* *fileshare* i još jedan deljivi direktorijum koji čvorovi AG mogu da koriste radi unutrašnje sinhronizacije.
  + Naziv mašine: DC1
  + IP adresa: 192.168.255.100
  + AD: DBMS.local
  + Nalog (koji se koristi u svim čvorovima kasnije): dbms@DBMS.local
* **N1** server predstavlja jedan čvor FCI klastera. Na njemu je instalirana *SQL Server 2019 Developer* (15.0.2000.5) instanca i *SQL Server Failover Cluster* funkcionalnost.
  + Naziv mašine: N1
  + IP adresa: 192.168.255.101
  + Naziv SQL instance: SQL\_N1
* **N2** server predstavlja drugi čvor FCI klastera. Na njemu je instalirana *SQL Server 2019 Developer* (15.0.2000.5) instanca i *SQL Server Failover Cluster* funkcionalnost.
  + Naziv mašine: N2
  + IP adresa: 192.168.255.102
  + Naziv SQL instance: SQL\_N2

Podešavanje klastera:

* Naziv *Availability Group*-e: **DBMS\_AG**
* IP adresa: 192.168.255.99
* Kvorum: *Node Majority*
* AG *Listener*:
  + Naziv: DBMS\_AG\_NS
  + IP adresa: 192.168.255.98

Krajnji produkt: baza podataka dostupna preko AG *Listener*-a bez obzira na to koji čvor je primarni. Za svrhe testiranja je otvorena baza preko SSMS-a na DC1 serveru (koji nije čvor klastera):



Ilustracija 15. Krajnji produkt

Sa ilustracije Ilustracija 15 se vidi naziv preko kojeg je povezano na bazu, baza, tabela u bazi, kolone i podaci tabele i upiti koji su pokretani.

### Podešavanje okruženja

Ovde ćemo izlistati korake koji su odrađeni u kreiranju klastera. Delove koji nisu vezani direktno za DBMS ćemo samo izreći i nećemo davati detaljnija uputstva kako to ne ulazi u temu ovog rada.

Prvo je napravljen i podešen DC1 server. Instaliran *Windows*. Podešena IP adresa. instalirana *Server Rola* pod nazivom *Active Directory Domain Services*. Nakon toga je podešen ovaj server kao *Domain Controller* (otuda i naziv *DC1* – kao skraćenica). DC je podešen sa sposobnostima DNS (*Domain Name System*) i GC (*General Catalog*) i napravljen je *domain* DBMS.local. Nakon toga je napravljen *Domain Account* sa nazivom **dbms** i pridružen je roli *Domain Admins*.

Zatim je napravljen i podešen N1 server. Instaliran *Windows*. Podešena IP adresa. Pridružen je DBMS.local *domain*-u. Nakon toga je instaliran *SQL Server Instance* i *SQL Server Failover Clustering*.

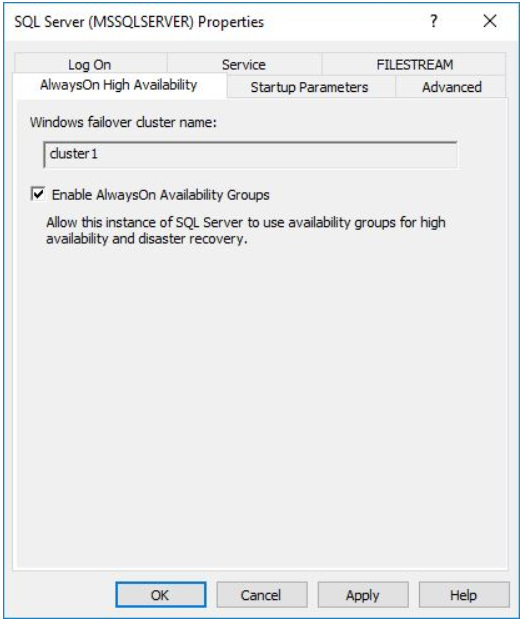
Zatim je napravljen i podešen N2 server (identično kao N1 samo sa svojim podešavanjima adresa i imena).

Nakon toga je instalirana *Server Role-*a na oba čvora (N1 i N2) pod nazivom *Failover Clustering*. U svrhe testiranja je na sve tri mašine ugašen *Firewall* kako bi se skratilo vreme podešavanja, a u realnom okruženju bi bilo potrebno i to pravilno podesiti otvaranjem određenih portova za ulazne i izlazne konekcije. Zatim je na prvom čvoru (N1) napravljen *Cluster* preko *Failover Cluster Manager*-a i dodata su oba servera (obzirom da su mogli serveri da vide jedan drugoga jer su u istoj mreži i na istom domenu). Podešen je naziv klastera, kvorum i IP adresa. Zatim je na DC1 serveru napravljen *FileShare* za kvorum svedoka i napravljen je *file share* kvorum svedok u klasteru koji referencira deljenu putanju na DC1 serveru. Klaster je ovime napravljen.

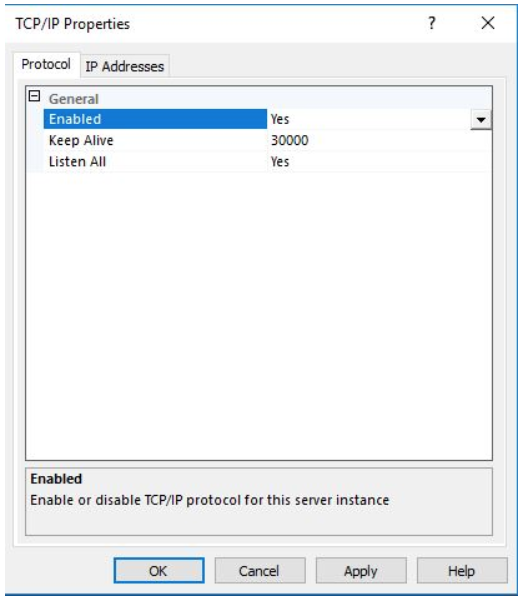
Nakon toga bilo je potrebno na oba servera omogućiti *AlwaysOn Availability Groups* i podesiti da su SQL Server instance dostupne na privatnoj adresi koja je dodeljena tom čvoru.



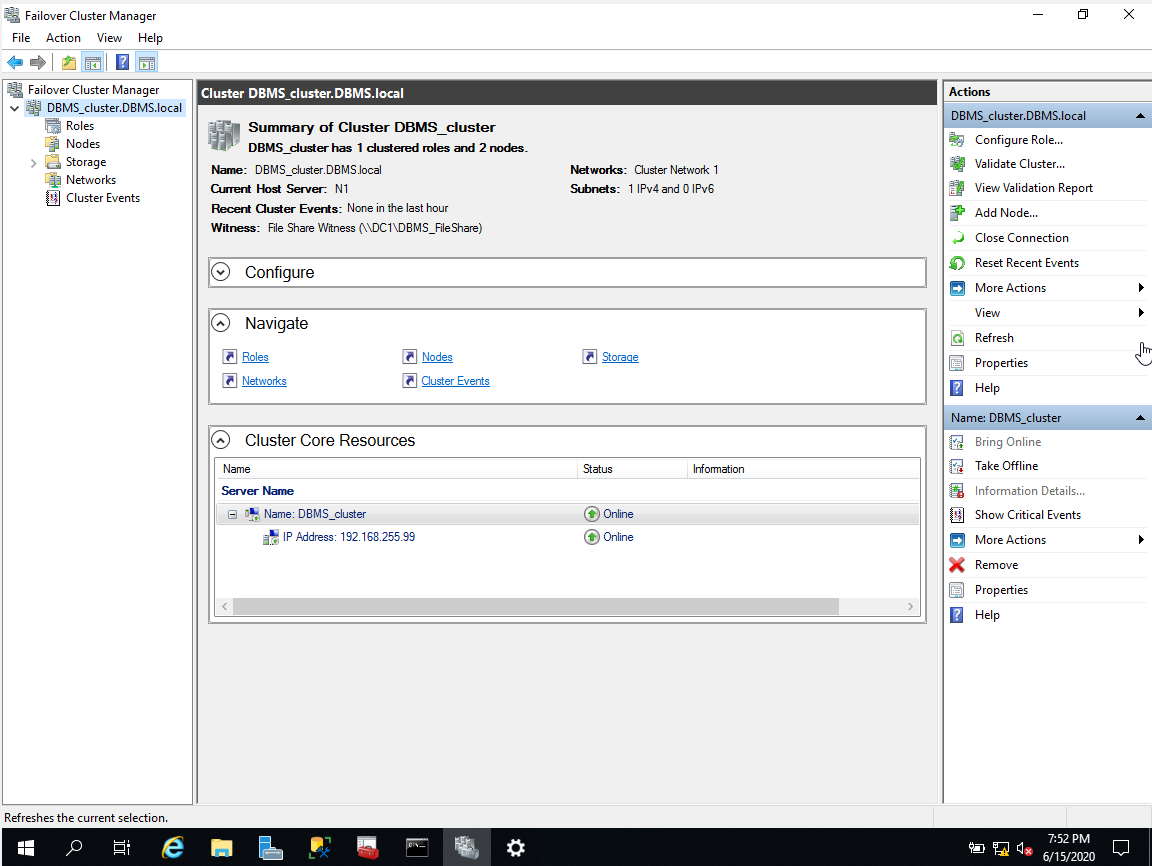
Ilustracija 16. Instaliranje SQL komponenti



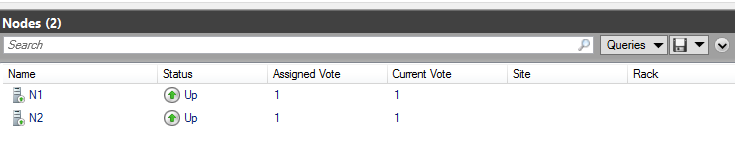
Ilustracija 17. Omogućavanje AlwaysOn AG



Ilustracija 17. Omogućavanje TCP/IP protokola na MSSQLSERVER-u



Ilustracija 18. Podešen klaster

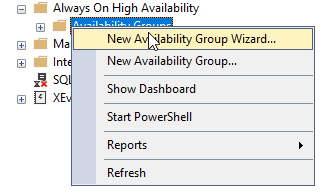


Ilustracija 19. Čvorovi u klasteru

### Kreiranje nove AG

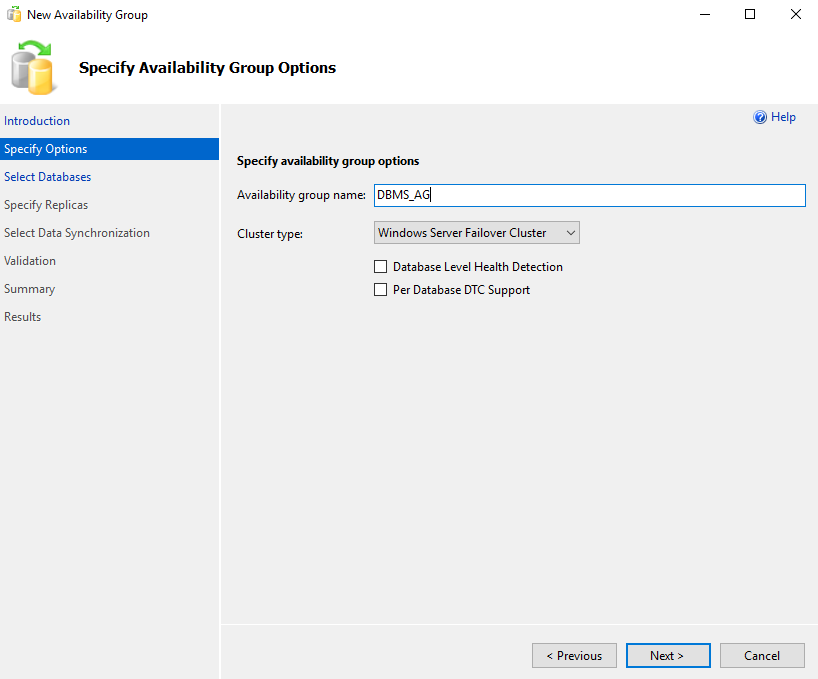
Sada ćemo proći kroz *Wizard* za kreiranje AG i kratko objasniti svaku opciju.

Prvo ide pokretanje *Wizard*-a (Ilustracija 20):



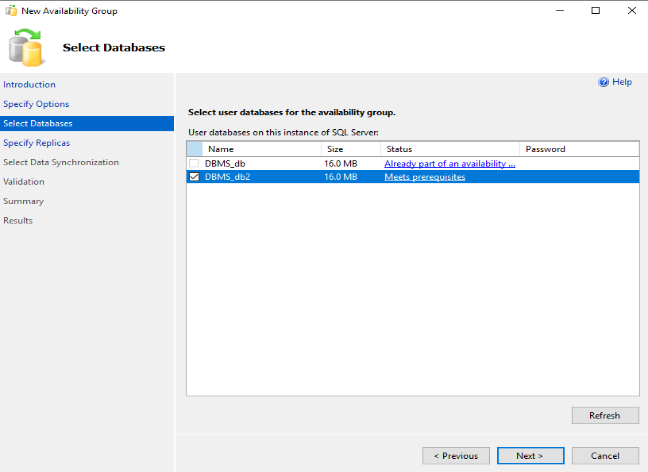
Ilustracija 20. Pokretanje AG Wizard-a

Zatim sledi dodeljivanje imena AG (Ilustracija 21):



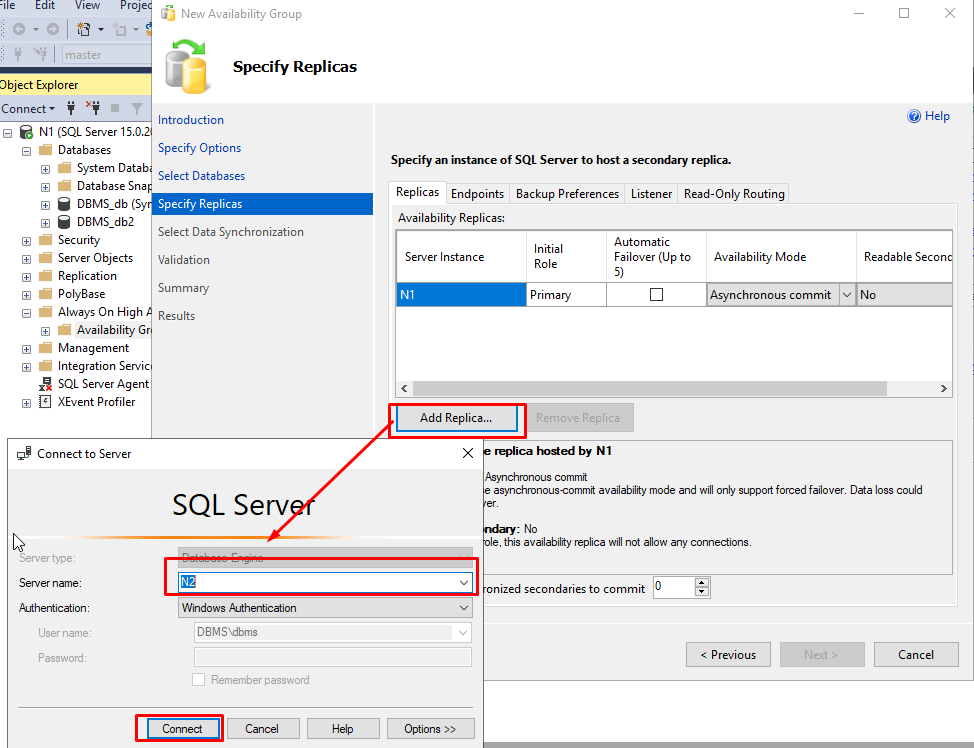
Ilustracija 21. Dodeljivanje imena AG

Nakon toga ide biranje baza koje će se replicirati (Ilustracija 22):



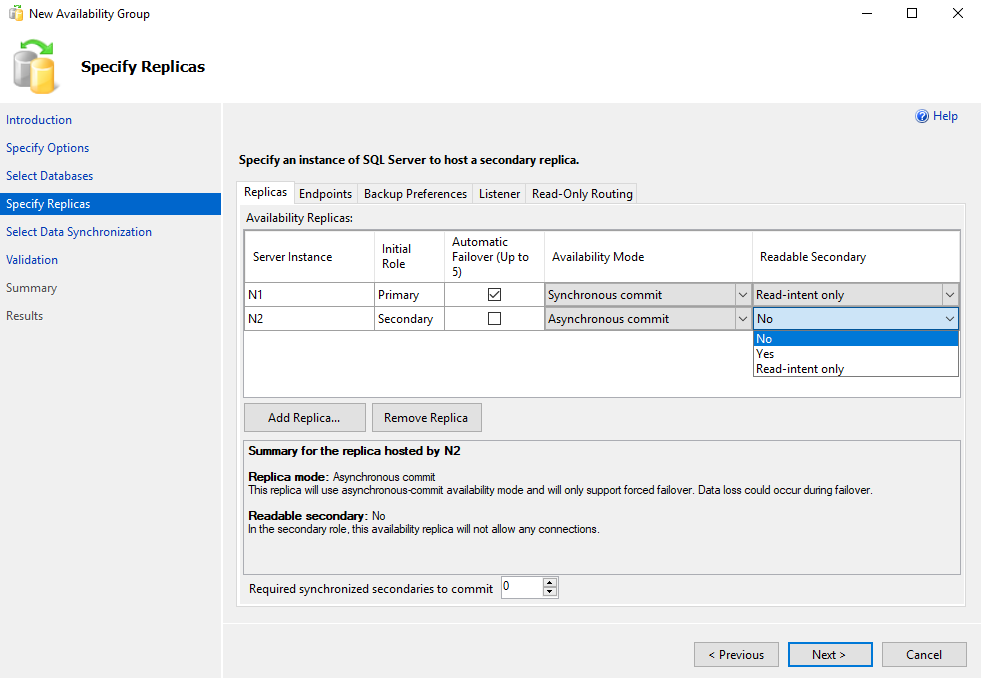
Ilustracija 22. Dodeljivanje baza AG

Nakon toga sledi dodavanje sekundarnih replika (Ilustracija 23):



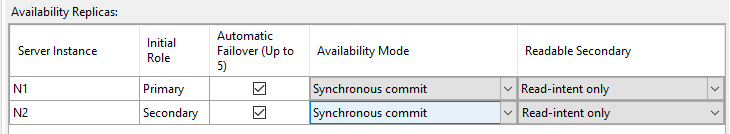
Ilustracija 23. Dodavanje nove replike

Nakon toga idu podešavanja replika (Ilustracija 24). Na prvom prozoru imamo 3 opcije za svaku repliku. Prva je da li je replika omogućena za automatski *failover*. Druga u kom modu sinhronizacije (mod dostupnosti) je svaki čvor (opcije: *asynchronous-commit* i *synchronous-commit*). Treća opcija je da li replika podržava čitanje u ulozi sekundara (tri opcije: *No*, *Yes* i *Read-intent only*). Sve ove opcije su već objašnjavane.



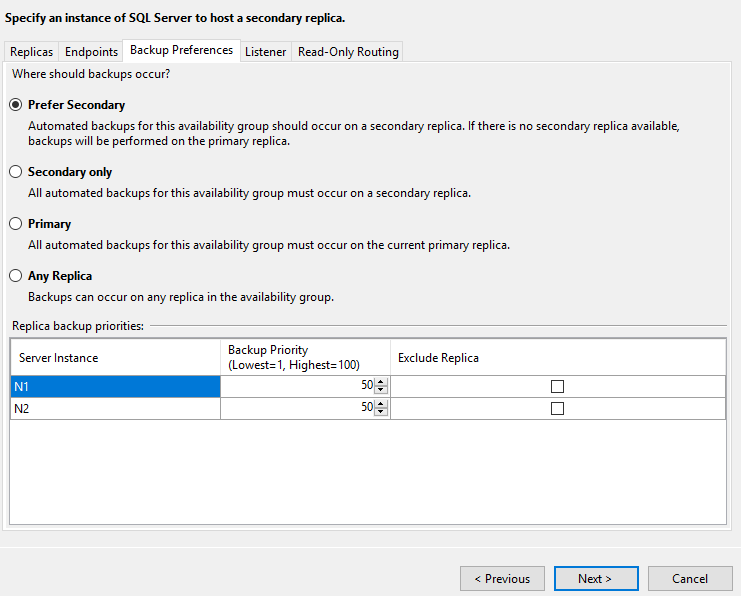
Ilustracija 24. Podešavanje replika. Opcije odabrane zbog prikaza svih opcija.

Naše replike su zapravo podešene ovako (Ilustracija 25):



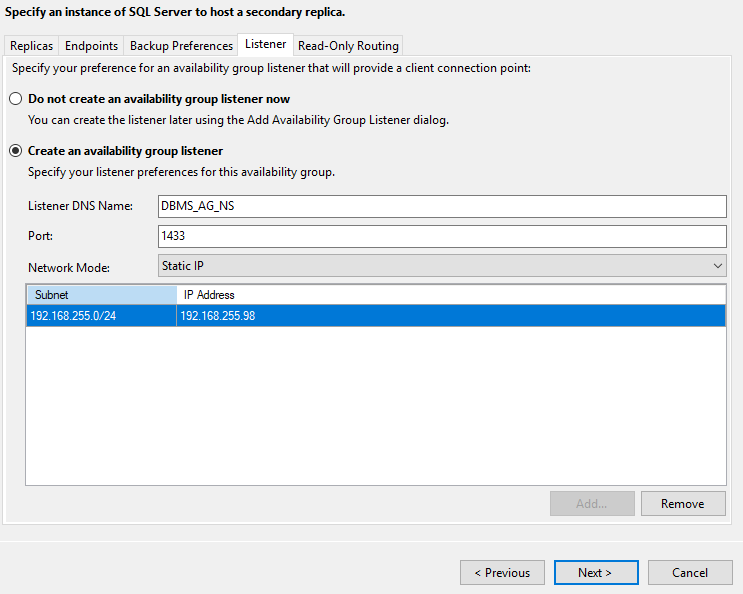
Ilustracija 25. Naše podešavanje replika

Sledeća opcija koja se podešava je *Backup Preferences* (Ilustracija 26). Ovde biramo gde želimo da se odrađuju automatski *backup*-ovi baze.



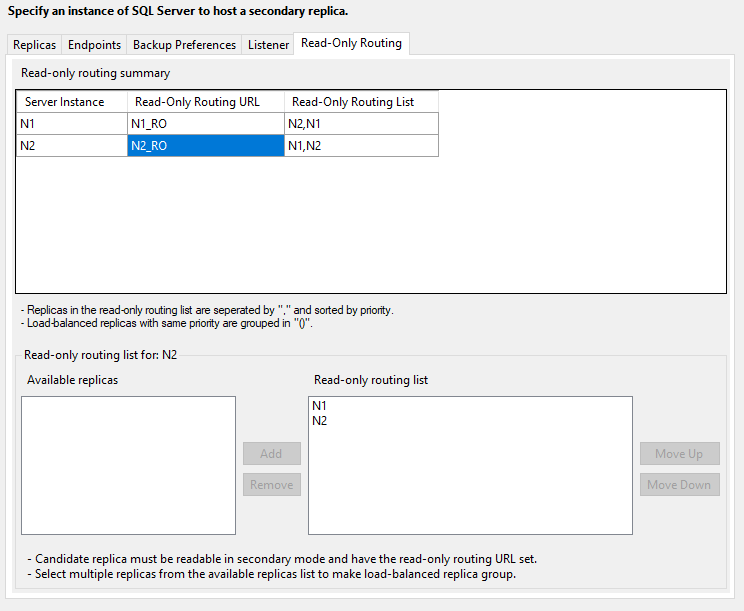
Ilustracija 26. Backup preferences

Na sledećem tabu podešavamo AG *listener* koji će služiti kao centralna logička lokacija za pristup bazama podataka koje su u AG-u (Ilustracija 27):



Ilustracija 27. AG listener

Na poslednjem tabu podešavanja replika se podešava *Read-Only* rutiranje (Ilustracija 28). U našem slučaju, podešeno je da ako je N1 primarna replika *read-intent* zahtevi će se slati prvo na N2 zatim na N1, a u slučaju da je N2 primarna replika *read-intent* zahtevi će se slati prvo na N1 zatim na N2.



Ilustracija 28. Read-only routing

Poslednje što je moguće podesiti je način početne inicijalizacije/sinhronizacije sekundarnih replika (Ilustracija 29). Postoji 4 opcije:

1. *Automatic seeding* – automatsko skriptanje baza na primarnoj replici i primenjivanje skripti na sekundarnim replikama. Brz i pouzdan način za manje baze. Jedino ograničenje je što sve replike moraju smeštati fajlove svojih baza i logova na apsolutno identičnim lokacijama. Ovo je opcija koja je dostupna od SQL Server 2019 verzije.
2. *Full database and log backup* – automatski *backup* na primarnoj replici i *restore* na sekundarnim replikama. Pogodan način za automatsku inicijalizaciju ako su velike baze u pitanju. Ograničenje je što mora postojati neka *file share* lokacija koja je dostupna svim čvorovima kako bi tamo primarna replika smestila svoj *backup* baza, a sekundarne replike odatle pokupile isti *backup* i primenile ga na svojoj instanci.
3. *Join only* – opcija koja se koristi ako su baze već postojeće na čvorovima, bilo ručnom intervencijom ili kao ostatak iz prethodnih AG koje su možda obrisane.
4. *Skip initial data synchronization* – opcija koja u potpunosti preskače inicijalnu sinhronizaciju bez obzira da li baze postoje ili ne postoje na sekundarnim replikama. Podrazumeva da će AG biti napravljen sa početnim greškama sve dok administrator naknadno ne sinhronizuje sve potrebne baze ručno.

Nakon svih ovih koraka sledi korak *Validation* koji samo proverava da li su prethodni koraci pravilno ispunjeni. Nakon njega ide *Summary* korak koji pokazuje šta će biti odrađeno. Na kraju je prozor *Results* koji pokazuje šta je odrađeno i da li je bilo uspešno. Kada se ovo uspešno završi, imamo kreiran AG.

Napomena: Sve ove korake i definicije smo mogli odraditi kroz T-SQL naredbe, ali nismo ovde dali primere za iste jer je nepotrebno zbog povoljnosti korišćenja *Enterprise* verzije (u našem slučaju *Developer*, ali ona se ponaša kao *Enterprise*) SQL Server-a. Da koristimo *Standard* verziju, morali bismo koristiti T-SQL naredbe.



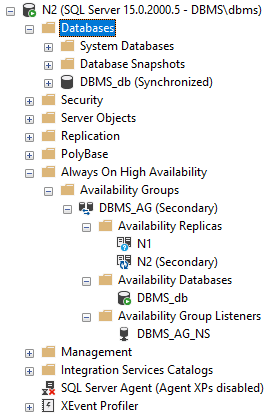
Ilustracija 29. Inicijalna sinhronizacija podataka

### Failover

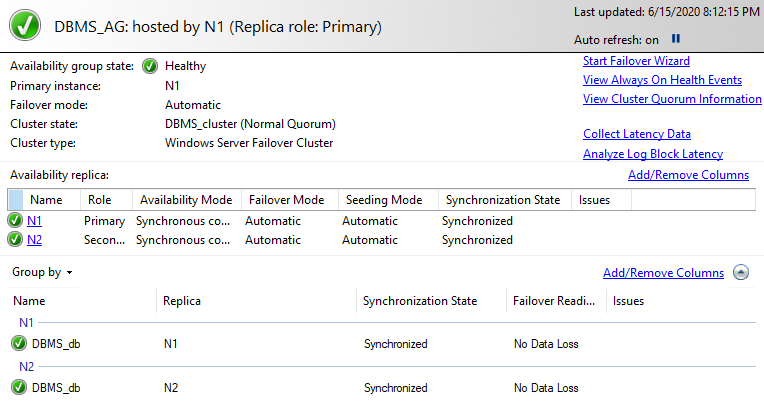
Simulacija otkaza na primarnom čvoru odrađena je jednostavnim pauziranjem virtuelne mašine. U pauziranom/suspendovanom stanju, virtuelna mašina kao takva ne funkcioniše i praktično nestaje svaki njen trag u mreži i tom virtuelnom okruženju. Prvo će situacija biti opisana, zatim date ilustracije stranja. Primetimo da N1 ima još jednu bazu pod nazivom DBMS\_db2 ali ona se ne replicira u ovoj AG pa je apsolutno nebitna.

Situacija pre ,,otkaza’’ je bila sledeća (Ilustracija 30 i Ilustracija 31):

* N1 je primarni čvor
* N2 je sekundarni čvor
* N1 i N2 su podešeni u *synchronous-commit* modu i treba se desiti automatski *failover* na čvor N2 nakon ,,otkaza’’ čvora N1

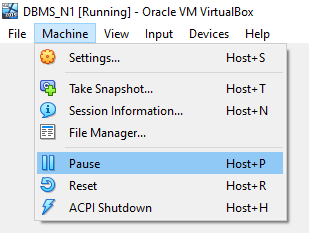


Ilustracija 30. Stanje na N1 i N2 čvorovima pre otkaza



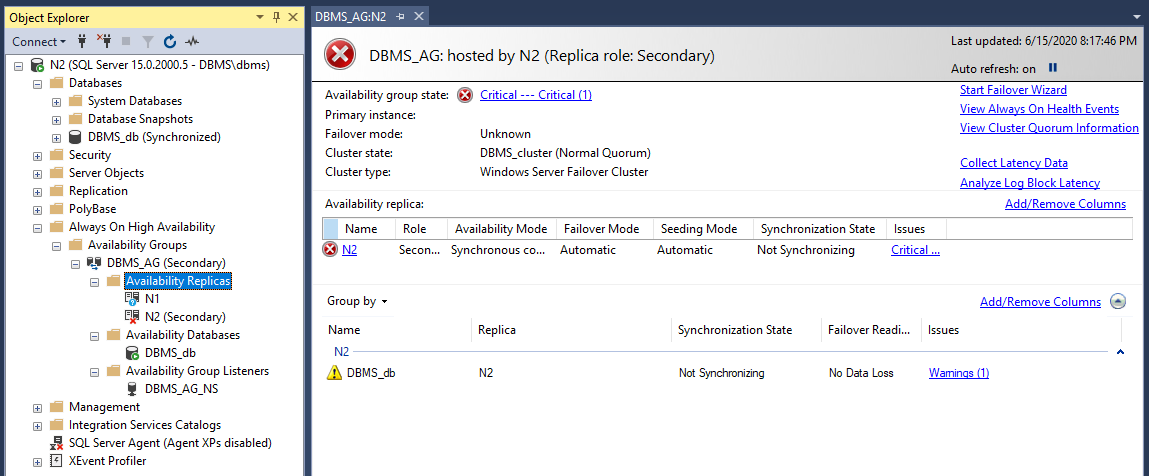
Ilustracija 31. Stanje AG pre otkaza

Nakon toga je pauzirana N1 virtuelna mašina:



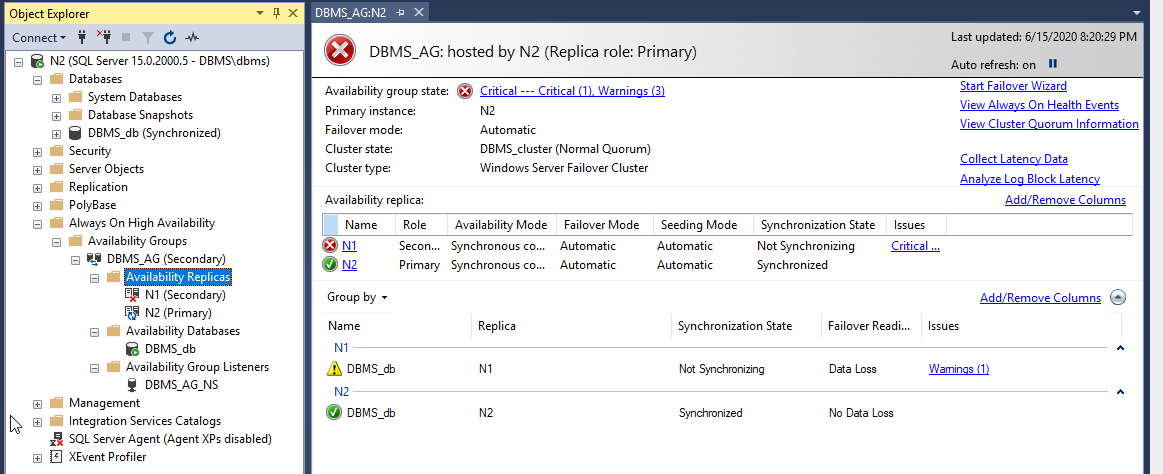
Ilustracija 32. Pauziranje N1 virtuelne mašine

Nakon otkaza, situacija na N2 i DBMS\_AG je par sekundi bila sledeća (Ilustracija 33):



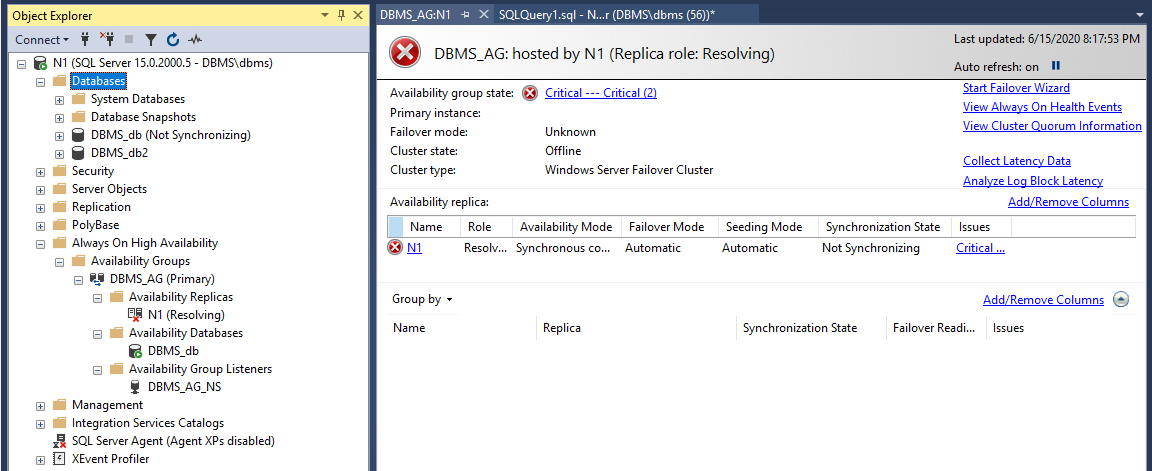
Ilustracija 33. Reagovanje N2 na otkaz

Nakon par sekundi, situacija na N2 se stabilizovala i N2 je postao primarni čvor (Ilustracija 34):



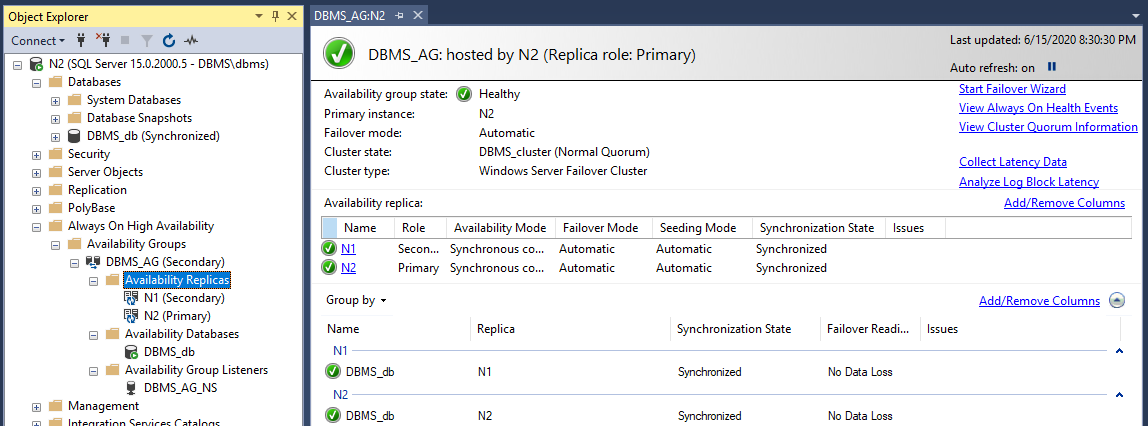
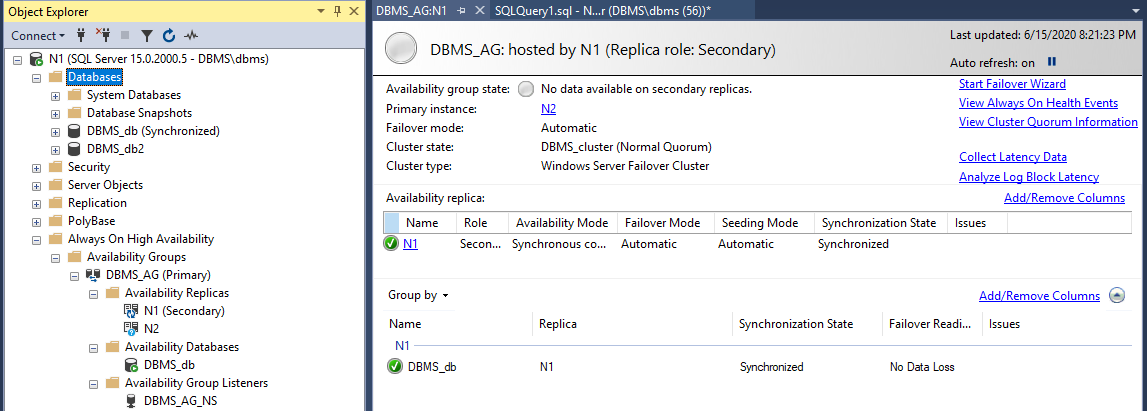
Ilustracija 34. N2 postao primarni čvor

Sada već možemo dopustiti oporavak N1 (odpauzirati). U početnom trenutku N2 ima identičan pogled kao prethodni, a N1 pokušava da nađe svoje mesto (Ilustracija 35):



Ilustracija 35. Oporavak N1

Nakon par sekundi će se N1 ponovo sinhronizovati sa N2 i klaster će se stabilizovati (Ilustracija 36):

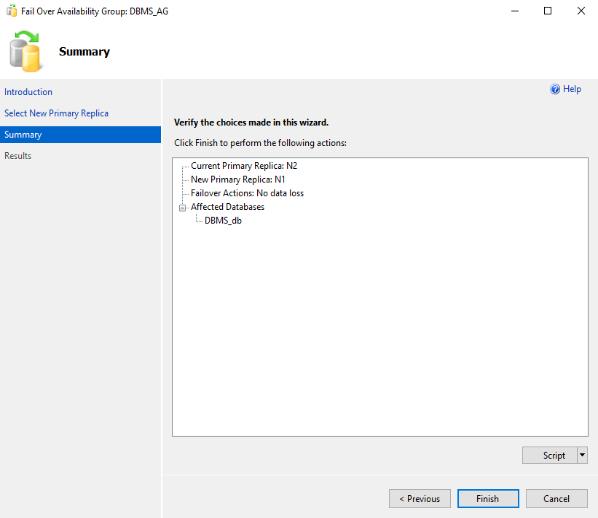
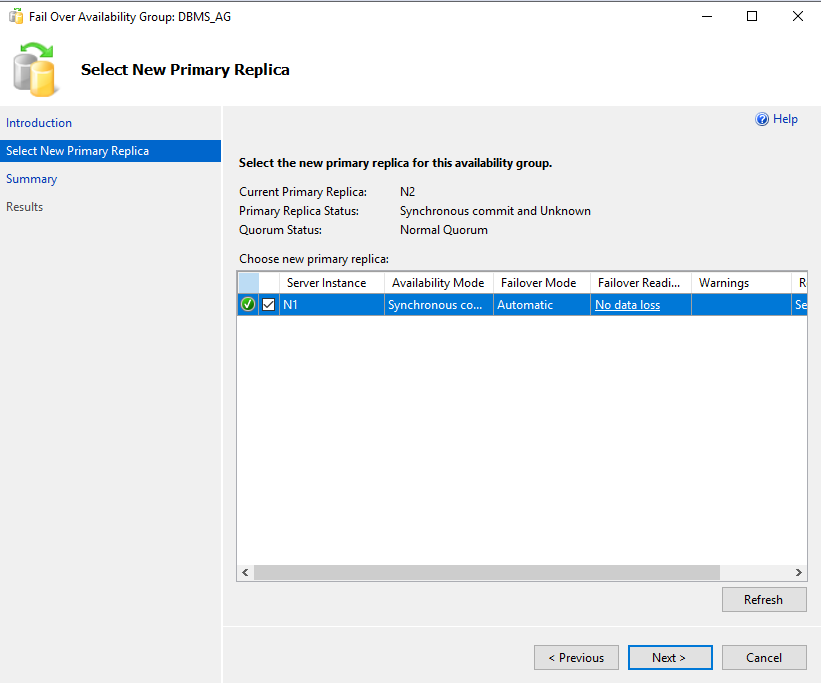
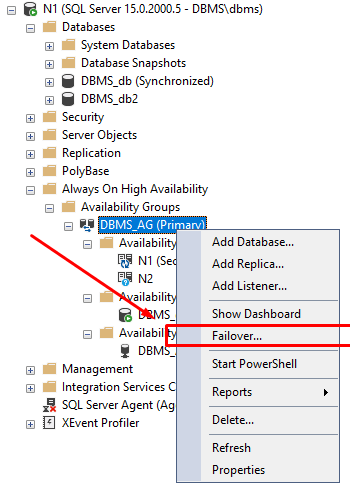


Ilustracija 36. Stanje N1 i N2 i AG nakon potpunog oporavka N1

Od ovog trenutka N2 je primarni a N1 sekundarni čvor u klasteru visoke dostupnosti.

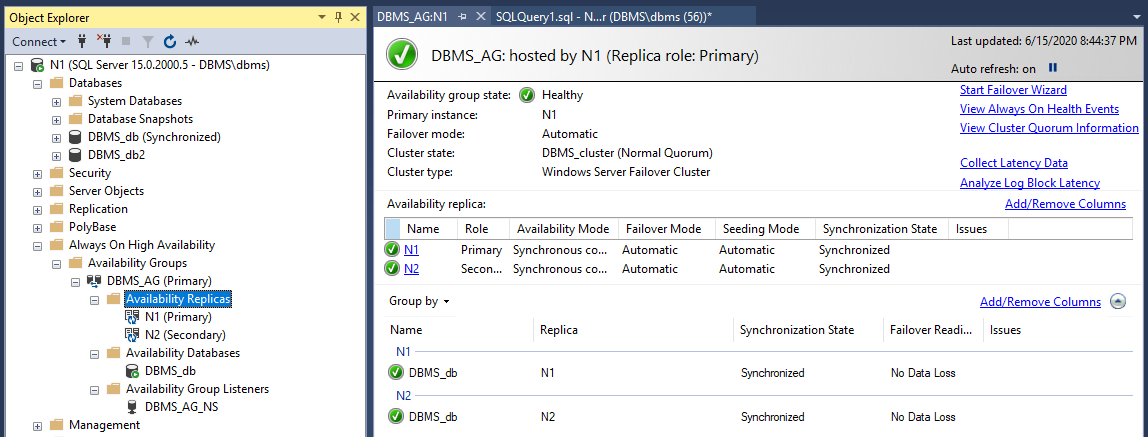
Sada možemo ilustrovati i ručni *failover*. Na bilo kom čvoru pozvati *Failover...* opciju, zatim biramo novu primarnu repliku, zatim nam se prikazuje spisak izmena i na kraju rezultati izvršenja (Ilustracija 37).

Krajnja situacija na N1 i AG (Ilustracija 38):



Ilustracija 37. Proces ručnog failover-a.

Redosled: Failover... -> Select New Primary Replica -> Summary -> Results



Ilustracija 38. Krajnja situacija N1 i AG

# Zaključak

U ovom seminarskom radu videli smo svrhu HA-DBMS-a koji predstavljaju rešenja povišene dostupnosti baze podataka bilo u slučaju problema sa hardverom, softverom ili samim podacima. Takođe pruža i skalabilnost pristupa čitanja.

Videli smo kako radi SQL Server *Always On Availability Groups* funkcionalnost, odrešene specifičnosti i ograničenja i kako se podešava kroz GUI SSMS-a.

SQL Server AG tehnologija pruža relativno uprošćen pristup klasterovanju baze podataka u svrsi povišene dostupnosti baze podataka. Koristi se u *Enterprise* rešenjima što je pokazatelj pouzdanosti tehnologije (ako se pravilno koristi, naravno).

# Literatura

1. Drake, S., Hu, W., McInnis, D., Sköld, M., Srivastava, A., & Thalmann, L., Tikkanen, M., Torbjørnsen, Ø., Wolski, A. (2005). Architecture of Highly Available Databases. In M. Malek, M. Reitenspieß, & J. Kaiser (Eds.), Service Availability (Vol. 3335, pp. 1–16). Springer.
2. Norman, M.G., Zurek, T., Thanisch, P.: Much Ado About Shared-Nothing. SIGMOD Record 25(3): 16-21 (1996)
3. Gray, J. and Reuter, A.: Transaction Processing Systems, Concepts and Techniques. Morgan Kaufmann Publishers, 1992
4. <https://docs.microsoft.com/en-us/sql/database-engine/availability-groups/windows/overview-of-always-on-availability-groups-sql-server?view=sql-server-ver15>
5. <https://docs.microsoft.com/en-us/sql/database-engine/availability-groups/windows/availability-modes-always-on-availability-groups?view=sql-server-ver15>
6. <https://sqlundercover.com/2017/12/18/creating-a-sql-server-test-lab-on-your-workstation-part-one-installing-the-domain-controller/>
7. <https://www.enisa.europa.eu/topics/threat-risk-management/risk-management/current-risk/bcm-resilience/bc-plan/it-service-continuity-plan>