|  |  |  |
| --- | --- | --- |
|  | UNIVERZITET U NOVOM SADU  **FAKULTET TEHNIČKIH NAUKA U NOVOM SADU** |  |

Aleksandar Momić  
E5-7/2019

**Sistem za upravljanje imovinom u elektrodistributivnoj mreži**

-Projektna dokumentacija-

-Master akademske studije-

-Novi Sad, 2020.-

Sadržaj

[1 Uvod 1](#_Toc40113051)

[2 Opis rešavanog problema 2](#_Toc40113052)

[3 Arhitektura 3](#_Toc40113053)

[3.1 Simulator 3](#_Toc40113054)

[3.2 Scada 3](#_Toc40113055)

[3.3 Calculation Engine (CE) 3](#_Toc40113056)

[3.4 NetworkModel Service (NMS) 3](#_Toc40113057)

[3.5 UI Adapter 4](#_Toc40113058)

[3.6 Publish-Subscribe mehanizam (PubSub) 4](#_Toc40113059)

[3.7 Transaction Service 4](#_Toc40113060)

[3.8 Korisnički interfejs (UI) 4](#_Toc40113061)

[4 Scada (simulacija elektroenergetskog sistema sa kritičnom misijom) 6](#_Toc40113062)

[4.1 Akvizicija podataka sa simulatora 6](#_Toc40113063)

[4.2 Obaveštenje o manipulaciji nad opremom 6](#_Toc40113064)

[5 Calculation Engine (napredni računarski sistemi sa kritičnom misijom u EE) 8](#_Toc40113065)

[5.1 Obrada akvizicije 8](#_Toc40113066)

[5.1.1 Čuvanje stanja pristiglih signala i povećavanje brojača manipulacija 8](#_Toc40113067)

[5.1.2 Upisivanje obrađenih podataka u istorijsku bazu 8](#_Toc40113068)

[5.1.3 Obaveštavanje UI adaptera o novom stanju 8](#_Toc40113069)

[5.2 Izveštaji 9](#_Toc40113070)

[5.2.1 Prikaz opreme koja je spremna za ugradnju 9](#_Toc40113071)

[6 Distribuirana transakcija (standardi i modeliranje elektroenergetskog softvera) 10](#_Toc40113072)

[7 NetworkModel Service (NMS) (sigurnost i bezbednost u Smart Grid sistemima) 12](#_Toc40113073)

[7.1 Nabavka određenog tipa opreme 12](#_Toc40113074)

[8 Azure Service Fabric (održavanje i kontrola kvaliteta elektroenergetskog softvera) 14](#_Toc40113075)

[8.1 Uvod 14](#_Toc40113076)

[8.2 Calculation Engine kao Service Fabric aplikacija 14](#_Toc40113077)

[8.2.1 DeltaTransaction mikroservis i primena modela na ostalim stateful mikroservisima 15](#_Toc40113078)

[8.2.1.1 Mikro-transakcija u okviru **CE** 15](#_Toc40113079)

[8.3 ServiceBus Queue struktura podataka 16](#_Toc40113080)

# Uvod

# Opis rešavanog problema

Napraviti aplikaciju za podršku poslovima održavanja u elektrodistributivnoj mreži, namenjenu unosu podataka o tehničkim karakteristikama i listama održavanja elektroenergetske opreme, u bazu podataka, kao i za pregled i obradu podataka iz baze i za pravljenje raznovrsnih izveštaja.

# Arhitektura

Arhitektura projekta je podeljena u tri sloja (Slika 3.1.1)



Slika 3.1.1 Slojevi aplikacije

* Prezentacioni sloj predstavlja korisnički interfejs kroz koji korisnik ima uvid u dešavanja u sistemu i vrši interakciju sa istim
* Biznis logiku obuhvataju servisi: **PubSub**, **CE**, **Scada**, **NMS**, **UI** **Adapter**. Logika je sakrivena od korisnika i može da bude distribuirana jer je komunikacija između svih servisa preko mreže.
* Sloj podataka je *Azure SQL Server* kome servisi pristupaju koristeći razvijenu biblioteku za komunikaciju sa istorijskom bazom podataka.

Slika 3.8.1 je slikovita reprezentacija arhitekture celokupnog sistema.

## Simulator

Simulator predstavlja *third-party* aplikaciju koja služi za simuliranje vrednosti elemenata u polju. Vrednosti mogu da se zadaju ručno kliktanjem po korisničkom interfejsu ili nekom aplikacijom koja komunicira sa njim i šalje komande. Za komunikaciju se koristi TCP, a format poruke mora da poštuje MODBUS protokol.

## Scada

Scada je servis koji vrši periodičnu akviziciju stanja vrednosti na simulatoru koristeći MODBUS protokol. Kada primi trenutno stanje, šalje se razlika trenutnog i prethodnog stanja na Calculation Engine servis na interpretaciju.

## Calculation Engine (CE)

Servis koji prima podatke sa Scada-e i interpretira njihovo značenje. Dužan je da proveri da li se neki od uređaja nalazi u alarmnom stanju i da zapisuje podatke u Azure SQL bazu podataka (na slici HIST). Za to vreme se pristigli podaci proslede na UI Adapter. Ako se utvrdi da je neki uređaj u alarmnom stanju, dodatne informacije se šalju na UI Adapter da bi se korisnički interfejs informisao o tom stanju. Na zahtev UI Adapter-a generiše izveštaje i šalje ih nazad.

## NetworkModel Service (NMS)

Servis koji čuva konektivnost mreže. Podatke o konektivnosti može da primi od **Importer** aplikacije i sa korisničkog interfejsa. Pristigle podatke validira i primenjuje ako su validni, u suprotnom informiše pošiljaoca o tome šta je bila greška u validaciji. Nakon primenjenog modela, ostali servisi mogu po potrebi da šalju zahteve i traže informacije o statičkom modelu.

## UI Adapter

Specifičnost ovog servisa je u tome da služi kao most između korisničkog interfejsa i ostatka sistema. Korisnički interfejs šalje jedinstvene zahteve i dužnost ovog servisa je da koordiniše zahteve na jedan ili više servisa da bi u potpunosti odgovorio na zahtev korisničkog interfejsa. Tako korisnički interfejs i ostatak sistema nemaju deljene modele podataka.

## Publish-Subscribe mehanizam (PubSub)

Da bi moglo da se podrži više od jednog korisničkog interfejsa i da to bude nezavisno od celog sistema koristi se publish-subscribe mehanizam koji će pristigle podatke da demultipleksira na svaki od korisničkih interfejsa tako da svi budu u sinhronizaciji.

## Transaction Service

Funkcionalnost ovog servisa je koordinacija distribuirane transakcije kroz sistem. Kada podaci stignu na **NMS** i uspešno prođu validacije proslede se ostalim servisima. Nakon toga se servisi od interesa prijavljuju transakcionom servisu i kada se svi prijave **NMS** obavesti transakcioni servis da distribuirana transakcija može da se pokrene.

## Korisnički interfejs (UI)

Korisnik za interakciju sa sistemom ima na raspolaganju korisnički interfejs. Funkcionalnosti mu omogućavaju manipulaciju uređajima koji se nalaze u sistemu, njihovo prebacivanje u funkciju i van funkcije, prikaz mnoštva izveštaja, popravku.



Slika 3.8.1 Arhitektura celog zadatka

# Scada (simulacija elektroenergetskog sistema sa kritičnom misijom)

## Akvizicija podataka sa simulatora

Pošto se stanje prekidačke opreme u polju (na simulatoru) može konstantno menjati, mora se periodično proveravati njihovo stanje. Scada periodično, koristeći MODBUS protokol, proziva simulator i „pita“ kakvo je trenutno stanje u polju. Od simulatora dobije informaciju o novom stanju prekidačke opreme. Po prijemu novog stanja primljena je informacija o stanju svakog prekidača u polju. Međutim, tu se nalaze i prekidači kojima se nije promenilo stanje, a čak može da se desi i da se nijednom nije promenilo stanje. Slika 4.1.1 pokazuje kako se utvrđuje gde je došlo do promena. Ovim metodom se smanjuje opterećenje ostalih servisa kojima se prosleđuje stanje.



Slika 4.1.1 Utvrđivanje promena u polju

* Trenutno stanje –trenutno upamćeno stanje na Scada servisu
* Novo stanje – stanje primljeno kroz akviziciju sa simulatora
* Razlika – indeksi promena i njihove nove vrednosti

Po prijemu novog stanja trenutno i novo stanje se proslede u funkciju koja utvrđuje razliku između ta dva stanja. Utvrđivanje se vrši tako što se izvrši logička operacija **ekskluzivno ili** između njih da bi se dobili indeksi izmenjenih vrednosti. Izlaz funkcije je novo stanje sa označenim promenjenim vrednostima. Nakon toga se novo stanje sačuva kao trenutno stanje i promenjene vrednosti pošalju na Calculation Engine.

Trenutno stanje se čuva u ugrađenoj klasi *BitArray* koja omogućava čuvanje niza bitova od N članova. Time se izbegava *overhead* koji bi se javio čuvanjem stanja svakog prekidača u članovima *Boolean* niza.

## Obaveštenje o manipulaciji nad opremom

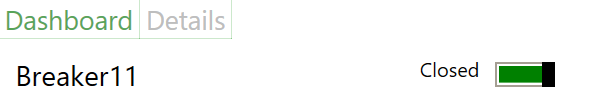
Manipulacija opremom se može vršiti direktno na simulatoru, a može i na korisničkom interfejsu. Na korisničkom interfejsu se na *Dashboard* tabu nalazi dugme za manipulaciju prekidačkom opremom (Slika 4.2.1 - Slika 4.2.3) i pokazuje trenutno stanje prekidačke opreme.



Slika 4.2.1 Prekidač u otvorenom stanju



Slika 4.2.2 Animacija za čekanje odgovora komandovanja



Slika 4.2.3 Prekidač u zatvorenom stanju

Obaveštenje o manipulaciji se svodi na dve nezavisne informacije:

1. Uspešnost slanja komande na simulator (u polje)
2. Obaveštenje o uspešnoj promeni vrednosti na simulatoru (u polju)

Klikom na dugme komanduje se odgovarajućom opremom. Šalje se komanda suprotna od trenutnog stanja. Ako je prekidač otvoren, poslaće se komanda za njegovo zatvaranje, u suprotnom se šalje da se otvori. Ta komanda se šalje na **CE** koji komandu prosleđuje **Scada** servisu. Povratna vrednost je uspešnost slanja komande u polje. Ako je komanda uspešno poslata u polje (na simulator), dugme ulazi u fazu čekanja gde se pojavi *loading* animacija (Slika 4.2.2) i čeka se informacija o tome da li se zapravo promenila vrednost na simulatoru (u polju). Informacija stiže u prvom narednom ciklusu akvizicije stanja i dugme se ažurira da pokazuje novo stanje tog prekidača. Ukoliko ne uspe da se manipuliše opremom, na korisničkom interfejsu se ispisuje odgovarajuća poruka.

# Calculation Engine (napredni računarski sistemi sa kritičnom misijom u EE)

## Obrada akvizicije

Po prijemu podataka obrađenih u poglavlju 4.1 potrebno je da se ti podaci interpretiraju i da im se da neka semantika. Pošto je **Scada** dizajnirana da nema nikakvu *biznis logiku*, odgovornost za tu logiku je prebačena na **CE**[[1]](#footnote-1).

Obrada se sastoji iz četiri segmenta:

1. Čuvanje stanja pristiglih signala i povećavanje brojača manipulacija
2. Provera da li se oprema povezana na taj signal nalazi u alarmu
3. Upisivanje obrađenih podataka u istorijsku bazu podataka
4. Obaveštavanje UI adaptera o novom stanju

### Čuvanje stanja pristiglih signala i povećavanje brojača manipulacija

Od **Scada** servisa stiže informacija na kom GID[[2]](#footnote-2)-u signala je koje novo stanje. Za dobijeni *GID* se iz **CE** modela uzima konkretan objekat diskretnog signala u kom se nalaze podaci potrebni za obradu. U slučaju da signal ne postoji u modelu, baca se izuzetak i tako se zna da **Scada** i **CE** nisu u konzistentnom stanju. Nakon dobijanja konkretnog signala njegovo stanje i uvećan brojač se čuvaju u **CE** radnom modelu. Treba primetiti da je ovo samo u radnom modelu i da još nije završilo u istorijskoj bazi podataka. Brojač manipulacija se uvećava kada signal prelazi iz otvorenog u zatvoreno stanje. Nakon ažuriranja podataka proverava se da li je odgovarajuća oprema koja je povezana na taj signal u alarmu.

### Upisivanje obrađenih podataka u istorijsku bazu

Obrađeni podaci se moraju čuvati u istorijskoj bazi iz razloga što se istorija koristi za:

* Inicijalizaciju **CE** i **NMS** servisa
* Kreiranje mnoštva izveštaja

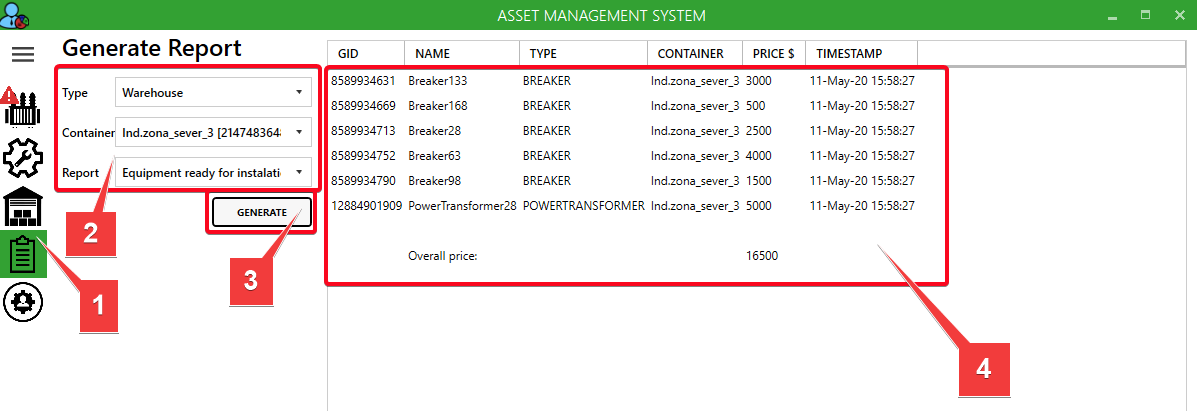
Zbog performansi podaci ne završavaju odmah u bazi podataka, nego se upisuju u grupama (*batch*). Količina podataka u jednoj grupi je konfigurabilna. Nakon obrade podaci se čuvaju u listama koje kada dostignu tu količinu, sve se upiše u istorijsku bazu i očiste se liste gde se čuva nova grupa podataka.

### Obaveštavanje UI adaptera o novom stanju

Odmah nakon obrade podataka (pre provere alarma) promene se „pakuju“ u format podataka poznat UI adapteru. Slanje je asinhrono. Tako pristigli podaci na UI adapteru se prosleđuju na ServiceBus topic nazvan ***changes*** odakle PubSub mehanizam prima poruke i dostavlja ih svakom otvorenom korisničkom interfejsu.

## Izveštaji

Zahtev za procesiranje svakog izveštaja se šalje sa korisničkog interfejsa (Slika 5.2.1). Postoji određeni tab (1) na njemu gde se bira željeni izveštaj (2). Klikom na dugme ***Generate***(3) šalje se zahtev na UI adapter koji prosleđuje to odgovarajućem servisu. Kao povratna vrednost dobija se objekat klase **ReportModel** i on je univerzalan za svaki izveštaj (4).



Slika 5.2.1 Tab sa izveštajima

### Prikaz opreme koja je spremna za ugradnju

Opremu za ugradnju predstavlja oprema koja je kupljena i oprema koja je popravljena. Kada **CE** primi zahtev za ovaj izveštaj, svi potrebni podaci se nalaze u istorijskoj bazi. Poziva se odgovarajuća procedura koja uzima kupljenu i popravljenu opremu. Povratni podaci se „pakuju“ u **ReportModel**, po potrebi dopunjuju dodatnim podacima sa **NMS-a** i šalju nazad UI adapteru. Dobijeni model se prikazuje na ***DataGrid-u*** uz opcije **export-a** istih u *PDF* i *CSV*.

# Distribuirana transakcija (standardi i modeliranje elektroenergetskog softvera)

Distribuirana transakcija obezbeđuje konzistentnost podataka između servisa. U ovom projektu se **NMS** ponaša kao pokretač transakcije i učestvuje u istoj zajedno sa **CE** i **Scada** servisom.

Za konzistentnost podataka između servisa zadužen je **Transaction Coordinator** koji koordiniše distribuiranu transakciju. Slika 5.2.1 predstavlja ceo proces distribuirane transakcije i tok podataka kroz istu. Cela transakcija se vrši asinhrono.



Slika 5.2.1 Tok podataka u distribuiranoj transakciji

Distribuirana transakcija se izvršava u pet koraka:

1. U prvom koraku **NMS** primi novu promenu na modelu i izvrši validaciju. Nakon validacije, ako je model validan prelazi se na drugi korak, u suprotnom se o nevalidnosti modela obavesti klijent koji je poslao promenu. U ovom projektu promenu modela šalju **Importer** i korisnički interfejs.
2. Nakon toga se validan model pošalje ostalim, zainteresovanim, servisima da ga sačuvaju da bi mogli da učestvuju u transakciji.
3. U trećem koraku se **NMS** registruje koordinatoru i tada koordinator čuva kanal za komunikaciju sa **NMS-om**.
4. U narednom koraku po uspešnom čuvanju primljenih podataka se i **Scada** i **CE** registruju na koordinator i „javljaju“ **NMS-u** da su spremni.
5. Kada **NMS** primi od svih servisa da su spremni, šalje koordinatoru informaciju da može da se izvrši transakcija. Ukoliko se neko ne javi ili javi grešku, transakcija se obustavlja.

Transakcioni protokol koji se koristi je **dvofazni *commit***:

* Prva faza je faza pripreme. U ovoj fazi **NMS** pravi *shallow-copy* kopiju svog radnog modela. **Scada** i **CE** čitaju podatke koje su sačuvali u koraku (2) i dopunjuju svoje modele potrebnim podacima tako što kontaktiraju **NMS** za nedostajuće podatke. Ti podaci se dobijaju iz transakcionog modela.
* Druga faza je primena ili odbacivanje promena. Ukoliko u bilo kom delu prve faze nešto rezultuje neuspehom, **NMS** informiše koordinator da se transakcija obustavlja i koordinator na svim učesnicima u transakciji poziva odbacivanje promena. U suprotnom se podaci primenjuju, gde **NMS** menja svoj radni model i on postaje ono što je bio transakcioni model, **CE** i **Scada** upisuju u svoje radne modele.

# NetworkModel Service (NMS) (sigurnost i bezbednost u Smart Grid sistemima)

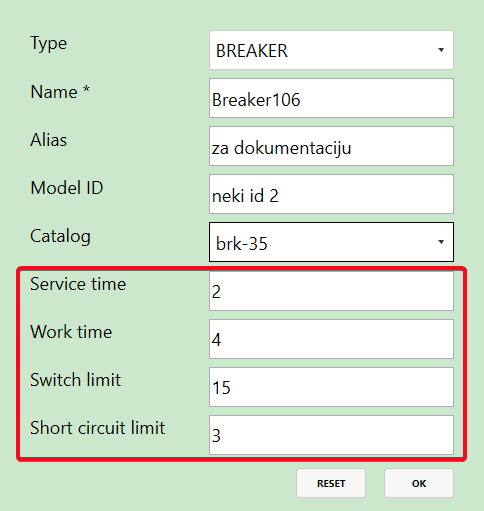
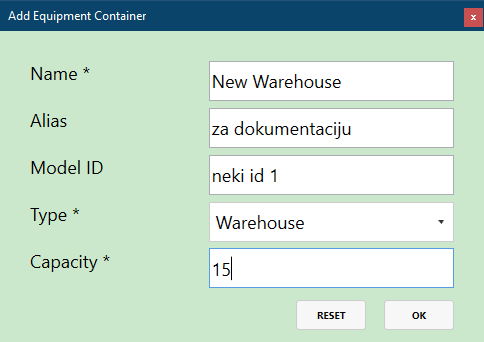
## Nabavka određenog tipa opreme

Primena modela preko **Importer** aplikacije predstavlja jedan način simulacije nabavke opreme. Sve što se primi iz **Importer**-a je kao jedan nabavka i primenjuje se u model svakog servisa kroz distribuiranu transakciju.

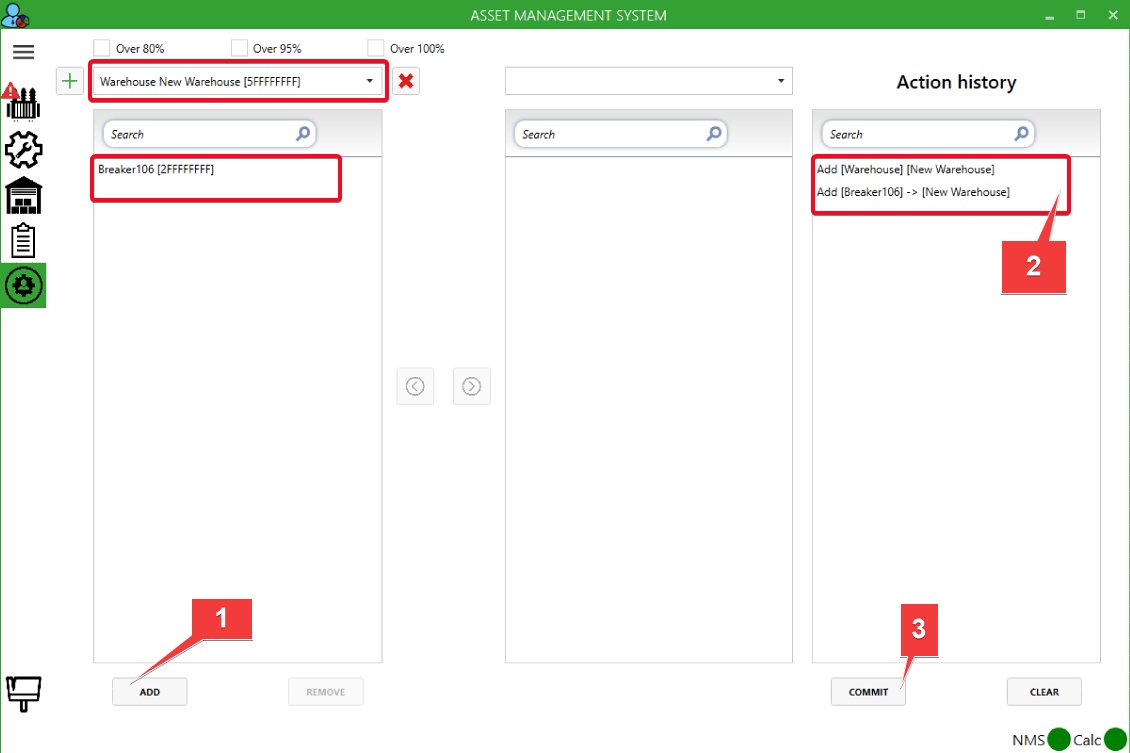
Da ne bi morao ručno da se piše *CIM-XML* za svaku nabavku, implementirana je i simulacija nabavke opreme na korisničkom interfejsu. U delu **Asset Manager** (Slika 7.1.2) se nalazi *user-friendly* način za obavljanje nabavke. Sva kupljena oprema ide prvobitno u skladište (*warehouse*). Ukoliko nema skladišta u sistemu, klikom na + ikonicu pored padajućeg menija gde se nalaze kontejneri za skladištenje opreme, može se „kupiti“ trafo stanica, skladište i kontejner za održavanje. U ovom slučaju se izabere skladište i time je „kupljeno“ skladište (Slika 7.1.1).

Nakon kupovine skladišta moguća je kupovina opreme jer postoji mesto gde se ta oprema može smestiti. Klikom na **Add** dugme (1) u donjem levom uglu „kupuje“ se oprema (prekidač, transformator). Izborom određene opreme pojavljuju se polja za popunjavanje specifikacija i izbor kataloga u kom se nalazi ta oprema (Slika 7.1.1, sa desne strane). Time se učitavaju dodatne informacije o opremi i te informacije ne mogu da se modifikuju (Slika 7.1.1, sa desne strane, uokviren deo).

Svaka akcija se čuva u listi akcija koje se klikom na **Commit** (3) konvertuju u **Delta** objekat poznat **NMS**-u i pokreće se validacija i distribuirana transakcija. Jedna akcija je, na primer, dodavanje prekidača. Skup akcija predstavljaju jedan objekat **Delte** i primenjuje se odjednom, a ne svaka ponaosob.



Slika 7.1.1 Dodavanje/kupovina novog skladišta i prekidača



Slika 7.1.2 Izgled Asset Manager-a nakon dodavanja skladišta i prekidača

# Azure Service Fabric (održavanje i kontrola kvaliteta elektroenergetskog softvera)

## Uvod

Azure Service Fabric je platforma Majkrosofta, koja omogućava lako pakovanje podataka, razvijanje aplikacija i omogućava skalabilne i pouzdane sisteme. Takvi sistemi su izgrađeni od manjih mikroservisa i kontejnera za skladištenje podataka.

Da bi se ostvario ovakav sistem, potrebno je razlikovati mikroservise i monolitičke aplikacije. Monolitičke aplikacije se grade po nivoima, gde na svakom nivou može da se koristi različita tehnologija. Interfejsi između ovih nivoa su usko povezani. Postoje mnoge prednosti kod izgradnje monolitičkih aplikacija u odnosu na skup mikroservisa, kao što su: jednostavniji dizajn, brža komunikacija između komponenti i lakše testiranje. Glavna mana monolitičkih aplikacija je otežana skalabilnost i uska povezanost komponenti, gde greška na jednom nivou može eskalirati na nekom drugom nivou.

Sa druge strane, mikroservisima se prevazilaze ovi problemi i korisnik može više da se posveti biznis problemima i da efikasnije ispuni zahteve klijenata. Mikroservisi mogu biti stateless i stateful. Kao što im i ime govori, stateless servisi ne čuvaju stanja promenljivih, a stateful servisi čuvaju. Svaki mikroservis predstavlja jedinstvenu celinu koja ima izdvojenu funkcionalnost, može da se skalira, testira i razvija nezavisno od ostalih mikroservisa. Različiti timovi mogu da razvijaju različite mikroservise na različitim tehnologijama. Brzina i kašnjenje podataka u komunikaciji mikroservisa je jedan od najvećih problema kod aplikacija koje su ovako izgrađene.

## Calculation Engine kao Service Fabric aplikacija

Projekat je prvobitno razvijan kao skup monolitičkih aplikacija koje međusobno komuniciraju preko mreže. To je prihvatljivo rešenje dok ne dođe do nekog problema koji se može pojaviti na mestu na kom nije nastao. Prelaskom na mikroservisnu arhitekturu, **CE** je uspešno razdvojen na devet mikroservisa gde je očigledno koji mikroservis ima koju odgovornost. Razdvojeno je tako što je pregledano koji servis na monolitičkom **CE** piše koje podatke i odlučeno da servisi koji pišu podatke budu *stateful* mikroservisi, a oni koji samo čitaju (iz istorijske baze ili sa drugih mikroservisa) budu *stateless*. Slika 8.2.1 pokazuje kako izgleda **CE** razdvojen na mikroservise.



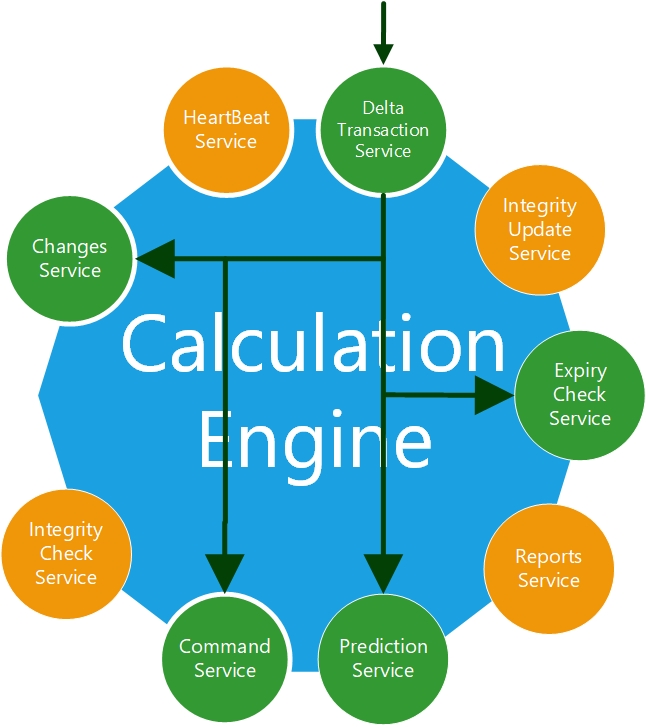
Slika 8.2.1 Calculation Engine mikroservisi

### DeltaTransaction mikroservis i primena modela na ostalim stateful mikroservisima

Ovaj mikroservis je zadužen za primenu promene modela u svim stateful mikroservisima u okviru **CE**. Kada se započne primena modela **NMS** šalje promenu samo na **DeltaTransaction** mikroservis. Ova odluka je doneta da **NMS** ne bi morao da zna koliko mikroservisa je zainteresovano za model na **CE** i tako je povećan nivo apstrakcije. Ako bi se odlučilo da je još jedan mikroservis zainteresovan, nema potrebe menjati ceo tok transakcije i da se još jedan mikroservis uključuje u transakciju, nego se samo ažurira „*mikro-transakcija“* u okviru **CE**.

#### Mikro-transakcija u okviru **CE**

Kada se primi promena modela sa **NMS**-a započinje se mikro-transakcija u okviru **CE**. Slika 8.2.2 pokazuje tok podataka u toku mikro-transakcije. **DeltaTransaction** mikroservis prosledi promenu ostalim servisima i javi se koordinatoru i **NMS**-u da je spreman za transakciju (deo objašnjen u poglavlju 6). U fazi pripreme, u okviru **CE**, **DeltaTransaction** se pripremi i pošalje svim stateful mikroservisima da se pripreme. Ukoliko se svi uspešno pripreme, jave **DeltaTransaction** mikroservisu da su spremni i tek tada se javlja koordinatoru da se **CE** u celini pripremio i prelazi se na fazu primene modela. U suprotnom, ako se bilo koji mikroservis ne pripremi uspešno, ulazi se u fazu odbacivanja promena.



Slika 8.2.2 Tok podataka u toku mikro-transakcije

## ServiceBus Queue struktura podataka

*ServiceBus Queue* (red) je struktura podataka, a ujedno i mehanizam koji Majkrosoft nudi kao gotovo rešenje, koja omogućava razdvajanje zavisnosti između **Scada** servisa i **Changes** mikroservisa. Slika 8.3.1 pokazuje kako promene stižu od **Scada** servisa do **Changes** mikroservisa koristeći *ServiceBus Queue*.



Slika 8.3.1 Tok podataka promena sa Scada servisa preko ServiceBus Queue-a do **CE**

Red kao struktura podataka je dobar izbor jer se iz reda poruke čitaju istim redosledom kao što su upisane. Da se koristi neka struktura koja nije na ovom principu, moglo bi da dođe do nekonzistentnosti stanja u polju (na simulatoru) i u celom sistemu. Ako bi se, recimo, prekidač uključio, pa isključio i tako pristigle poruke pročitale u pogrešnom redosledu, u polju (na simulatoru) bi prekidač bio isključen, a sistem bi imao informaciju o tome da je uključen, što bi dovelo do, na primer, pogrešnih alarma.

Za to se brine Majkrosoftov ugrađen mehanizam koji pored svega nudi i dodatne funkcionalnosti kao što su životni vek poruke, povratna vrednost ukoliko se ne pročita poruka, detekciju duplikata, itd.

1. Calculation Engine [↑](#footnote-ref-1)
2. Globalni identifikator [↑](#footnote-ref-2)