Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.41
DOI: https://doi.org/10.24867/17BE11Dupljanin

RAZVOJ APLIKACIJE ZA UPRAVLJANJE ELEKRIČNOM ENERGIJOM U OKVIRU CLOUD OKRUŽENJA

DEVELOPMENT OF AN APPLICATION FOR ELECTRICITY MANAGEMENT WITHIN THE CLOUD ENVIRONMENT

Snežana Dupljanin, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *Aplikacija EMS predstavlja sistem upravljanja energijom, koja vrši monitoring potrošnje elekrtične energije u realnom vremenu i teži da zadovolji jednakost između potrošene i proizvedene električne energije, koristeći se optimizacionim gentskim algoritmom. Sistem je implementiran u Cloud okruženju korišćenjem Azure Service Fabric platforme. U radu su izmjerena vremena izvršavanja genetskog algoritma za različit broj klijenata kako za mikroservisnu arhitekturu u Cloud okruženju, tako i za monolitnu aplikaciju.*

Ključne reči: *Azure, Cloud, CE, stateful i stateless mikroservisi, Azure Storage.*

Abstract – *EMS (Energy Management System) monitors electricity consumption in real time and strives to satisfy the equality between consumed and produced electricity, using an optimization genetic agorithm. The system environment is Cloud using the Azure Service Fabric platform. The paper measures the execution times of the genetic algorithm for a different numbers of clients, for the architecture of microservices in the Cloud environment and the monolithic application.*

Keywords: *Azure, Cloud, CE, stateful and stateless microservices, Azure Storage.*

1. UVOD

1.1 Azure Service Fabric

Azure Service Fabric [1] je platforma kao usluga (PaaS), tj. distribuirana servisna platforma open-source koja olakšava pakovanje, primjenu i upravljanje skalabilnim i pouzdanim mikroservisima i kontejnerima. Ona pruža mogućnost upravljanja životnim ciklusom aplikacija zasnovanih na mikroservisima.
Mikroservisi se hostuju unutar kontejnera koji su raspoređeni u klasteru Service Fabric-e. Uvođenjem mikroservisa moguće je skalirati svaku komponentu aplikacije zasebno, što pojednostavljuje proces uvođenja promjena, za razliku od tradicionalne monolitne arhitekture. Azure Servise Fabric podržava stateful i stateless mikroservise. Mikroservisi komuniciraju međusobno uz pomoć API-ja.

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Srđan Vukmirović, vanr. prof.

1.2 Calculation Engine

Calculation Engine je u okviru monolitičke arhitekture predstavljao jedan kompleksan servis koji je obavljao više funkcionalnosti, pa ga je zbog toga bilo neophodno podijeliti na više mikroservisa. Ta podjela usluga je raspoređena na četiri mikroservisa, a to su: 1. Servis koji se bavi proračunom vezanim za CO2 (emisija/redukcija). 2. Servis koji se bavi proračunom profita i troška. 3. Servis za komunikaciju sa bazom. 4. Servis u kom se primjenjuje genetski algoritam.

Prva tri servisa su stateless jer je njihov zadatak da manipulišu podacima koji se skladište u bazi ili da vrše proračune i dobijene rezultate prosleđuju dalje. Četvrti servis je servis koji izvršava algoritam i on je stateful jer je neophodno da sačuva najbolju jedinku iz prethodnog računanja koja predstavlja prvu jedinku u prvoj populaciji novog računanja.

1.3 Azure Storage

Azure Storage [2] je platforma za skladištenje podataka na cloud platformi. Svi podaci su šifrovani od strane Azure Storage-a koji pruža kontrolu pristupa. Dizajniran je tako da bude skalabilan i zadovolji potrebe za skladištenjem podataka i performansama današnjih aplikacija. On takođe vodi računa o održavanju hardvera, update-a i kritičnih problema. Azure storage platforma nudi više različitih data servisa, od kojih su Azure Tables i Azure Blobs korišćeni u projektu.

2. OPIS KORIŠĆENIH TEHNOLOGIJA I ALATA

Azure Service Fabric je platforma distribuiranih sistema koja olakšava implementaciju i upravljanje skalabilnim i pouzdanim mikroservisima.

Service Fabric Explorer je open-source alat koji omogućava pregled i upravljanje Azure Service Fabric klasterima.

Microsoft Visual Studio je integrisano razvojno okruženje koje omogućava razvoj aplikacija.

Microsoft Azure Storage Explorer je samostalna aplikacija koja olakšava rad sa Azure Storage podacima .

C# je moderan objektmo orjentisan programski jezik višeg nivoa, koji je razvila kompanija Microsoft.

.NET Framework je platforma za razvoj softvera koju je Microsoft razvio za izradu i pokretanje aplikacija.

WPF (*Windows Presentation Foundation*) je tehnologija za pravljenje klijentskih aplikacija na Windows platformi, koja nudi razne opcije za podešavanje izgleda aplikacije.

3. OPIS PROBLEMA

Zadatak ove aplikacije jeste da vrši monitoring potrošnje električne energije u realnom vremenu i teži da zadovolji jednakost između potrošene i proizvedene električne energije, koristeći se optimizacionim gentskim algoritmom. Ovim algoritmom se teži ostvariti minimum potrošnje goriva u konvencionalnim generatorima, tj. povećati profit uzimajući u obzir cijenu goriva kao i proizvedenu električnu energiju iz generatora, koji koriste obnovljive izvore. Pored optimizacije vrše se proračuni za redukciju i emisiju CO2 izraženu u tonama, kao i profit i trošak izražen u dolarima. Mikroservisna arhitektura je sačinjena od skupa nezavisnih servisa od kojih svaki izvršava zasebno svoj zadatak. Glavni cilj jeste da se podijeli aplikacija na što više manjih nezavisnih mikroservisa i da obezbijedi njihovu međusobnu komunikaciju.

4. TEORIJSKE OSNOVE

4.1 Mikroservisna arhitektura

Monolitske aplikacije su dobro rješenje samo za manje aplikacije koje nisu distribuiranje i od kojih se ne zahtjevaju visoke performanse opsluživanja klijenata, ali sa stanovišta održavanja aplikacije i distribuiranosti, kod ovih aplikacija će se uvijek javiti problem kako sistem održati stabilnim.

Mikroservisna arhitektura sa druge strane omogućava tu pouzdanost, skalabilnost i proširivost na brži i jednostavniji način. Čitava poenta je razbijanje aplikacije na mikroservise kao zasebne cjeline u sistemu koje odrađuju neki svoj specifičan dio posla. Mikroservisna arhitektura sve više dobija na popularnosti, sa sve više kompanija koje dijele pozitivna iskustva u vezi sa njenom primjenom. Servisi unutar aplikacije su izdjeljeni na stateless i stateful.

4.1.1 Stateless mikroservisi

Stateless mikroservisi [4] obezbjeđuju zahtjeve samo na osnovu informacija prenijetih uz svaki zahtjev i ne oslanjaju se na informacije iz ranijih zahjeva. To znači da servisi ne moraju da drže informacije o stanju između zatjeva. Stateless servisi ne čuvaju lokalno promjenjiva stanja. Uglavnom se potrebni podaci za rad čuvaju eksterno u bazi podataka.

Instance stateless servisa se raspoređuju po čvorovima i u slučaju otkaza bilo koje instance ili čvora na kom se instanca nalazi, se vrši ponovno startovanje na ispravnom čvoru. Stateless servisi su jednostavniji od stateful servisa, samim tim što ne moraju da paze na kopiranje podataka prilikom spuštanja servisa. Samim tim što ne čuvaju podatke, servisi ovog tipa komuniciraju sa steteful servisima radi dobavljanja podataka. Na podacima koje su primili, obave određene proračune i vrate ih servisima koji ih mogu čuvati ili ih prosto proslijede da bi se prikazali korisniku.

4.1.2 Stateful mikroservisi

Stateful servisi [4] održavaju promjenljivo stanje. Programski kod se izvršava samo na jednom čvoru koji se naziva primarni, dok su ostali čvorovi određenog servisa pasivni. U toku rada, primarni čvor konstantno replicira podatke na sekundarne čvorove i na taj način održava konzistentno stanje sistema. Odnosno, u slučaju otkaza primarnog čvora, jedan od sekundarnih preuzima posao i nastavlja njegovo izvršavanje. Takođe, vrši se oporavak čvora koji je otkazao, pri čemu se on formira kao sekundarni čvor.

5. IMPLEMENTACIJA RJEŠENJA

5.1 Arhitektura rješenja

Tokom izrade projekta korišćeno je razvojno okruženje (Local cluster) koje je isto kao produkciono okruženje, pa tako implementirane aplikacije na lokalnom klasteru mogu lako biti primjenjene na nekom drugom okruženju. Aplikacija u Cloud okruženju je na local host-u i sastoji se od jedne Service Fabric-e, koja ima 14 mikroservisa. Slika 1.

Calculation Engine Microservice je servis zadužen za izvršavanje genetskog algoritam i on je stateful jer je neophodno da sačuva najbolju jedinku iz prethodnog računanja koja predstavlja prvu jedinku u prvoj populaciji novog računanja.

Emission and Reduction CO2 Microservice je zadužen za računanje emisije CO2 pri čemu se ukupna snaga dobijena iz generatora pomnoži sa emisionim faktorom koji je različit za svako gorivo. Redukcija CO2 se računa na način da se pretpostavi koliko bi bilo emitovano CO2 da se snaga dobijena iz generatora na obnovljivi izvor energije, proizvodila iz konvencionlanih generatora i da se pri tome uzima najveći emisioni faktor.

Cost and Profit CE Microservice je zadužen za računanje profita tako što snagu iz generatora na obnovljive izvore energije pomnožimo sa najjeftinijim gorivom. Cost se računa tako što se uzme suma svih snaga generatora pomnožena sa cijenom za to gorivo koje generator troši.

Repository CE Microservice je servis zadužen za komunikaciju sa bazom, tj. za čuvanje svih izračunatih vrjednosti na CE to su: profit, cost, emisija i redukcija. Izračunati podaci se čuvaju u Azure Table Storage-u.

Calculation Engine PunSub Microservice Komunikacija između Calculation Engine-a i UI-a je ostvarena preko Publish – Subscribe je paterna. Nakon što genetski algoritam završi optimizaciju, trenutna proizvodnja generatora, profit i trošak izražen u dolarima, emisija i redukcija CO2 se prosleđuju na UI. U ovom slučaju CE predstavlja Publisher-a, dok UI predstavlja Subscriber-a.

SCADA Collecting Microservice je zadužen za prikupljanje podataka iz polja. Simulator korišćen za simuliranje podataka je EasyModbus, koji koristi Modbus protokol za komunikaciju. Scada uspostavlja konekciju sa simulatorom koristeći ModbusClient klasu. Nakon uspostavljene komunikacije, počinje prikupljanje podataka sa simulatora koje se vrši na svake 4 sekunde.

SCADA Processing Microservice Nakon što se podaci pročitaju sa simulatora, oni se šalju na SCADA Processing mikroservis. On je zadužen za procesiranje podataka, tj. vrši se konverzija promjenljivih iz sirovog formata u format inženjerskih jedinica (EGU values).

SCADA Commanding Microservice ima zadatak da prima komande i omogući njihovo dalje prosleđivanje u polje. Komandovanje digitalnim i analognim vrijednostima moguće je manuelno ili automatski.

Putem grafičkog korisničkog interfejsa, korisnik može da uključi/isključi generator i da zadaje vrijednosti generatora. Automatsko komandovanje je od strane genetskog aloritma koji zadaje vrijednosti u zavisnosti od potreba sistema.

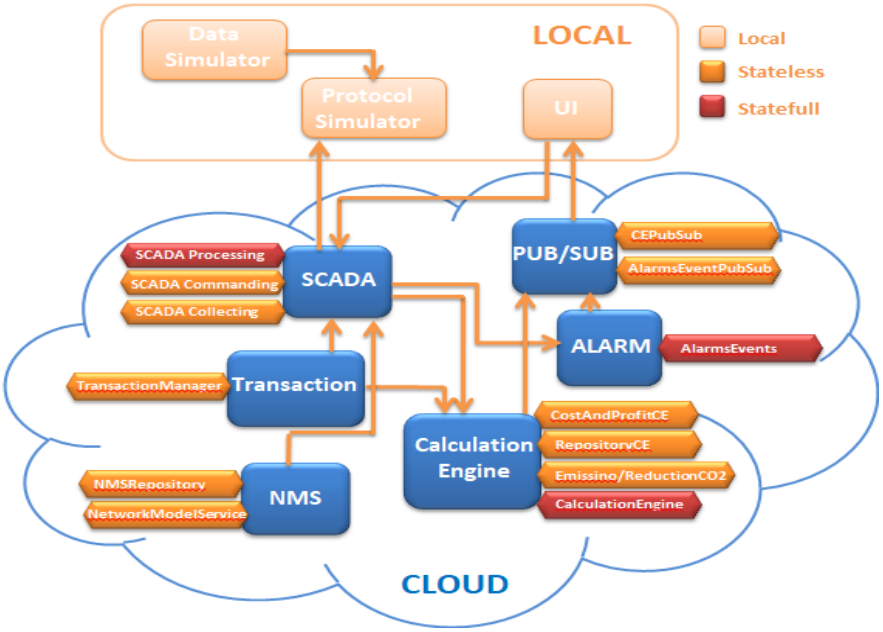
Alarms Events Microservice Procesirane vrijednosti sa SCADA Processing mikroservisa se šalju na Alarms Event mikroservis kako bi se provjerilo da li su vrijednosti u opsegu alarmnih vrijednosti. Ukoliko se desi alarm na nekoj od tačaka, ovaj mikroservis upisuje podatke u bazu podataka u Azure Table Storage.

Alarms Events PubSub Microservice Publish – Subscribe patern je iskorišćen i u komunikaciji Alarm Service-a sa UI-om. Uz pomoć ovog mikroservisa UI dobija sva alarmna stanja analognih i diskretnih signala, kako bi se omogućio korisniku uvid u iste.

Transaction Manager Microservice je servis koji kontroliše koordinaciju transakcije (promjena NMS modela) i odgovoran je za stvaranje jednog ili više transakcionih objekata i njihovo prosleđivanje na servise kako bi se očuvala konzistentnost modela.

NMS Repository Microservice je servis koji je zadužen za komunikaciju sa bazom, tj. služi za čitanje i upisivanje Delte u Azure Blob Storage.

Network Model Microservice je servis koji je isključivo zadužen za rad sa NMS modelom, vrši učitavanje i konverziju modela, kao i prosleđivanje konvertovanog modela ostalim servisima.



Slika 1. *Arhitektura rješenja*

5.2 Service Fabric Explorer

Service Fabric Explorer (SFX) je open-source alat koji omogućava pregled, razvoj i upravljanje Azure Service Fabric klasterima.

Pokretanje SFX u bilo kom web pretraživaču ostvaruje se putem linka https://localhost:19080/Explorer. ServiceFabric ima dva mode-a simulaciju sa jednim ili pet čvorova. Ukoliko se koristi mode sa pet čvorova, moguće je deaktivirati pojedine čvorove. Omogućeno je ponovno pokretanje čvorova ili njihovo deaktiviranje pa se na taj način može lako testirati failover. Za testiranje većeg broja mikroservisa, simulator troši dosta procesorke moći i radne memorije. Praćenje stanja servisa i upravljanje njima vrši se kroz SFX, a u njemu je omogućen i prikaz svih aplikacija i node-ova, kao i pracnje zdravlja istih.

5.3 Testiranje

Aplikacija je testirana za brzinu izvršavanja genetskog algoritma prije i poslije njene implementacije u Cloud okruženju. Na samu brzinu izvršavanja algoritma utiču vrijednosti svih ulaznih parametara, a to su: broj iteracija, broj populacija, elitism i mutation rate. Default-ne vrijednosti su ostavljene prilikom testiranja: broj iteracija= 200, broj populacija = 100, elitism = 5, mutation rate = 1. Sve ove vrijednosti je moguće komandovati i preko UI.

Opšti koraci genetskog algoritma su prikazani na slici 2. Teorijski postoje različiti uslovi zaustavljanja genetskog algoritma, u ovom projektu taj *uslov* je dok se ne postigne određen broj iteracija tj. 200.

```
Genetski algoritam{

        gererisanje pocetne populacije P;
        za svako rjesenje iz P izracunati osobine;

        ponavljaj(uslov)
            {
                Odabir roditelja iz P;
                dijete = ukrstanjeGena(roditelj1,roditelj2);
                mutacija(dijete);
                izracunavanje osobine djeteta;
            }
        Ispisi rjesenje;
}
```

Slika 1. *Genetski algoritam*

Tabela 1. *Rezultati testiranja*

Br. klijenata	Monolitna aplikacija [s]	Aplikacija u *Cloud* okruženju [s]
10	1	1
50	2	1
100	3	2
300	9	6
600	19	12
1000	33	21

U Tabeli 1 prikazana su vremena izvršavanja akcije. Prva kolona predstavlja broj klijenata koji zahtjevaju postizanje elergetskog balansa između proizvedene i potrošene energije, koji se dobija putem genetskog algoritma. U drugoj koloni se nalaze vremena izvršavanja akcije za monolitnu aplikaciju, a u trećoj koloni vremena vezana za aplikaciju implementiranu u Cloud okruženju.

Vrijednosti u tabeli predstavljene su u sekundama i zaokružene su na cjelobrojne vrijednosti. Iz tabele se jasno može zaključiti da su vremena izvršavanja genetskog algoritma za mali broj korisnika približna, dok se sa porastom broja klijenata vrijednosti međusobno razlikuju. Sa povećanjem broja korisnika, monolitnoj aplikaciji treba više vremena za izvršenje u odnosu na aplikaciju implementiranu u Cloud okruženju i na osnovu toga se može zaključiti da je rad monolitne aplikacije sporiji u odnosu na aplikaciju koja je implementirana u Cloud okruženju.

6. ZAKLJUČAK

Zbog pojave nekih vrsta problema poput: opadanja performansi, stabilnosti sistema i sporog razvojnog ciklusa, mikroservisa arhitektura je bolji odabir u odnosu na tradicionalnu monolitnu arhitekturu. U mikroservisnoj arhitekturi promjene na jednom dijelu sistema ne utiču na neki drugi dio sistema iako nisu logički povezani.

Veliki problem nastaje kada uvođenje bilo kakave promjene u aplikaciju počne da oduzima previše vremena.

Mikroservisna arhitektura omogućava pouzdanost, skalabilnost i proširivost na brži i jednostavniji način razbijanjem aplikacije na mikroservise kao zasebne cjeline.

7. LITERATURA

[1] https://www.c-sharpcorner.com/article/understand-azure-service-fabric-in-three-minutes/
[2] https://docs.microsoft.com/en-us/azure/storage/common/storage-introduction
[3] https://docs.microsoft.com/en-us/azure/architecture/guide/architecture-styles/microservices
[4] https://www.xenonstack.com/insights/stateful-and-stateless-applications
[5] https://docs.microsoft.com/en-us/azure/service-fabric/service-fabric-visualizing-your-cluster

Kratka biografija:

Snežana Dupljanin rođena je 27.02.1997. godine u Sokocu. Diplomirala je na Fakultetu tehničkih nauka iz oblasti Elektrotehnike i računarstva – Primenjeno softversko inženjerstvo 2019. godine.