Зборник радова Факултета техничких наука, Нови Сад

UDK: 004.41
DOI: https://doi.org/10.24867/08BE02Stevanovic

ПРИМЕНА *SERVERLESS* АРХИТЕКТУРЕ НА АПЛИКАЦИЈУ ЗА УПРАВЉАЊЕ РАДОВИМА У ЕЕ СИСТЕМУ

THE APPLICATION OF SERVERLESS ARCHITECTURE TO THE APPLICATION FOR MANAGEMENT OF WORKS IN EE SYSTEM

Немања Стевановић, *Факултет техничких наука, Нови Сад*

Област – ЕЛЕКТРОТЕХНИКА И РАЧУНАРСТВО

Кратак садржај – *У овом раду представљена је примена Serverless архитектуре на апликацију која симулира управљање радовима у електроенергетском систему. Приказан је начин на који се може смањити оптерећење једног од сервиса апликације употребом Azure Functions које представљају Serverless архитектуру креиране од стране Microsoft Azure-a.*

Abstract – *This paper presents the application of Serverless architecture to the application which simulates management of works in electric power system. It shows way that can reduce load of one application service using Azure Functions which represents Serverless arhitecture made by Microsoft Azure.*

Кључне речи: Cloud, Serverless, Azure Functions

1. УВОД

Електричне мреже представљају један од најзначајнијих делова електроенергетског система. Помоћу њих се обезбеђује непрекидно напајање потрошача електричном енергијом. Колико год да је електрична мрежа отпорна, увек постоји могућност њеног квара. Због тога, оператери који врше надгледање мреже морају имати могућност извршавања планираних или непланираних радова на мрежи помоћу својих екипа. Како би се олакшао посао оператерима креирана је апликација која врши управљање радовима. Апликација је базирана на примени информационих технологија, *Smart Grid* система и *Cloud* технологија.

Идеја овог рада је да се постигне смањење оптерећења сервиса апликације који је задужен за креирање и управља планираним радовима. Циљ је да се упозна апликација и окружење у којем ради, као и да се упозна *Azure Functions*, начин њиховог рада, начин њихове интерграције са апликацијом, као и предности и мане коришћења ових функција као решење. Како се читав рад базира на употреби *Cloud* окружења, укратко ће бити описани нивои услуга које нам *Cloud* пружа као и *Service Fabric* окружење у којем сама апликација и ради.

НАПОМЕНА:

Овај рад проистекао је из мастер рада чији ментор је био др Срђан Вукмировић, ванр. проф.

2. ТЕОРИЈСКЕ ОСНОВЕ

2.1 *Smart Grid* системи

Smart Grid уводи значајне промене у начину на који се производи, преноси, дистрибуира и троши електрична енергија. Тачна дефиниција *Smart Grid*-а не постоји, већ се он посматра као интегрисани скуп технологија који смањује трошкове, повећава социјалну корист и смањује загађење животне средине. Под интегрисаним скупом технологија, *Smart Grid* подразмева повезивање електричне и комуникационе инфраструктуре са аутоматизованим постројењима и информационим технологијама у једну целину како би та целина пружила максималну корист. Циљеви којима *Smart Grid* тежи су следећи:

- *Интелигентна мрежа* која има могућност идентификације кварова и аутоматску минимизацију њихових утицаја
- *Ефикаснија мрежа* која је у стању да прихвати све веће захтеве за енергијом, а истовремено се фокусира на квалитет енергије
- *Прилагодљива мрежа* која прима енергију из свих извора и дозвољава интеграцију нових технологија
- *Отпорна мрежа* је отпорна на климатске катастрофе и терористичке нападе
- *Мотивишућа мрежа* нуди могућност реалтиме комуникације кроз цео систем
- *Зелена мрежа* омогућава значајно унапређење у процесу очувања животне средине
- *Мрежа нових шанси* дозвољава креирање нових типова послова, фирми, клијената, опреме итд

2.2 *CLOUD*

Cloud се односи на дељење ресурса, софтвера и информација путем мреже. Информације и подаци се чувају на физичким или виртуалним серверима које одржава и контролише *Cloud* провајдер. Коришћење *Cloud*-а се ослања на исказ "*pay as you go*". Ово значи да, ако су нам у неком тренутку потребни додатни ресурси, ми их нећемо купити већ ћемо их изнајмити од провајдера, такође, ако нам ти ресурси више не буду били потребни, отказаћемо њихову употребу и више нећемо плаћати за њих. Приликом коришћења *Cloud*-а, обавезе провајдера и клијента се описују преко нивоа услуга које клијент изнајмљује. Три главна нивоа услуга су:

- *Инфраструктура као сервис (IaaS)* - састоји се од високо аутоматизованих и скалабилних рачунарских ресурса који су употпуњени могућностима складиштења и повезивања на мрежу. *IaaS* омугућава предузећима да изнајмљују хардвер на захтев и по потреби уместо да га директно купују.
- *Платформа као сервис (PaaS)* - обезбеђује платформу која омогућава клијентима да развијају, покрећу и управљају апликацијама без сложених проблема изградње и одржавања инфраструктуре која је повезана са развојем апликација.
- *Софтвер као сервис (SaaS)* - представља најчешће коришћен ниво услуга где нуди готов производ крајњим клијентима. Клијенти уопште не воде рачуна о развоју и управљању апликација. Приступна тачка овим апликацијама се махом нуди преко web претраживача.

Поред ових нивоа услуга, због саме теме рада, ми ћемо навести и *функцију као сервис (FaaS)* која представља *Serverless* начин за изградњу дела или читаве апликације. Клијент пише и ажурира парче кода које се покреће на одређени догађај. *FaaS* се као ниво услуга сврстава на врху *PaaS* нивоа.

2.3 *Service Fabric*

Service Fabric представља платформу дистрибуираних система који олакшавају инжењерима рад јер они могу остати усредсређени на саме захтеве апликација. *Service Fabric* нуди могућност управљања скалабилности и поузданости микросервиса који ће се користити у апликацији. На овај начин се тежи ка преласку са монолитних апликација на микросервисне. Микросервисну апликацију описују сервиси где се сваки сервис бави једном бизнис логиком. На овај начин добијамо разне предности, као на пример: сваки сервис се може посматрати као независна компонента, лакше се врши тестирање над сервисима који се баве само једном логиком, као што је овде случај, над сервисима могу радити мањи тимови инжењера итд. Треба споменути да пребацивањем са монолитне апликације на микросервисну постоје и могуће мане као што је случај са повећаном комуникацијом како би сервиси могли да размењују податке међусобно, а те комуникације можда нису ни постојале у монолитној апликацији. *Service Fabric* нам даје могућност креирања *stateless* и *stateful* сервиса. Разлика је у томе да ли у тим сервисима постоји неко стање које треба да се сачува, ако дође до пада сервиса, или не. Предност *stateful* сервиса је у томе што перзистирају своја стања кроз *reliable* колекције [1] помоћу којих ће имати брз приступ подацима, док је предност *stateless* сервиса у томе што су погодни за скалирање у случају већег оптерећења.

2.4 *Serverless* архитектура

Serverless представља архитектуру извршавања где *Cloud* провајдер обезбеђује подизање сервиса и руковање ресурсима. Иако носи назив *serverless* (без сервиса) у позадини се подиже сервис који ће извршавати код. Назив је добио по томе што клијенти не плаћају провајдеру сервис већ само ресурсе који су утрошени за време извршавања његовог кода. Микросервисна архитектура се развила како би се логика велике апликације разводијала и олакшао развој јер се ради на мањим целинама. Таквом архитектуром смо имали предност јер смо могли да скалирамо само онај сервис који трпи веће оптерећење док друге сервисе нисмо морали да дирамо. *Serverless* архитектура је отишла и корак напред где ми добијамо могућност скалирања само једног дела једног сервиса. *Serverless* апликације су *event-driven* апликације. То значи да се њихово покретање ослања на неки догађај.

Провајдери који нуде коришћење *Serverless* оркужења цену одређују преко изабраних планова под којим ће се *Serverless* користити. Сваки од тих планова описује шта нуди, која су му ограничења и на који начин се формира цена. Неке од предности *Serverless*-а су цена где клијент плаћа само за потрошене ресурсе, еластичност и скалабилност где ће провајдер да води рачуна о овим концептима и продуктивност где су јединице кода спољашњем свету изложене као једноставне функције, па тако нпр. клијенти не морају да воде рачуна о *multithreading*-u.

Мане које *Serverless* носи са собом су те што клијент нема могућност избора машине на којој ће се његова функција покренути и ограничења која се могу појавити употребом коришћеног плана.

3. СИСТЕМ УПРАВЉАЊА РАДОВИМА

Као што је у уводу већ наведено, систем управљања радовима представља софтверски производ који се заснива на примени *Smart Grid* система и *Cloud* технологија. Апликација је направљена помоћу програмског језика C# и извршава се у *Service Fabric Cloud* окружењу. Оператери помоћу ове апликације имају могућност да преко графичког приказа мреже посматрају тренутно стање, као и могућност измена стања мреже. Под изменама стања мреже се подразумева ручно мењање стања прекидача као и креирање планираних радова над жељеним секцијама мреже у одређеном периоду.

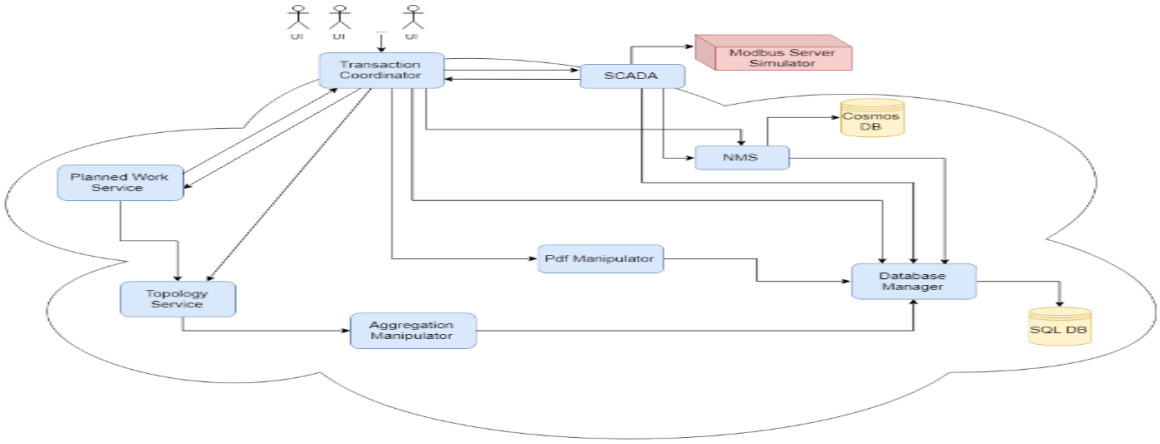
Приликом промене страња мреже мења се и њена енергизација, што се рефлектује и на графичком приказу за кориснике. Систем води рачуна о томе да приликом креирања и извршења планираних радова не дође до преклапања рада над истим секцијама као и да не дође до инцидентних ситуација. Такође, приликом креирања планираног рада систем алгоритамски одређује и генерише оптималне инструкције за његову реализацију. Циљ приликом креирања планираних радова јесте да што мање корисника остане без напајања. Планирани радови се могу извршити и пре заказаног времена уколико су испуњени безбедносни услови. У супротном се планирани рад извршава аутоматски у заказно време. Разлози за вршење планираних радова су потреба за ремонтом одређених делова мреже прилком квара или одржавање мреже, као и координација са осталим радовима на мрежи.

Предности система за управљање планираним радовима су следеће:

- Већа сигурност радника на терену, као и опреме
- Брз увид у стање мреже
- Могућност управљања на даљину из диспечерског центра
- Смањење трошкова управљања
- Лакше прикупљање статистичких података о испадима корисника
- Генерисање извештаја наспрам унетих параметара и на основу прикупљених података

3.1 Архитектура система

Систем се састоји од осам сервиса и две базе података које они користе. Уз то, *SCADA* сервис користи *Modbus Server Simulator* за симулацију *RTU* уређаја. Архитектура решења је приказана на слици 3.1.1. Како је апликација подигнута у *Service Fabric*-у, NMS, Topology и Planned Work сервис су изграђени као *stateful* сервиси док су остали *stateless*.



Слика 3.1.1 *Архитектура система*

За даље разумевање рада можемо још рећи да ће се план рада који клијент креира проследити до Transaction Coordinator сервиса који имплементира *API Gateway* образац [2] и да ће он након тога проследити план Planned Work сервису који ће у даљем раду бити повезан са *Serverless* архитектуром.

4. ОПИС ПРОБЛЕМА

Како ова апликација и носи назив, систем за управљање радовима, очекивано је да њен фокус буде на креирању и извршавању тих радова. За ово ће бити задужен Planned Work сервис који поред руковања акција које долазе са клијентске стране, има имплементирану и аутоматику у себи која ће, такође, радити над планираним радовима. Због овога ће Planned Work сервис трпети највише оптерећења или ће макар бити један од сервиса који трпе највише оптерећења. Пошто је овај сервис изграђен као *stateful* сервис, код њега скалирање није лако као код *stateless* сервиса. *Service Fabric* нуди решење и за скалирање *stateful* сервиса, али оно подразумева његово партиционисање. На овај начин, сервис би имао више партиција где би свака партиција била одговорна за део података. Мана овог приступа је што сервис који позива партиције мора да имплементира логику партиционисања да би могао да погоди жељену партицију. На овај начин се ствара зависност између сервиса, а у случају лошијег плана партиционисања, гађане партиције неће имати равномерно оптерећење и опет постоји могућност да нека партиција трпи превише оптерећења.

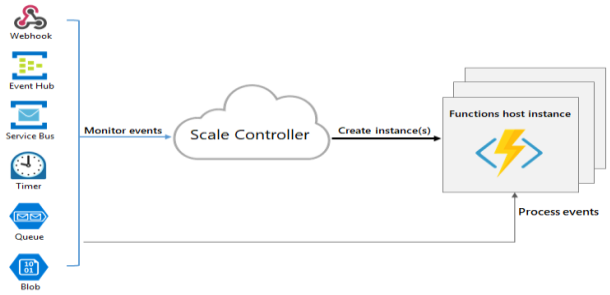
5. РЕШЕЊЕ ПРОБЛЕМА

За решење претходно описаног проблема је изабрана *Severless* архитектура коју је креирао *Microsoft Azure* преко својих *Azure Functions*. Циљ је да се креирање планова обавља преко *Azure Functions* како би се смањило процесорско оптерећење машине на којој ће бити покренут Planned Work сервис у замену за нешто дуже извршавање клијентског захтева.

5.1 Опис *Azure Functions*

Azure Functions се јавља у своја два облика. Један је у виду *Durable* функција, док је други обичне *Azure* функције. Разлика је што *Durable* представљају *stateful* функције што их чине нешто споријим од обичних па смо због тога ми користили обичне. Као догађај на који ће се функција позивати изабрали смо *HTTP* окидач. Функција се састоји из два дела где је први део њено заглавље док је други тело функције које представља њену имплементацију. У заглављу је постављена рута *HTTP* захтева, тип захтева који је постављен на „*post*“ и ауторизациони ниво. Због ауторизационог нивоа, свако ко жели да погоди функцију мораће да поседује кључ. Поред функције, креира се и њена конфигурација где се постављају параметри за њен рад. Овим параметрима смо поставили ниво њеног логовања као и лимит за њено конкурентно извршавање. Свака функција се покреће из два могућа стања. Прво стање је *cold*, а друго је *warm*. Функција се покреће из *cold* стања ако се не користи одређено време или ако дође до оптерећења што ће изазвати њено скалирање, где ће се нове инстанце, такође, покренути из *cold* стања. Овим се повећава латенција што утиче на перформансе. Са друге стране, *warm* стање подразумева да је функција у могућнпсти да се одмах покрене и изврши свој посао. О скалирању функција води рачуна *Scale Controller* који је приказан на слици 5.1.1. Скалирање може бити у размаку између 0 и 200 инстанци, где се може додавати највише једна инстанца по секунди. Као план над којим ће се покретати функција изабран је *Consumption* план. По овом плану на цену утичу

број извршавања, време извршавања и искоришћена меморија. Ограничења које он поставља су да извршавање функције не може да буде дуже од 10 минута, постоји *cold* стање, највише једно процесорко језгро се користи у току рада и максимална искоришћена меморија је до 1.5GB.



Слика 5.1.1. *Scalle Controler*

5.2 Имплементација решења

Имплементација решења је обухватала имплементацију саме функције и њено повезивање на Planned Work сервис. Тело функције се састоји из два корака где први корак обухвата десеријализацију пристиглих података, а други корак обухвата коришћење алгоритма за креирање планова. Функција ће вратити грешку у случају невалидних података или у случају покушаја неауторизованог приступа.

Planned Work сервис је морао да имплементира логику којом ће се одлучивати када ће се звати функција, а када ће извршити локално креирање плана. Коришћена су два параметара за ово и то лимит позива који се може мењати у конфигурацији и број извршених позива у сату који ће се чувати у *reliable* колекцији. У случају да лимит није достигнут, позиваће се функција, ако је број позива једнак лимиту, сервис ће послати *email* администратору и остале захтеве за креирањем планова у току тренутног сата ће извршавати локално. Planned Work сервис ће у случају позивања функције одрадити то у три корака. Први корак је припремање садржаја за слање што обухвата серијализацију података. Други корак представља слање *HTTP* захтева који у заглављу садржу кључ ауторизације и у телу садржи припремљене податке. Трећи корак обухвата логику након позива функције која зависи од пристиглог одговора. Последњи корак води рачуна и о могућим грешкама које могу настати у току слања захтева.

За слање захтева преко *HTTP*-а, изабрана је његова верзија 2. У односу на претходну верзију 1.1 која је била заступљена 15 година, а и даље је у употреби, ова верзија има могућност знатно бржег слања. Нова верзија је увела доста новина, али две најзачајније за наш пројекат су: мултиплексирање захтева и компресија заглавља. Мултиплексирање захтева представља начин слања више захтева кроз једну *TCP* конекцију. Ово нарочито може да утиче на перформансе апликације. Како је *HTTP* протокол *stateless*, то значи да сваки захтев мора да носи онолико података колико је потребно серверу да би он могао да изврши жељену акцију. Овим механизмом се проузрокује слање оквира информација који се понављају у захтевима. *HTTP*/2 користи *HPACK* спецификацију за једноставну и сигурну компресију заглавља којом се избегава наведени проблем.

6. ЗАКЉУЧАК

У овом раду је представљено једно од могућих решења за постизање смањења оптерећења неког сервиса. Описана је *Serverless* архитектура као и њена употреба преко *Azure Functions*. Одрађени тестови су показали да употребом *Azure Functions* заиста долази до смањења оптерећења на Planned Work сервису и да се латенција код клијентског позива није много повећала.

7. ЛИТЕРАТУРА

[1] *Programming Microsoft Azure Service Fabric, Second Edition*, Haishi Bai

[2] *Microservicess Patterns*, Chris Richardson

Кратка биографија

Немања Стевановић рођен је 09.06.1995. год. у Смедереву. Завршио је економску школу у Смедереву 2014. год. Исте године је уписао основне академске студије на Факултету техничких наука у Новом Саду. Дипломски рад из области Електротехника и рачунарства је одбранио 2018. године, након чега је исте године уписао мастер академске студије. Испунио је све обавезе и положио све испите предвиђене студијским програмом.